


Space Details

Key:	WW
Name:	WebWork
Description:	
Creator (Creation Date):	plightbo (Apr 18, 2004)
Last Modifier (Mod. Date):	matthew (Feb 08, 2005)

Available Pages

- WebWork 
 - Documentation
 - Cookbook
 - Access to Webwork objects from JSP 2.0 EL
 - Accessing application, session, request objects
 - App Servers
 - WebLogic
 - WebLogic 6.1
 - Application, Session, Request objects in jsp
 - Application, Session, Request objects in vm
 - Describing a bean in velocity
 - Exposing webwork objects to JSTL, with a JSTL and DisplayTag Example
 - File Upload Interceptor
 - GroovyResult
 - Handling IoC Components to Interceptors and Validators
 - How do I populate a form bean and get the value using the taglib
 - How to format dates and numbers
 - How to validate field formats, such as a phone number
 - Interceptor Order
 - Iterator tag examples
 - JFreeChartResult
 - Tabular inputs with XWorkList
 - Transparent web-app I18N
 - Using Checkboxes
 - Using Checkboxes - EditAction.java
 - Using Checkboxes - User.java
 - Using Checkboxes - Velocity and HTML
 - Using WebWork and XWork with JSP 2.0 and JSTL 1.1
 - Using WebWork Components
 - Value Stack Internals
 - Webwork 2 HTML form buttons Howto
 - Webwork 2 skinning

- Webwork file upload handling
- Webwork reference to OGNL access
- WebworkVelocity and Sitemesh velocity combined
- FAQ
 - How do I get the latest version of WebWork
- Overview
 - Articles and press
 - Strutting the OpenSymphony way
 - Comparison to other web frameworks
 - Comparison to JSF
 - Comparison to Ruby on Rails
 - Comparison to Spring MVC
 - Comparison to Struts
 - Comparison to Tapestry
 - Projects Using WebWork
 - Testimonials
 - What is WebWork
- Project Information
 - Dependencies
 - Deployment Notes
 - Previous releases
 - Release Notes - 2.1
 - Release Notes - 2.1.1
 - Release Notes - 2.1.2
 - Release Notes - 2.1.3
 - Release Notes - 2.1.4
 - Release Notes - 2.1.5
 - Release Notes - 2.1.6
 - Upgrading from 1.4
 - JSP Expression Language Comparison with WebWork 1.x
 - Upgrading from 2.0
 - Upgrading from 2.1
 - Upgrading from 2.1.1
 - Upgrading from 2.1.2
 - Upgrading from 2.1.3
 - Upgrading from 2.1.4
 - Upgrading from 2.1.5
 - WebWork 2.1.7
 - WebWork 2.2
 - WebWork 2.2 Migration Notes
 - WebWork 2.3
- Reference
 - 3rd Party Integration
 - Hibernate
 - JSTL

- JUnit
- Pico
- Quartz
- SiteMesh
- Spring
 - Other Spring Integration
- Action Chaining
- Action Configuration
 - Interceptor Configuration
 - Namespace Configuration
 - Package Configuration
 - Result Configuration
 - Default results
 - Global results
- ActionMapper
- Architecture
- Configuration
 - Reloading configuration
 - velocity.properties
 - web.xml
 - web.xml 2.1.x compatibility
 - webwork-default.xml
 - webwork.properties
 - xwork.xml
- Continuations
- FreeMarker
- Interceptors
 - Alias Interceptor
 - Chaining Interceptor
 - Component Interceptor
 - Conversion Error Interceptor
 - Exception Interceptor
 - Execute and Wait Interceptor
 - HibernateAndSpringEnabledExecuteAndWaitInterceptor
 - I18n Interceptor
 - Logger Interceptor
 - Model Driven Interceptor
 - Parameters Interceptor
 - Prepare Interceptor
 - Scope Interceptor
 - Servlet Config Interceptor
 - Static Parameters Interceptor
 - Timer Interceptor
 - Token Interceptor
 - Token Session Interceptor

- Validation Interceptor
 - Workflow Interceptor
- Internationalization
- Inversion of Control
 - Components
 - IoC Configuration
 - IoC Overview
 - Xwork's Component Architecture
- J2SE 5 Support
- JasperReports
- JSP
- OGNL
 - OGNL Basics
- Related Tools
 - QuickStart
 - SiteGraph
- Result Types
 - Action Chaining Result
 - Dispatcher Result
 - FreeMarker Result
 - WebWork Freemarker Support
 - HttpHeaders Result
 - JasperReports Result
 - Redirect Result
 - Stream Result
 - Velocity Result
 - Resources Available to Velocity Views
 - XSL Result
- Tags and UI Components
 - Common Tags
 - Control Tags
 - append
 - else
 - elseIf
 - generator
 - if
 - iterator
 - merge
 - sort
 - subset
 - Data Tags
 - action
 - bean
 - debug
 - i18n

- include
- param
- push
- set
- text
- url
- Form Tags
 - checkbox
 - checkboxlist
 - combobox
 - datepicker
 - doubleselect
 - file
 - form
 - Remote Form Validation
 - hidden
 - label
 - password
 - radio
 - select
 - submit
 - textarea
 - textfield
 - token
- FreeMarker Tags
- JSP Tags
- Non Form Tags
 - a
 - Configured for AJAX
 - component
 - div
 - panel
 - tabbedpane
 - tabbedPanel
 - table
- Non-UI Tags
 - URL tag
- Tag Syntax
 - altSyntax
- Themes and Templates
 - Templates
 - Themes
 - WebWork 2 UI Tag Guide
- UI Tags
 - Checkbox tag

- Checkboxlist tag
 - Combobox tag
 - Component tag
 - File tag
 - Form tag
 - Hidden tag
 - I18n tag
 - Label tag
 - Password tag
 - Radio tag
 - Select tag
 - Submit tag
 - Tabbedpane tag
 - Table tag
 - Text tag
 - Textarea tag
 - Textfield tag
 - Token tag
 - TokenInterceptor
 - Velocity Tags
 - Velocity Tags - Old
- Type Conversion
- Validation
 - Client-Side Validation
 - Simple validators
 - Validation Examples
 - VisitorFieldValidatorExample
 - Visitor validation
- Velocity
- Related Projects
 - EclipseWork
 - IDEA Plugin
 - Struts Ti
- Style Guide
- Tutorial
 - Getting Started
 - Lesson 1 - Setting up webwork in a web application
 - Lesson 2 - An html form with no data
 - Lesson 3 - An html form with data
 - Lesson 4 - An html form with data, without getters or setters
 - Quick Start Guide
 - TutorialLesson05
 - TutorialLesson04-01
 - TutorialLesson04-01-01
 - TutorialLesson04-02

- TutorialLesson04-03
- TutorialLesson06

WebWork

This page last changed on Oct 05, 2005 by [digi9ten](#).

Welcome to the **WebWork** wiki. WebWork's official homepage is <http://www.opensymphony.com/webwork/>. There you can find documentation for the latest released version of WebWork. This wiki is used for additional information as well as documentation for the latest developing version (see [Previous releases](#)).

- [Documentation](#)
 - [API JavaDocs](#)
- [Press Releases](#)
- [Download Binaries](#)
- [CVS](#)
- [Examples](#)
- [Meetings Minutes](#)
- [Misc](#)

Documentation

This page last changed on Oct 07, 2005 by [plightbo](#).

Note: for anyone contributing to the documentation, please read the [Style Guide](#) and make sure you follow it. Also, please look in to [JIRA](#) for a list of open items to do related to documentation (search for items in the Documentation component).

*If you're new to WebWork, please read the **Overview** and proceed to the **Tutorial** to get started. Experienced users can refer to the **Cookbook** for advanced topics. Use the **Reference** on an as-needed basis for more specific details. For detailed information about WebWork project, read the section **Project Information**. Information about many projects related to WebWork can be found in **Related Projects***

*If you have any questions, you can ask them at the user forum/ mailing list. Please be sure to read the **FAQ** before asking any questions.*

1. [Overview](#)
2. [Project Information](#)
3. [FAQ](#)
4. [Tutorial](#)
5. [Cookbook](#)
6. [Reference](#)
7. [Related Projects](#)

This page last changed on Oct 30, 2005 by [plightbo](#).



The cookbook currently contains a lot of information that may be out of date. These pages will be updated over time and this warning will eventually be removed when the WebWork team feels that the content is 100% correct.

Webwork Cookbook

Welcome to the Webwork Cookbook. This page is geared towards providing an exchange of information for developers. Your welcome to share knowledge and any helpful tips here.

[Deployment notes](#)

[App Servers](#)

[Accessing application, session, request objects](#)

[How to format dates and numbers](#)

[Iterator tag examples](#)

[Exposing webwork objects to JSTL, with a JSTL and DisplayTag Example](#)

[Value Stack Internals](#)

[Using WebWork Components](#)

[Webwork file upload handling](#)

[How do I populate a form bean and get the value using the taglib](#)

[Interceptor Order](#)

[Tabular inputs with XWorkList](#)

[Using WebWork and XWork with JSP 2.0 and JSTL 1.1](#)

[Webwork 2 skinning](#)

[Transparent web-app I18N](#)

[Webwork 2 HTML form buttons Howto](#)

[Using Checkboxes](#)

[JFreeChartResult](#)

[Webwork reference to OGNL access](#)

[Application, Session, Request objects in jsp](#)

[Application, Session, Request objects in vm](#)

[Describing a bean in velocity](#)

[File Upload Interceptor](#)

[Resources Available to Velocity Views](#)

[GroovyResult](#)

[How to validate field formats, such as a phone number](#)

Access to Webwork objects from JSP 2.0 EL

This page last changed on Aug 29, 2005 by [plightbo](#).

To access Webwork ValueStack from third party JSP taglibs you have to expose property values to JSP.

You can use Webwork2 tag `<ww:set/>` to set named parameter in a JSP page, request, session or application scope. Following example, sets a request scoped parameter 'a' to list of integers:

```
<ww:set name="'a'" value="{ 1, 2, 3, 4 }" scope="request"/>
```

After setting parameter, third party JSP taglibs can access variables, or you can use JSP 2.0 EL (Expression Language). This is convenient as short hand EL expression syntax

\$

Unknown macro: {expression}

can be used in a text or inside of tag attributes:

```
a[0] = ${a[0]}  
  
<sample:tag value="${a[1]}" />
```

In practice, you've got to expose a lot of different variables to make effective use of third party taglibs like `displaytag` or `wurfl`. This leads to a lot of `<ww:set/>` tags what made me investigate how to make access to ValueStack and OGNL more transparent.



Why can't we just replace EL with OGNL?

Unfortunately, it isn't that simple. I've tinkered with `JSPFactory.setDefault()` to wrap around `getPageContext()` and create `ExpressionEvaluator` that would use OGNL.

This works in practice, but code generated by Jasper2 doesn't call

	<p>JSPFactory.getPageContext().getExpressionEvaluator() but goes directly to static method that is hardwired to jakarta commons-el implementation.</p> <p>Even if it would work it wouldn't be <i>clean</i> as JSPFactory.setDefault() should only be called by JSP implementation.</p>
--	---

There is a simple, if not elegant, solution available in JSP 2.0 EL, for exposing ValueStack to OGNL. It is possible to create custom functions that can be called from EL expressions. Functions have to be 'public static' and specified in a TLD file. Just import TLD in a JSP file where you've want to use a function.

For example, you could access action properties by evaluating OGNL expression by a function 'vs' (for valuestack) in EL:

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

a[0] = ${x:vs('a[0]')}
a[0] * 4 = ${x:vs('a[0] * 4')}

Current action name: ${x:name()}
Top of ValueStack: ${x:top()}
```

To use this code you've got to add wwel.tld and Functions.java to your webapp project.

I would urge webworkers to define a set of functions that would be usable to wide community and include this in some future Webwork release.

wwel.tld
<pre><?xml version="1.0"?> <taglib xmlns="http://java.sun.com/xml/ns/j2ee"xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-jsptaglibrary_2_0.xsd" version="2.0"> <description> This taglib enables access to WebWork2 ValueStack</pre>

```

from JSP 2.0 Expression Language
</description><tlib-version>1.0</tlib-version><short-name>wwel</short-name><function><name>vs</name><function-class>com.nmote.wwel.Fun
    java.lang.Object findOnValueStack(java.lang.String)
</function-signature></function><function><name>name</name><function-class>com.nmote.wwel.Fun
    java.lang.Object getActionName()
</function-signature></function><function><name>top</name><function-class>com.nmote.wwel.Fun
    java.lang.Object getTopOfValueStack()
</function-signature></function></taglib>

```

Functions.java

```

package com.nmote.wwel;

import com.opensymphony.xwork.ActionContext;

/**
 * Utility functions for accessing webwork value stack and action context
 * from JSP 2.0 EL taglibs.
 *
 * @author Vjekoslav Nesek (vnesek@nmote.com)
 */
public class Functions {

    public static Object findOnValueStack(String expr) {
        ActionContext a = ActionContext.getContext();
        Object value = a.getValueStack().findValue(expr);
        return value;
    }

    public static Object getTopOfValueStack() {
        ActionContext a = ActionContext.getContext();
        Object value = a.getValueStack().peek();
        return value;
    }

    public static Object getActionName() {
        ActionContext a = ActionContext.getContext();
        Object value = a.getName();
        return value;
    }
}

```

Accessing application, session, request objects

This page last changed on Nov 30, 2004 by [jcarreira](#).

Webwork provides several access helpers to access Session, Application, Request scopes.

Web agnostic (independent of the servlet API) with calls:

```
Map session = (Map) ActionContext.getContext().get("session");
session.put("myId", myProp);
```

The following gives you the same thing as above:

```
ServletActionContext.getRequest().getSession()
```

Note: Be sure not to use `ActionContext.getContext()` in the constructor of your action since the values may not be set up already (returning null for `getSession()`).

Note also: `ActionContext.getContext().get("session")` is the same as `ActionContext.getContext().getSession()` with a cast to `Map`.

If you really need to get access to the `HttpSession`, use the `ServletConfigInterceptor` (see [Interceptors](#)).

In your views, you can access with your jsps as such

```
<ww: property value="#session.myId" />

<ww: property value="#request.myId" />
```

All the servlet scopes can be accessed like above.

```
Map request = (Map) ActionContext.getContext().get("request");
request.put("myId", myProp);
Map application = (Map) ActionContext.getContext().get("application");
application.put("myId", myProp);
Map session = (Map) ActionContext.getContext().get("attr");
attr.put("myId", myProp);
```

The 'attr' map will search the `javax.servlet.jsp.PageContext` for the specified key. If the `PageContext` doesn't exist, it will search request, session, application maps

respectively.

App Servers

This page last changed on Jun 06, 2005 by [plightbo](#).

- [WebLogic](#)
- [WebLogic 6.1](#)
- WebSphere
- JRun
- Jetty
- Tomcat/JBoss
- Resin
- Orion
- OC4J

```
> The classloaders of WLS seem not to play nice with velocity when
> deploying this way.
>
> If you haven't already tried, do the following, it makes it all work
> for us:
>
> 1) In the webwork.properties file (which should be in your
> WEB-INF/classes directory) put a line like this:
>
> webwork.velocity.configfile = my-velocity.properties
>
> 2) create a "my-velocity.properties" file under WEB-INF/classes and
> put into it the contents of the velocity.properties file that is in
> webwork's velocity-dep.jar
>
> 3) in your new "my-velocity.properties" file, find the section titled
> "T E M P L A T E L O A D E R S", and change this section to look like
> this:
>
> =====
> resource.loader = class
>
> file.resource.loader.description = Velocity File Resource Loader
> file.resource.loader.class =
> org.apache.velocity.runtime.resource.loader.FileResourceLoader
> file.resource.loader.path = .
> file.resource.loader.cache = false
> file.resource.loader.modificationCheckInterval = 2
>
> class.resource.loader.class =
> org.apache.velocity.runtime.resource.loader.ClasspathResourceLoader
> class.resource.loader.cache = true
> =====
>
```

```
> ... which straightens out the class resource loading problems (for us  
> at least).  
>  
> hope this helps,  
> james
```

Note: The "when deploying this way" comment above refers to deploying a war file (not expanded) into the deployment directory of WebLogic; with WL 8.x, this is typically at <bea_home>/user_projects/domains/mydomain/.

Running WebWork 2 on Weblogic Server 6.1

This document describes why WebWork 2 doesn't work "as-is" on Weblogic Server 6.1 and shows how to build an additional JAR that will fix the problems.

Note: the service pack of Weblogic Server 6.1 used is SP4.

The first part of this document describes the technical problems and the theoretical solution.

Why WebWork Doesn't Work

Weblogic 6.1 was published just prior to the finalization of the Servlet 2.3 specification. The incompatibility is that servlet filters and listeners in Weblogic 6.1 do not work with the 2.3 spec primarily because the servlet context is not retrieved in the same way. This causes virtually all filter initialization operations to fail with an `AbstractMethodError` exception.

How WebWork Is Modified

In Servlet 2.3, the servlet context is available from the session object; this is not true for Weblogic Server 6.1. Hence, filters and listeners must be modified to retrieve the servlet context from a different source; this is accomplished by retrieving the servlet context from the `FilterConfig` passed to the servlet filters during initialization.

However, the WebWork code cannot be modified to do this, because this will break the Servlet 2.3 specification. The goal is to leave the "original" WebWork modified so that

it is still Servlet 2.3 compatible, and then to add an additional JAR that "breaks" WebWork to work on Weblogic Server 6.1.

Hence, if you want to run WebWork under Servlet 2.3, the default, then simply build WebWork as usual.

But if you want to run WebWork under Servlet 2.3, you need to build the additional JAR and put it into your WAR file, and then modify your web.xml to use the new classes instead of the standard ones.

The standard WebWork has already been modified slightly to make the above effort possible:

1. RequestLifecycleFilter is modified to retrieve its servlet context from the method `getServletContext()`. This method, `getServletContext()`, is then implemented to return the servlet context from where it is available in Servlet 2.3: the session object. The logical operation is unchanged, but now subclasses can override `getServletContext()` to retrieve the servlet context from a different location as we'll see below.
2. SessionLifecycleListener is modified in the same way as RequestLifecycleFilter. The method, `getServletContext()`, is implemented to return the servlet context, in this case also from the session object. Again, subclasses can override the `getServletContext()` method to restore the servlet context from a different source. Again, this class's functionality is unchanged.

Now, in a separate project, the following classes are added and compiled into a separate JAR:

RequestLifecycleFilterCompatWeblogic61

This subclass of RequestLifecycleFilter simply overrides `getServletContext()` to retrieve the servlet context from the filter config, creates a singleton class, SessionContextSingleton, and assigns the servlet context to the singleton so that the listeners will have the ability to retrieve it.

SessionLifecycleListenerCompatWeblogic61

This subclass of `SessionLifecycleListener` simply overrides `getServletContext()` to retrieve the servlet context from the singleton created above.

FilterDispatcherCompatWeblogic61

Although the superclass of this class, `FilterDispatcher`, is commented out, this subclass retrieves the servlet context in the same way as `RequestLifecycleFilterCompatWeblogic61` in case it is ever resurrected. At this time, this class is unnecessary.

ServletContextSingleton

A singleton class whose sole purpose is to hold the servlet context so that listener classes have access to it.

Setting Up WebWork 2 to Run on Weblogic 6.1

Building Your Own Project

In the `web.xml` file, make the following class name substitutions:

Old Class Name	New Class Name
<code>RequestLifecycleFilter</code>	<code>RequestLifecycleFilterCompatWeblogic61</code>
<code>SessionLifecycleListener</code>	<code>SessionLifecycleListenerCompatWeblogic61</code>
<code>FilterDispatcher</code>	<code>FilterDispatcherCompatWeblogic61</code>

FAQ

I still get the `AbstractMethodError` Exception when Weblogic Server starts up. What am I doing wrong?

1. Check to see if a webwork-example.war is still lingering in your mydomain/applications folder and delete it if it is there.
2. See next FAQ question.

The server behavior seems like it is from a previous source code base; I can't debug it. What's the clue?

Sometimes BEA Weblogic Server doesn't "rebuild" its temporary files. Do the following to force the temporary files to rebuild:

1. Stop the server.
2. Delete the .wlnotdelete folder in mydomain/applications.
3. Restart the server.

Application, Session, Request objects in jsp

This page last changed on Nov 30, 2004 by [jcarreira](#).

The application, session and request objects are available from within ww tags in jsp wherever ognl can be evaluated. Use the #session syntax to get the object and access values by their keys using ['key'].

```
<ww:property value="#application\['foo'\]"/>
<ww:property value="#session\['baz'\]"/>
```

Conversely, if you would like to make webwork objects available to say the jsp/jstl request scope. The property tag can be used like this.

```
<ww:set name="jobz" value="jobs" scope="request" />
```

A full example below shows a webwork variable "jobs" being exposed as "jobz" and being used with jstl and the display tag.

[WW:Exposing webwork objects to JSTL, with a JSTL and DisplayTag Example](#)

Application, Session, Request objects in vm

This page last changed on Nov 30, 2004 by [jcarreira](#).

```
$req.session.servletContext.getAttribute(...)  
$req.session.getAttribute(...)  
$req.getAttribute(...)
```

To get parameters from the QueryString or from a POSTed form, do not use `getAttribute`, use:

```
$req.getParameter(...)
```

But that's quite obvious, since `$req` is the request object and we all know how it works.

Example:

`_test.jsp_`:

```
<html><head></head><body>  
<%  
session.setAttribute("sessionFoo", "sessionBar");  
session.getServletContext().setAttribute("applicationFoo", "applicationBar");  
%>  
  
<p>The following information should be available when sending the form below:  
  
<ul>  
  <li>Request parameter 'queryStringFoo' with value 'queryStringBar';</li>  
  <li>Request parameter 'formFoo' with value 'formBar';</li>  
  <li>Session attribute 'sessionFoo' with value 'sessionBar';</li>  
  <li>Application attribute 'applicationFoo' with value 'applicationBar'.</li>  
</ul>  
</p>  
  
<form action="test.vm?queryStringFoo=queryStringBar" method="post">  
<input type="hidden" name="formFoo" value="formBar">  
<p><input type="submit" value="Test!"></p>  
</form>  
</body></html>
```

`_test.vm_`:

```
<html><head></head><body>  
  
#set ($ses = $req.getSession())
```

```
#set ($app = $ses.getServletContext())

<p>applicationFoo = $!app.getAttribute("applicationFoo")
<code>(app.getAttribute("applicationFoo"))</code></p>
<p>sessionFoo = $!ses.getAttribute("sessionFoo")
<code>(ses.getAttribute("sessionFoo"))</code></p>
<p>formFoo = $!req.getParameter("formFoo")
<code>(req.getParameter("formFoo"))</code></p>
<p>querystringFoo = $!req.getParameter("querystringFoo")
<code>(req.getParameter("queryStringFoo"))</code></p>

</body></html>
```

Describing a bean in velocity

This page last changed on Dec 10, 2004 by [sutter2k](#).

The follow snippet might be useful during debugging to list the properties inside an arbitrary bean. Or for handing to a UI developer that use unaware of the getters/setters inside an object.

```
## prints out the property names for a bean
#macro (describeBean $name)
#set($bu = $webwork.bean("com.opensymphony.util.BeanUtils"))
  #foreach($propName in $bu.getPropertyNames($name))
    <li>$propName</li>
  #end
#end
```

i.e. assuming \$obj is a PersonObject that has properties(firstName, lastName, and zip).

```
#describeBean($obj)
```

would print

```
<li>firstName</li>
```

```
<li>lastName</li>
```

```
<li>zip</li>
```

One might also expand upon this to build a dynamic interface with via reflection. e.g.

```
$webwork.evaluate("$obj.${propName}")
```

Exposing webwork objects to JSTL, with a JSTL and DisplayTag Example

This page last changed on Nov 30, 2004 by jcarreira.

```
<ww:set name="jobz" value="jobs" scope="request" />
```

The full example below shows a webwork variable "jobs" being exposed as "jobz" to the request scope and being used with jstl and the display tag.

```
<%@ taglib uri="/WEB-INF/tlds/c.tld" prefix="c" %>
<%@ taglib uri="/WEB-INF/tlds/fmt.tld" prefix="fmt" %>
<%@ taglib uri="/WEB-INF/tlds/displaytag-el-12.tld" prefix="display" %>
<%@ taglib uri="/WEB-INF/tlds/webwork.tld" prefix="ww" %>

<ww:set name="jobz" value="jobs" scope="request" />

<h1><fmt:message key="title.listAllJobs"/></h1>
<display:table name="jobz" class="simple" id="row" >
  <display:column titleKey="label.global.actions" >
    <c:url var="viewurl" value="/viewJobDetail.action">
      <c:param name="name" value="{row.name}"/>
      <c:param name="groupName" value="{row.group}"/>
    </c:url>
    <c:url var="exeurl" value="/viewJobDetail.action">
      <c:param name="name" value="{row.name}"/>
      <c:param name="groupName" value="{row.group}"/>
      <c:param name="executeJobAction" value="execute"/>
    </c:url>
    <c:url var="editurl" value="/viewJobDetail.action">
      <c:param name="name" value="{row.name}"/>
      <c:param name="groupName" value="{row.group}"/>
      <c:param name="editAction" value="edit"/>
    </c:url>
    <a href='{c:out value="{viewurl}"/>}'><fmt:message key="label.global.view"/></a>
    |
    <a href='{c:out value="{editurl}"/>}'><fmt:message key="label.global.edit"/></a>
    |
    <a href='{c:out value="{exeurl}"/>}'><fmt:message
key="label.global.execute"/></a> &nbsp;
  </display:column>

  <display:column property="group" titleKey="label.job.group" sortable="true" />
  <display:column property="name" titleKey="label.job.name" sortable="true" />
  <display:column property="description" titleKey="label.job.description" />
  <display:column property="jobClass" titleKey="label.job.jobClass" sortable="true"
/>

</display:table>
```

Please note, at the time of this writing the "titleKey" attribute of the display tag's column tag is not yet released into a final version. It is a feature that is currently, only available through cvs.

File Upload Interceptor

This page last changed on Oct 24, 2005 by [plightbo](#).

Content pulled from external source. Click [here](#) to refresh.

Interceptor that is based off of MultiPartRequestWrapper, which is automatically applied for any request that includes a file. It adds the following parameters, where [File Name] is the name given to the file uploaded by the HTML form:

- [File Name] : File - the actual File
- [File Name]ContentType : String - the content type of the file
- [File Name]FileName : String - the actual name of the file uploaded (not the HTML name)

You can get access to these files by merely providing setters in your action that correspond to any of the three patterns above, such as setDocument(File document), setDocumentContentType(String contentType), etc.

This interceptor will add several field errors, assuming that the action implements ValidationAware. These error messages are based on several i18n values stored in webwork-messages.properties, a default i18n file processed for all i18n requests. You can override the text of these messages by providing text for the following keys:

- webwork.messages.error.uploading - a general error that occurs when the file could not be uploaded
- webwork.messages.error.file.too.large - occurs when the uploaded file is too large
- webwork.messages.error.content.type.not.allowed - occurs when the uploaded file does not match the expected content types specified

Parameters

Content pulled from external source. Click [here](#) to refresh.

- maximumSize (optional) - the maximum size (in bytes) that the interceptor will allow a file reference to be set on the action. Note, this is **not** related to the various properties found in webwork.properties.

- `allowedTypes` (optional) - a comma separated list of content types (ie: text/html) that the interceptor will allow a file reference to be set on the action.

Extending the Interceptor

Content pulled from external source. Click [here](#) to refresh.

You can extend this interceptor and override the `#acceptFile` method to provide more control over which files are supported and which are not.

Examples

Content pulled from external source. Click [here](#) to refresh.

```
<action name="someAction" class="com.examples.SomeAction"><interceptor-ref  
name="fileUpload"/><interceptor-ref name="basicStack"/><result  
name="success">good_result.ftl</result></action>
```

GroovyResult - Groovy scripts as a view

This is an attempt to create a Result type that uses Groovy (<http://groovy.codehaus.org>) files as a view. It exposes the current ActionContext to a groovy script. This doesn't really have much practical use, but it's fun nonetheless and shows how easy creating Webwork Results is. There is another Result (JFreeChartResult) in the [Cookbook](#)

Installation

Not much - just make sure you have Groovy in your classpath, and the antlr, asm-* and groovy jars available to your webapp.

Configuration

xwork.xml - result-types definitions

```
<result-types>
  <result-type name="groovy" class="myapp.webwork.extensions.GroovyResult"/>
</result-types>
```

xwork.xml - action definitions

```
<action name="MyAction" class="myapp.webwork.actions.MyAction">
  <result name="success" type="groovy">
    <param name="file">test.groovy</param>
  </result>
</action>
```

The result type takes one parameter (for now), namely 'file', which contains the name of the groovy script in our script directory.

Show me the code !

Here's the code of the actual GroovyResult. This is a verbose version, with a lot of error checking.

GroovyResult.java - source code

```
public class GroovyResult implements Result {

    publicfinalstaticString GROOVY_DIR_NAME = "groovy";

    privatefinalstatic Logger logger = Logger.getLogger(GroovyResult.class);
    //our groovy source file name
    privateString file;
    //a groovy shell
    private GroovyShell shell;
    //our parsed script
    private Script script;
    //the outputstream that will replace the 'out' in our groovy stream
    private OutputStream out;
    //directory containing groovy scripts
    privateString scriptDirectory;
    /*
     * (non-Javadoc)
     *
     * @see
     * com.opensymphony.xwork.Result#execute(com.opensymphony.xwork.ActionInvocation)
     */
    public void execute(ActionInvocation inv) {

        //check the scriptDirectory - if it doesn't exists, use the default one
        //WEBAPP + Groovy files directory
        if (scriptDirectory == null) {
            //not pretty, but this allows us to get the app root directory
            String base = ServletActionContext.getServletContext().getRealPath(
                "/");
            //iffor some reason (.war, apache connector, ..) we can't get the
            // base path
            if (base == null) {
                logger
                    .warn("Could not translate the virtual path \"/\\" to set the
                        default groovy script directory");
                return;
            }
            scriptDirectory = base + GROOVY_DIR_NAME;
            //issue a warning that this directory should NOT be world readable
            // !!
            logger
                .warn("Please make sure your script directory is NOT world
                    readable !");
        }

        // first of all, make sure our groovy file exists, is readable, and is
        // an actual file

        File groovyFile = new File(scriptDirectory, file);
```



```

        if (!groovyFile.exists()) {
            //log an error and return          logger.warn("Could not find
destination groovy file: "
            + groovyFile.getAbsolutePath());
            return;
        }
        if (!groovyFile.isFile()) {
            //log an error and return          logger.warn("Destination is not a
file: "
            + groovyFile.getAbsolutePath());
            return;
        }
        if (!groovyFile.canRead()) {
            //log an error and return          logger.warn("Can not read file: " +
groovyFile.getAbsolutePath());
            return;
        }

        if (logger.isDebugEnabled())
            logger.debug("File " + groovyFile.getPath()
                + " found, going to parse it ..");

        /*
        * Here we create a Binding object which we populate with the webwork
        * stack
        */
        Binding binding = new Binding();

        binding.setVariable("context", ActionContext.getContext());

        /*
        * We replace the standard OutputStream with our own, in thiscase the
        * OutputStream from our httpResponse
        */
        try {
            //the out will be stored in an OutputStream
            out = ServletActionContext.getResponse().getOutputStream();
        } catch (IOException e1) {
            logger.error("Could not open outputstream", e1);
        }
        if (out != null){
            binding.setVariable("out", out);
        }
        else {
            logger
                .warn("OutputStream not available, using defaultSystem.out
instead");
            binding.setVariable("out", System.out);
        }

        //create a new shell to parse and run our groovy file
        shell = new GroovyShell(binding);
        try {
            //try to parse the script - the returned script could be cached
for//performance improvent
            script = shell.parse(groovyFile);
        } catch (CompilationFailedException e) {
            logger.error("Could not parse groovy script", e);
            return;
        } catch (IOException e) {
            logger.error("Error reading groovy script", e);
            return;
        }

```

```

    }
    //the binding is set, now run the script
    Object result = script.run();

    if (logger.isDebugEnabled()) {
        logger.debug("Script " + groovyFile.getName()
            + " executed, and returned: " + result);
    }
    try {
        out.flush();
    } catch (IOException e2) {
        logger.error("Could not flush the outputstream", e2);
    }
}

/**
 * @return Returns the script.
 */
public Script getScript() {
    return script;
}

/**
 * @param file
 *          The file to set.
 */
public void setFile(String file) {
    this.file = file;
}

/**
 * @param out
 *          The out to set.
 */
public void setOut(OutputStream out) {
    this.out = out;
}

```

Explanation

The first part of the result is little more than:

- determining the script directory - defaults to MYWEBAPP/groovy/
- checking the file - make sure it exists, is readable, ..



Make sure the groovy scripts directory is not world readable !

The groovy part starts at:

```

Binding binding = new Binding();
binding.setVariable("context", ActionContext.getContext());

```

A Binding object allows us to 'bind' objects to a groovy script, so they can be used as variables. In this case, I took the ActionContext and exposed it as 'context'.

```
out = ServletActionContext.getResponse().getOutputStream();
...
binding.setVariable("out", out);
```

We also bind an OutputStream to the groovy script (as 'out') - it simply serves as a replacement for the standard System.out, so any printing goes directly to the http response outputstream.

```
shell = new GroovyShell(binding);
```

Next step; we create a GroovyShell, and pass our populated Binding to the constructor. Any script ran by this shell will have access to the passed variables (ActionContext and OutputStream).

```
script = shell.parse(groovyFile);
```

Before you can run a groovyFile, you need to parse it. Any syntax errors will be reported here - I also suggest adding a better error reporting in this case if you actually want to use this Result.

Upon successful parsing, a Script is returned (which could be cached if you want to increase performance) which will be run by our Shell.

```
Object result = script.run();
```

As a test, you might want to create a little 'groovy' script to test our Result.
test.groovy - a simple groovy script

```
for (item in context.contextMap){
    println "item: ${item}"
}
```

Place the test.groovy file in your groovy scripts directory. You should now see the result when you invoke MyAction.action in your browser.

Possible improvements are binding all objects on the stack so they become available to the groovy script, refactoring to an InputStream instead of a File, etc .. Comments welcome !

Reason for use

I have recently found the need for my Interceptors and Validators to be able to access Components - such as a Validators which is UserAware and checks the UserManager to see if the user exists. Or a Interceptor which is ApplicationAware and asks the ApplicationManager if it is setup yet - if not, then redirecting to a setup action instead.

Currently WebWork (at version 2.1.7) only supports component management of Action, but this can be changed quite easily - if you know where to look.

Extending the Object Factorys

WebWork uses a **com.opensymphony.xwork.ObjectFactory** object instance to generate the various objects that WebWork utilises - Validators, Interceptors, Actions, and Results for example. This is the object we are going to extend to add some of this functionality.

The methods **buildInterceptor** and **buildValidator** do what they say on the tin. I have overridden them to do the following:

```
public Interceptor buildInterceptor(InterceptorConfig ic, Map map) throws
ConfigurationException {
    Interceptor i = super.buildInterceptor(ic, map);
    cm.initializeObject(i);
    return i;
}

public Validator buildValidator(String string, Map map) throws Exception {
    Validator v = super.buildValidator(string, map);
    cm.initializeObject(v);
    return v;
}
```

Creating a Component Manager

The variable **cm** is a **ComponentManager**. As I am unsure of how to access the ComponentManager that is used in the ComponentInterceptor (or used when initializing Action objects), we have to create our own. As the ObjectFactory is a singleton the overhead of this is relatively minor, even though not ideal.

The ComponentManager is created in the constructor like this:

```
private static final Log log = LogFactory.getLog(ObjectFactory.class);

private ComponentConfiguration cc;
private ComponentManager cm;

public ObjectFactory() {
    super();
    cm = (ComponentManager) ActionContext.getContext().get(
        ComponentInterceptor.COMPONENT_MANAGER );

    if (cm == null) {
        cc = new ComponentConfiguration();
        InputStream configXml =
            Thread.currentThread().getContextClassLoader().getResourceAsStream("components.xml");
        try {
            cc.loadFromXml(configXml);
        } catch (Exception e) {
            log.info("No component.xml found. They test will continue without
initializing components.");
            cc = null;
        }
    }

    cm = new DefaultComponentManager();
    if (cc != null) {
        cc.configure(cm, "session");
        cc.configure(cm, "application");
        cc.configure(cm, "request");
    }
}
```

Using our new ObjectFactory

The ObjectFactory is a singleton which allows you to set the object it hands out. To do this I have chosen to override the **init** method of the **com.opensymphony.webwork.dispatcher.ServletDispatcher** class. The method

looks something like this:

```
public void init(ServletConfig servletConfig) throws ServletException {
    ObjectFactory.setObjectFactory( new planb.jobsite.xwork.ObjectFactory() );
    super.init(servletConfig);
}
```

Code Results

The following full files result from this article.

Object Factory

ObjectFactory.java

```
import com.opensymphony.xwork.interceptor.Interceptor;
import com.opensymphony.xwork.interceptor.component.ComponentManager;
import com.opensymphony.xwork.interceptor.component.DefaultComponentManager;
import com.opensymphony.xwork.interceptor.component.ComponentInterceptor;
import com.opensymphony.xwork.interceptor.component.ComponentConfiguration;
import com.opensymphony.xwork.config.entities.InterceptorConfig;
import com.opensymphony.xwork.config.ConfigurationException;
import com.opensymphony.xwork.validator.Validator;
import com.opensymphony.xwork.ActionContext;

import java.util.Map;
import java.io.InputStream;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;

public class ObjectFactory extends com.opensymphony.xwork.ObjectFactory {

    private static final Log log = LogFactory.getLog(ObjectFactory.class);

    private ComponentConfiguration cc;
    private ComponentManager cm;

    public ObjectFactory() {
        super();
        cm = (ComponentManager) ActionContext.getContext().get(
            ComponentInterceptor.COMPONENT_MANAGER );

        if (cm == null) {
            cc = new ComponentConfiguration();
            InputStream configXml =
                Thread.currentThread().getContextClassLoader().getResourceAsStream("components.xml");
            try {
                cc.loadFromXml(configXml);
            } catch (Exception e) {
                log.info("No component.xml found. They test will continue without");
            }
        }
    }
}
```

```

initializing components.");
        cc = null;
    }

    cm = new DefaultComponentManager();
    if (cc != null) {
        cc.configure(cm, "session");
        cc.configure(cm, "application");
        cc.configure(cm, "request");
    }
}

public Interceptor buildInterceptor(InterceptorConfig ic, Map map) throws
ConfigurationException {
    Interceptor i = super.buildInterceptor(ic, map);
    cm.initializeObject(i);
    return i;
}

public Validator buildValidator(String string, Map map) throws Exception {
    Validator v = super.buildValidator(string, map);
    cm.initializeObject(v);
    return v;
}
}

```

Servlet Dispatcher

ServletDispatcher.java

```

import com.opensymphony.xwork.ObjectFactory;

import javax.servlet.ServletConfig;
import javax.servlet.ServletException;

public class ServletDispatcher extends
com.opensymphony.webwork.dispatcher.ServletDispatcher {

    public void init(ServletConfig servletConfig) throws ServletException {
        ObjectFactory.setObjectFactory( new planb.jobsite.xwork.ObjectFactory() );
        super.init(servletConfig);
    }
}

```

web.xml

Replace the reference to the webwork ServletDispatcher to point to the above ServletDispatcher class.

Important Notes

You should find your Interceptors and Validators are now componentized just like Actions, however there are some important notes to be made.

Lifecycle Issues

Interceptors and Validators are both cached by webwork and reused instead of being reinstanciated - this will mean that you may experience issues with components outside of the application scope. As all of my Interceptor / Validator required components are in this scope, this isn't an issue to me.

One solution to this constraint would be to investigate how webwork caches its Interceptors and Validators, then check to see if the objects use session / request scoped components and cache accordingly. Maybe a thought for the guys planning the next release of webwork!

Conclusion

For now this concludes this article - feel free to add your ideas!

How do I populate a form bean and get the value using the taglib

This page last changed on Nov 30, 2004 by jcarreira.

First off, if you're coming from Struts, you may feel more comfortable using FormBeans instead of using the Action as your form bean. Be aware, though, that in Webwork you DO have the option of having the properties directly in the Action class. If you want to use a FormBean, here's an example:

```
public class TestAction extends ActionSupport {
    private TestBean myBean;

    public TestBean getMyBean() {
        return myBean;
    }

    public void setMyBean(TestBean myBean) {
        this.myBean = myBean;
    }

    protectedString doExecute() throws Exception {
        myBean = new TestBean();
        BeanUtil.setProperties(ActionContext.getContext().getParameters(), myBean);
        return SUCCESS;
    }
}
```

Then, in your success.jsp, which is mapped as the success result of TestAction in the views.properties or actions.xml (see the docs for how to configure actions and view mappings), you can do this:

```
<!-- This will call getMyBean() on your action and put it on the top of the value
stack -->
<webwork:property value="myBean">
<!-- This will call getName() on your TestBean and print it to the page -->
The name is: <webwork:property value="name"/>
</webwork:property>
```

This is a good way to do it if you have several parameters from the TestBean that you want to display, but, if you have just one, like in this case, it's probably better to do this:

```
<webwork:property value="myBean/name"/>
```

Which will call getMyBean.getName() and print that out to the page.

How to format dates and numbers

This page last changed on Jan 23, 2005 by [mgreer](#).

A frequently asked question is how best to display dates and numbers using a specified format. There are a number of approaches for this, the most naive of which would be to add a method to your action class to do the formatting for you. This method would take in a `Date` (or subclass) object as a parameter, and return a formatted `String`.

That approach however suffers from a number of flaws. For example, it is not i18n aware. The date format specified is rigid, and will not adapt to different locales easily (assuming you're not using a default formatter that is). It also clutters up your actions with code that has nothing to do with the action itself.

Instead, the recommended approach is to use Java's built-in date formatting features via use of the `webwork:text` tag.

The `webwork:text` tag should be used for all i18n values. It will look up the properties file for your action, and from that select the value for the key that you specify. This is best illustrated in an example:

```
<!-- display the number of items in a cart -->
<webwork:text name="'cart.items'" value0="cartItems" />
```

The above tag will work as follows. `value0` will result in a call to **`getCartItems()`** on your action class. The **`cart.items`** name is escaped, so it is treated as a literal key into the actions' properties file. Your `MyAction.properties` file will contain the following:

`cart.items=You have {0} items in your cart.`

Normal Java **`MessageFormat`** behaviour will correctly substitute `{0}` with the value obtained from `getCartItems`.

Needless to say, this can get a lot more elaborate, with the ability to specify both date and number formatting. Let us consider another example. The goal here is to display a greeting to the user, as well as the date of their last visit.

```
<webwork:text name="'last.visit'" value0="userName" value1="lastVisit(userName)" />
```

MyAction.java contains:

```
publicString getUsername() { ... };  
public Date getLastVisit(String userName) { ... };
```

Your **MyAction.properties** file will then contain:

last.visit=Welcome back {0}, your last visit was at {1,date,HH:mm dd-MM-yyyy}

As you can see, this is a very powerful mechanism and allows you to easily display numbers and dates using any formatting rules that Java allows.



value0 interface deprecated

The examples above pass in the values as:

```
<webwork:text name="'text.message'"  
value0="userName" />
```

These values should now (>2.1.7?) be passed as params:

```
<webwork:text name="'text.message'">  
  <webwork:param  
value="'userName'" />  
</webwork:text>
```



Some message format examples

Here are some examples of formatting in the properties file:

```
format.date = {0,date,MM/dd/yy}  
format.time = {0,date,MM/dd/yy ha}  
format.percent = {0,number,##0.00'%'}
```

```
format.money = {0,number,$##0.00}
```

How to validate field formats, such as a phone number

This page last changed on Sep 23, 2005 by [jhouse](#).

Validating the format of String fields for patterns (such as a phone number) is easy with StringRegexValidator (named "regex" in the default validator configuration).

Simply add the validator the field in question, and supply a regular expression to match it against.

```
<validators>
  <field name="phone">
    <field-validator type="regex">
      <param name="regex">\\([\\d][\\d][\\d])
[\\d][\\d][\\d]-[\\d][\\d][\\d][\\d]</param>
      <message>Phone number must be in the format (XXX) XXX-XXXX</message>
    </field-validator>
  </field>
</validators>
```

If your expression tests against alpha characters, you may be interested in the "caseSensitive" parameter of with Validator as well. It defaults to "true".

Interceptor Order

This page last changed on Nov 30, 2004 by [jcarreira](#).

Interceptors provide an excellent means to wrap before/after processing. The concept reduces code duplication (think AOP).

Order of interceptors...

```
<interceptor-stack name="xaStack">
  <interceptor-ref name="thisWillRunFirstInterceptor"/>
  <interceptor-ref name="thisWillRunNextInterceptor"/>
  <interceptor-ref name="followedByThisInterceptor"/>
  <interceptor-ref name="thisWillRunLastInterceptor"/>
</interceptor-stack>
```

Note that some interceptors will interrupt the stack/chain/flow... so the order is very important.

Interceptors implementing `com.opensymphony.xwork.interceptor.PreResultListener` will run after the Action executes its action method but before the Result executes

```
thisWillRunFirstInterceptor
  thisWillRunNextInterceptor
    followedByThisInterceptor
      thisWillRunLastInterceptor
        MyAction1
        MyAction2 (chain)
        MyPreResultListener
        MyResult (result)
      thisWillRunLastInterceptor
    followedByThisInterceptor
  thisWillRunNextInterceptor
thisWillRunFirstInterceptor
```

Iterator tag examples

This page last changed on Oct 23, 2005 by [digi9ten](#).

This follows on from [Iteration Tags](#) which you should read first, but beware of references to '0' and 'that'; what you really want in WW2 is 'top', as illustrated below. (I finally worked this out from the source code - hopefully this page means you won't have to.)

Referencing the current value

The simple examples print out values from the list using the property tag, which uses the value at the top of the stack by default:

```
Days:
<ul>
<ww:iterator value="days">
  <li><ww:property/>
</ww:iterator>
</ul>
```

But if you're doing anything other than print the value, you probably need to refer to it specifically. Do this:

```
Most days:
<ul>
<ww:iterator value="days">
  <ww:if test="top != 'Monday'">
    <li><ww:property/>
  </ww:if>
</ww:iterator>
</ul>
```

Iterating over a list of objects

```
<ww:iterator value="employees">
  <ww:property value="name"/> is the <ww:property value="jobTitle"/><br>
</ww:iterator>
```

For 'name' and 'jobTitle' you could be more explicit and write 'top.name' and 'top.jobTitle', as 'top' refers to the object on the top of the stack. It's not necessary here, but it is in the next example.

Iterating over a list of lists

```
<table>
  <ww:iterator value="grid">
    <tr>
      <ww:iterator value="top">
        <td><ww:property/></td>
      </ww:iterator>
    </tr>
  </ww:iterator>
</table>
```

The trick here is to use 'top' as the value for the inner iterator. This example probably uses a two-dimensional array, but you can use the pattern for any list of lists.

A more complex example

In this example, 'countries' is a list of country objects, each of which has a name and a list of cities. Each city has a name.

```
<ww:iterator value="countries">
  <ww:iterator value="cities">
    <ww:property value="name"/>, <ww:property value="[1].name"/><br>
  </ww:iterator>
</ww:iterator>
```

The output looks like

```
Wellington, New Zealand
Auckland, New Zealand
Moscow, Russia
Glasgow, Scotland
Edinburgh, Scotland
Stockholm, Sweden
```

Both the country and city objects have a 'name' property. As you'd expect, the reference to 'name' on its own gives you the city name. To access the country name - effectively "hidden" by the city name - we refer to a specific position on the stack: '1'. The top of the stack, position 0, contains the current city, pushed on by the inner iterator; position 1 contains the current country, pushed there by the outer iterator.

Actually, as Patrick points out in his comment on [Iteration Tags](#), the 'n' notation refers

to a sub-stack beginning at position n , not just the object at position n . Thus '0' is the whole stack and '1' is everything except the top object. In our example, we could have been more specific about getting the country name and said '1.top.name'.

Intro

I am rendering a chart to the output stream. Instead of streaming it directly to the response.out, I create a ChartResult, and let webwork do the chaining for me.

I generate the chart in one class, and I render it out in another class, effectively decoupling the view from the actions. You can easily render it out to a file or some view other than a web response.out if you wish.

Configuration

xwork.xml - result-types definitions

```
<result-types>
  <result-type name="chart" class="myapp.webwork.extensions.ChartResult"/>
</result-types>
```

xwork.xml - action definitions

```
<action name="viewModerationChart"
class="myapp.webwork.actions.ViewModerationChartAction">
  <result name="success" type="chart">
    <param name="width">400</param>
    <param name="height">300</param> </result>
</action>
```

Source Codes

My result class searches for a "chart" in the ValueStack and renders it out...

```
public class ChartResult implements Result {
    private int width;
```

```

private int height;

public void execute(ActionInvocation invocation) throws Exception {
    JFreeChart chart =
        (JFreeChart) invocation.getStack().findValue("chart");
    HttpServletResponse response = ServletActionContext.getResponse();
    OutputStream os = response.getOutputStream();
    ChartUtilities.writeChartAsPNG(os, chart, width, height);
    os.flush();
}

public void setHeight(int height) {
    this.height = height;
}

public void setWidth(int width) {
    this.width = width;
}
}

```

My action class creates the JFreeChart to render...

```

public class ViewModerationChartAction extends ActionSupport {

    private JFreeChart chart;

    public String execute() throws Exception {
        // chart creation logic...
        XYSeries dataSeries = new XYSeries(null);
        for (int i = 0; i <= 100; i++) {
            dataSeries.add(i, RandomUtils.nextInt());
        }
        XYSeriesCollection xyDataset = new XYSeriesCollection(dataSeries);

        ValueAxis xAxis = new NumberAxis("Raw Marks");
        ValueAxis yAxis = new NumberAxis("Moderated Marks");

        // set my chart variable
        chart =
            new JFreeChart(
                "Moderation Function",
                JFreeChart.DEFAULT_TITLE_FONT,
                new XYPlot(
                    xyDataset,
                    xAxis,
                    yAxis,
                    new StandardXYItemRenderer(StandardXYItemRenderer.LINES)),
                false);
        chart.setBackgroundPaint(java.awt.Color.white);

        return super.SUCCESS;
    }

    public JFreeChart getChart() {
        return chart;
    }
}

```

Explanation

```
public JFreeChart getChart() {  
    return chart;  
}
```

makes the chart available on the ValueStack, which the result gets via

```
JFreeChart chart = (JFreeChart) invocation.getStack().findValue("chart");
```

From what I can deduce, the webwork pulls in the height and width variables from the xwork.xml definitions for that particular action...

```
<param name="width">400</param>  
<param name="height">300</param>
```

Suggestions for the next developer...

Currently the "chart" property is hardcoded. There should be a better way of transferring data from the Action to the Result, via some externally defined variable or something.

As mentioned by John Patterson (mailing list), the Action is still dependant on a JFreeChart Chart class. This can be improved. The seperation between Action and View can be made cleaner. A chart-agnostic List or Array can be used as the data, and the configuration of the chart details (font, axis, etc...) be done via the result properties in the xwork.xml.

But hey, the above works for now. Any suggestions are welcome.

Creating charts via CeWolf directly in Velocity templates

See [WW:CeWolf charts using Velocity templates.](#)

Sometimes you need a way to enter tabular data such as list of quantity for products in a shopping cart, marks from a list of examination candidates, etc. If you just have one input value per line item, you can use a HashMap to store the value. This can be expanded to support multiple input values by having multiple HashMap. This describes a number of alternatives using some of more advanced features of WebWork. Assume you want to capture the quantity and a gift note for a list of products in a shopping cart (i.e Amazon).

1. When the number of line items is known

If you are using JSP:
the cart.jsp file in altSyntax

```
<ww:iterator value="cart.items">
  <ww:hidden name="cart.items[%{#rowstatus.index}].productId" value="%{productId}">
  <ww:textfield name="cart.items[%{#rowstatus.index}].qty" value="%{qty}" />
  <ww:textfield name="cart.items[%{#rowstatus.index}].note" value="%{note}" />
</ww:iterator>
```

the cart.jsp file (non altSyntax)

```
<ww:iterator value="cart.items">
  <ww:hidden name="'cart.items[' + #rowstatus.index + '].productId'"
value="productId">
  <ww:textfield name="'cart.items[' + #rowstatus.index + '].qty'" value="qty" />
  <ww:textfield name="'cart.items[' + #rowstatus.index + '].note'" value="note" />
</ww:iterator>
```

Alternatively, if you use Velocity as your view technology of choice:
the cart.vm file

```
#foreach ( $item in $cart.items )
  #set($index = $velocityCount - 1)
  <input type="hidden" name="cart.items[$index].productId" value="$item.productId">
  <input type="text" name="cart.items[$index].qty" value="$item.qty">
  <input type="text" name="cart.items[$index].note" value="$item.note">
#end
```

the UpdateCartAction.class

```
public class UpdateCartAction extends ActionSupport {

    public Cart getCart() {
        // Lazy initialization
        Cart result = ActionContext.getContext().getSession.get("cart.key");
        if ( result == null ) {
            result = new Cart();
            ActionContext.getContext().getSession.put("cart.key",
result);
        }
        return result;
    }

    public String execute() throws Exception {
        // Just ensuring our cart is initialized...
        Cart cart = getCart();

        // loop through a
    }
}
```

the Cart.class

```
public class Cart implements Serializable {
    private List items = new ArrayList();

    public List getItems() {
        return items;
    }

    public void addItem(CartItem item) {
        ...
    }
}
```

the CartItem.class

```
public class CartItem implements Serializable {
    private int qty;
    private int productId;
    private String note;

    // getters/setters...
}
```

Explanation

The resulting html code is rendered as


```
<input type="hidden" name="cart.items[0].productId" value="1">
<input type="text" name="cart.items[0].qty" value="2">
<input type="text" name="cart.items[0].note" value="This is a fun book!">

<input type="hidden" name="cart.items[1].productId" value="2">
<input type="text" name="cart.items[1].qty" value="2">
<input type="text" name="cart.items[1].note" value="You love this one">

<input type="hidden" name="cart.items[2].productId" value="3">
<input type="text" name="cart.items[2].qty" value="$item.qty">
<input type="text" name="cart.items[2].note" value="">
```

Webwork will populate all the entries in Cart with the correct values.

In depth, the ParametersInterceptor would apply the form results to our model, leading to the call similar like

```
((CartItem) updateCartAction.getCart().getItems().get(0)).setProductId(1);
```

for the first shown line in the rendered result.

2. When the number of line items is unknown

For example, you want to allow the user to enter any number of ISBN, quantity and a note. You can replace ArrayList with XWorkList, which will automatically create new items if the index is greater than the size of the list.

3. Use Type Conversion

If you want more advanced way to do this, check out [XW:Null Property Access](#) for type conversion.

Transparent web-app I18N

This page last changed on Nov 30, 2004 by [jcarreira](#).

Consider adding transparent i18n with simple on-the-fly locale switching to your application via I18NInterceptor.

The main idea:

Interceptor could track locale switch requests, persist selection in current session and set locale for all (or appropriate) actions invoked.

```
package neuro.util.xwork;

import com.opensymphony.xwork.ActionSupport;
import com.opensymphony.xwork.ActionInvocation;
import com.opensymphony.xwork.interceptor.Interceptor;

import java.util.Locale;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;

/**
 * I18nInterceptor
 * @author Aleksei Gopachenko
 */
public class I18nInterceptor implements Interceptor
{
    protectedstaticfinal Log log = LogFactory.getLog(I18nInterceptor.class);

    publicstaticfinalString DEFAULT_SESSION_ATTRIBUTE = "WW_TRANS_I18N_LOCALE";
    publicstaticfinalString DEFAULT_PARAMETER = "request_locale";

    protectedString parameterName = DEFAULT_PARAMETER;
    protectedString attributeName = DEFAULT_SESSION_ATTRIBUTE;

    /**
     */
    public I18nInterceptor()
    {
        if(log.isDebugEnabled()) log.debug("new I18nInterceptor()");
    }

    public void setParameterName(String parameterName) {
        this.parameterName = parameterName;
    }

    public void setAttributeName(String attributeName) {
        this.attributeName = attributeName;
    }

    /**
     */
    public void init() {
        if(log.isDebugEnabled()) log.debug("init()");
    }
}
```

```

/**
 */
public void destroy() {
    if(log.isDebugEnabled()) log.debug("destroy()");
}

/**
 */
public String intercept(ActionInvocation invocation) throws Exception {
    if(log.isDebugEnabled()) log.debug("intercept '"
        +invocation.getProxy().getNamespace()+"/"
        +invocation.getProxy().getActionName()+"' { ");

    //get requested locale
    Object requested_locale =
    invocation.getInvocationContext().getParameters().get(parameterName);
    if(requested_locale!=null && requested_locale.getClass().isArray() &&
    ((Object[])requested_locale).length==1) {
        requested_locale=((Object[])requested_locale)[0];
    }
    if(log.isDebugEnabled()) log.debug("requested_locale="+requested_locale);
    //save it in session
    if (requested_locale!=null) {
        Locale locale = (requested_locale instanceof Locale)?
        (Locale) requested_locale :
        localeFromString(requested_locale.toString());
        if(log.isDebugEnabled()) log.debug("store locale="+locale);
        if(locale!=null)invocation.getInvocationContext().getSession().put(attributeName,locale);
    }
    //set locale for action
    Object locale = invocation.getInvocationContext().getSession().get(attributeName);
    if (locale!=null && locale instanceof Locale) {
        if(log.isDebugEnabled()) log.debug("apply locale="+locale);
        invocation.getInvocationContext().setLocale((Locale)locale);
    }

    if(log.isDebugEnabled()) log.debug("before
Locale="+((ActionSupport)invocation.getAction()).getLocale());
    final String result = invocation.invoke();
    if(log.isDebugEnabled()) log.debug("after
Locale="+((ActionSupport)invocation.getAction()).getLocale());

    if(log.isDebugEnabled()) log.debug("intercept } ");
    return result;
}

Locale localeFromString(String localeStr) {
    if ((localeStr == null) || (localeStr.trim().length() == 0) ||
(localeStr.equals("_"))) {
        return Locale.getDefault();
    }
    int index = localeStr.indexOf('_');
    if (index < 0) {
        return new Locale(localeStr);
    }
    String language = localeStr.substring(0,index);
    if (index == localeStr.length()) {
        return new Locale(language);
    }
    localeStr = localeStr.substring(index +1);
    index = localeStr.indexOf('_');

```

```

        if (index < 0) {
            returnnew Locale(language,localeStr);
        }
        String country = localeStr.substring(0,index);
        if (index == localeStr.length()) {
            returnnew Locale(language,country);
        }
        localeStr = localeStr.substring(index +1);
        returnnew Locale(language,country,localeStr);
    }
}

```

Can be enabled for whole package via something like:

```

<interceptor name="i18n" class="neuro.util.xwork.I18nInterceptor">
    <!-- on which request parameter we should react, optional, defaults to
    "request_locale" -->
    <param name="parameterName">set_locale</param>
    <!-- under which session attribute locale should be stored, optional, defaults to
    "WW_TRANS_I18N_LOCALE" -->
    <param name="attributeName">ww_locale</param>
</interceptor>

<interceptor-stack name="i18nStack">
    <interceptor-ref name="i18n"/>
    <interceptor-ref name="defaultStack"/>
</interceptor-stack>

<default-interceptor-ref name="i18nStack"/>

```

...and invoked in web-app just by adding few links to menu:

```

<a href="?set_locale=en">EN</a>
<a href="?set_locale=ru">RU</a>
<!-- etc -->

```

Of course you still need to move all explicitly defined messages or labels to appropriate ResourceBundles and make translations. Be sure to check out your

- Actions - to use getText(...)
- *-validation.xml files - to use <message key=...>
- results/views - to use WW i18n services by <webwork:text ...> tag, or directly by evaluating getText(...) OGNL expression on current stack.

If this Interceptor is generally useful, may be it should go into codebase?

Using Checkboxes (General)

The biggest gotcha for newbies is that you must set the 'value' attribute in the html `<input>` field to use Checkboxes with WW. By default your browser will set this to some value. Firefox uses "on" - not sure what IE or others use. You must make this a sensible value for whatever property you are setting.

Using Checkboxes to set boolean fields

HTML:

```
<input type="checkbox" name="user.lockedOut" value="true"/>
```

If the user checks this box, the browser will send "user.lockedOut=true" in the QueryString and `action.getUser().setLockedOut(true)` will be called. If the user does not check the box, the browser will not send anything, so make sure that you have initialised `lockedOut` to false to start with.

```
private boolean m_lockedOut = false;

public void setLockedOut(boolean lockedOut) { m_lockedOut = lockedOut; }
```

Using Checkboxes to set a collection

Our user has a number of privileges that are stored as a Set of strings. To use checkboxes for these, we have HTML that looks like:

```
<input type="checkbox" name="user.priv" value="boss"/>
<input type="checkbox" name="user.priv" value="admin"/>
<input type="checkbox" name="user.priv" value="manager"/>
```

Say a user checks the first 2; the browser will send the query string:
user.priv=boss&user.priv=admin.

OGNL will end up calling

```
action.getUser().setPriv(String[] {"boss", "admin"})
```

You can write this method like:

```
Set m_privileges = new HashSet();

public void setPriv(String[] privs) {
    for (int i = 0; i < privs.length; i++) {
        m_privileges.add(privs[i]);
    }
}
```

Full Detailed example:

This example uses a kind-of [XW:Interceptors#ModelDriven](#) Action. The action returns a single getter for the User object whose values are populated.

- [WW:Using Checkboxes - EditAction.java](#)
- [WW:Using Checkboxes - Velocity and HTML](#)
- [WW:Using Checkboxes - User.java](#)

Using Checkboxes - EditAction.java

This page last changed on Jun 18, 2004 by [plightbo](#).

```
package cash.action;

import org.apache.log4j.Logger;

import cash.config.ConfigManager;
import cash.model.User;
import cash.util.HibernateUtil;
import cash.validator.PasswordFormatValidator;

import net.sf.hibernate.LockMode;

/**
 * Edit a user
 * @author Joel Hockey
 * @version $Id: $
 */
public class EditAction extends HibernateAction {
    private static final Logger LOG = Logger.getLogger(EditAction.class);

    private User m_user = new User();
    private String m_repeatPassword;

    /** return user to be edited. */
    public User getUser() { return m_user; }

    /** @param pwd repeat of password */
    public void setRepeatPassword(String pwd) { m_repeatPassword = pwd; }
    /** @return repeat password */
    public String getRepeatPassword() { return m_repeatPassword; }

    /** override super */
    public String execute() throws Exception {
        LOG.debug("EditAction started");

        // get original user from session, check that password is valid, update and
        save.
        User u = (User) get("user");
        HibernateUtil.currentSession().lock(u, LockMode.NONE);

        // check that password has actually changed before updating
        if (!PasswordFormatValidator.PASSWORD_MASK.equals(m_user.getPassword())) {
            if (!u.changePassword(m_user.getPassword())) {
                addFieldError("user.password", "password must be different to
previous "
                                + ConfigManager.getConfig().getUser().getNoRepeatHistory() + "
passwords");
                return INPUT;
            }
        }

        m_user.copy(u);
        HibernateUtil.currentSession().save(u);
        User loginUser = (User) get(LoginAction.LOGIN_USER);
        if (u.getId() == loginUser.getId()) {
            set(LoginAction.LOGIN_USER, u);
        }
        return SUCCESS;
    }
}
```

```
}  
}
```


Using Checkboxes - User.java

This page last changed on Jun 18, 2004 by [plightbo](#).

```
package cash.model;

import net.sf.hibernate.HibernateException;

import org.apache.log4j.Logger;

import java.security.GeneralSecurityException;
import java.security.MessageDigest;
import java.security.SecureRandom;
import java.util.ArrayList;
import java.util.Date;
import java.util.HashSet;
import java.util.Iterator;
import java.util.List;
import java.util.Locale;
import java.util.Set;
import java.util.SortedSet;
import java.util.TimeZone;
import java.util.TreeSet;

import cash.config.ConfigManager;
import cash.util.Hex;
import cash.util.HibernateUtil;
import cash.util.UtcDate;
import cash.validator.PasswordFormatValidator;

/**
 * Represents a User object.  Clients of this class should instantiate a User object
 * with the
 * multi-arg constructor rather than using setters.
 *
 * @author Joel Hockey
 * @version $Id: $
 * @hibernate.class
 *     table="user"
 *     dynamic-update="true"
 *     optimistic-lock="version"
 */
public class User implements java.io.Serializable {
    private static final Logger LOG = Logger.getLogger(User.class);

    private static MessageDigest s_md5;
    private static SecureRandom s_random;

    private static final int MAX_LOGIN_FAILURE_COUNT = 20;
    private static final boolean RESET_LOCKED_OUT_AFTER_TIME = true;
    private static final long RESET_LOCKED_OUT_TIME = 1 * 60 * 60 * 1000; // 1 hour
    private int m_id;
    private int m_version;
    private String m_username;
    private String m_password;
    private Date m_passwordChangeDate;
    private String m_hashedPassword;
    private SortedSet m_passwordHistory = new TreeSet();
    private String m_salt;
    private byte[] m_saltBytes;
    private Date m_createDate;
```

```

privateString m_email;
private Locale m_locale;
private TimeZone m_timeZone;
privateString m_telephone;
private Date m_lastSuccessfulLogin;
privateString m_lastSuccessfulLoginIp;
private Date m_lastFailedLogin;
privateString m_lastFailedLoginIp;
privateint m_loginFailureCount;
privateint m_maxLoginFailureCount = MAX_LOGIN_FAILURE_COUNT;
privateboolean m_resetLockedOutAfterTime = RESET_LOCKED_OUT_AFTER_TIME;
privatelong m_resetLockedOutTime = RESET_LOCKED_OUT_TIME;
privateboolean m_lockedOut = false;
privateboolean m_disabled = false;
privateboolean m_isSuperUser = false;
privateboolean m_passwordNeverExpires = false;
private Set m_privileges = new HashSet();

static {
    try {
        s_md5 = MessageDigest.getInstance("MD5");
        s_random = SecureRandom.getInstance("SHA1PRNG");
    } catch (GeneralSecurityException gse) {
        // shouldn't happen
        LOG.error("Error creating MD5 or SHA1PRNG", gse);
        thrownew RuntimeException("Error creating MD5 or SHA1PRNG");
    }
}

/** default constructor for Hibernate */
public User() { }

/**
 * Create a User.
 *
 * @param username The username for logging in
 * @param password The user's password
 * @param email The user's email
 * @throws InvalidPasswordException if password is invalid.
 */
public User(String username, String password, String email) throws
InvalidPasswordException {

    m_username = username;

    // password
    initSalt();
    if (!PasswordFormatValidator.checkPasswordFormat(password)) {
        thrownew InvalidPasswordException();
    }
    m_hashedPassword = hashPassword(password);

    m_createDate = UtcDate.createUtcDate();
    m_email = email;
    m_locale = Locale.getDefault();
    m_timeZone = TimeZone.getDefault();
}

/** @param id The id to set */
public void setId(int id) { m_id = id; }

/**

```

```

    * @return unique id of this User. Generated by DB.
    * @hibernate.id
    *     generator-class="native"
    */
    public int getId() { return m_id; }

    /** @param version The version of this object */
    public void setVersion(int version) { m_version = version; }

    /**
     * @return version of this object
     * @hibernate.version
     */
    public int getVersion() { return m_version; }

    /** @param username The username to set */
    public void setUsername(String username) { m_username = username; }

    /**
     * @return username
     * @hibernate.property
     *     length="32"
     *     unique="true"
     *     not-null="true"
     */
    public String getUsername() { return m_username; }

    /**
     * Set's the user's password without updating history or checking validity.
     * This should only be used at User creation time, and password validity
     * should be checked externally to this method.
     * Do not use to update password, see {@link #changePassword(String)}
     * @param password user's password
     */
    public void setPassword(String password) {
        m_password = password;
        if (m_salt == null) {
            initSalt();
        }
        m_hashedPassword = hashPassword(password);
        m_passwordChangeDate = UtcDate.createUtcDate();
    }

    /**
     * This method is provided to help at User creation time. It will only return
     * valid values if {@link #setPassword(String)} has already been called.
     * @return plaintext password.
     */
    public String getPassword() { return m_password; }

    /** @param time Date (UTC) user last changed password. */
    public void setPasswordChangeDate(Date time) { m_passwordChangeDate = time; }

    /**
     * @return UTC date of last password change
     * @hibernate.property
     *     type="cash.model.TimestampType"
     *     length="23"
     */
    public Date getPasswordChangeDate() { return m_passwordChangeDate; }

    /**

```

```

    * Sets the user's hashed password. This method is provided only for the use
    * of hibernate. Users of this class should not call this method.
    * Use the {@link #setPassword(String)} method to set the plaintext password.
    * @param hash The hashed password to set
    */
    public void setHashedPassword(String hash) {
        m_hashedPassword = hash;
    }

    /**
     * @return hashed password
     * @hibernate.property
     *     column="pwd"
     *     length="32"
     *     not-null="true"
     */
    public String getHashedPassword() { return m_hashedPassword; }

    /**
     * @param oldPasswords The last n passwords, where n
     * is defined as noRepeatHistory in User configuration. Passwords are ordered
     * in descending order of creation.
     */
    public void setPasswordHistory(SortedSet oldPasswords) { m_passwordHistory =
oldPasswords; }

    /**
     * @return Password history
     * @hibernate.set
     *     lazy="true"
     *     sort="cash.model.PasswordHistory"
     *     inverse="true"
     *     cascade="all"
     * @hibernate.collection-key
     *     column="userId"
     * @hibernate.collection-one-to-many
     *     class="cash.model.PasswordHistory"
     */
    public SortedSet getPasswordHistory() { return m_passwordHistory; }

    /** @param random The random salt to be used with password */
    public void setSalt(String random) {
        m_salt = random;
        m_saltBytes = Hex.fromString(random);
    }

    /**
     * @return random salt used with password
     * @hibernate.property
     *     length="32"
     *     not-null="true"
     */
    public String getSalt() { return m_salt; }

    /** @param time create date */
    public void setCreateDate(Date time) { m_createDate = time; }

    /**
     * @return Date in UTC user was created.
     * @hibernate.property
     *     update="false"
     *     not-null="true"

```

```

        *         type="cash.model.TimestampType"
        *         length="23"
    */
    public Date getCreateDate() { return m_createDate; }

    /** @param email User's email */
    public void setEmail(String email) { m_email = email; }

    /**
     * @return User's email
     * @hibernate.property
     *         length="255"
     *         not-null="true"
     */
    public String getEmail() { return m_email; }

    /** @param locale The User's locale. This should be a 2 character field. */
    public void setLocale(Locale locale) { m_locale = locale; }

    /**
     * @return User's locale. Uses 2 character ISO-something value.
     * @hibernate.property
     *         not-null="true"
     */
    public Locale getLocale() { return m_locale; }

    /** @param timeZone User's time zone */
    public void setTimeZone(TimeZone timeZone) { m_timeZone = timeZone; }

    /**
     * @return User's timezone
     * @hibernate.property
     *         not-null="true"
     */
    public TimeZone getTimeZone() { return m_timeZone; }

    /** @param telephone User's telephone */
    public void setTelephone(String telephone) { m_telephone = telephone; }

    /**
     * @return Telephone of user
     * @hibernate.property
     *         length="16"
     */
    public String getTelephone() { return m_telephone; }

    /** @param time user's last successful login date in UTC. */
    public void setLastSuccessfulLogin(Date time) { m_lastSuccessfulLogin = time; }

    /**
     * @return UTC date of last successful login
     * @hibernate.property
     *         type="cash.model.TimestampType"
     *         length="23"
     */
    public Date getLastSuccessfulLogin() { return m_lastSuccessfulLogin; }

    /** @param ip IP address used for user's last successful login. */
    public void setLastSuccessfulLoginIp(String ip) { m_lastSuccessfulLoginIp = ip; }
}

/**

```

```

    * @return IP address used for last successful login
    * @hibernate.property
    */
    public String getLastSuccessfulLoginIp() { return m_lastSuccessfulLoginIp; }

    /** @param time user's last failed login date in UTC. */
    public void setLastFailedLogin(Date time) { m_lastFailedLogin = time; }

    /**
     * @return UTC date of last failed login
     * @hibernate.property
     *     type="cash.model.TimestampType"
     *     length="23"
     */
    public Date getLastFailedLogin() { return m_lastFailedLogin; }

    /** @param ip IP address used for user's last failed login. */
    public void setLastFailedLoginIp(String ip) { m_lastFailedLoginIp = ip; }

    /**
     * @return IP address used for last failed login
     * @hibernate.property
     */
    public String getLastFailedLoginIp() { return m_lastFailedLoginIp; }

    /**
     * Sets the number of times that a user has failed when attempting to login.
     * This value is reset when a user logs in successfully, or their account is
reset.
     * @param count the value to set.
     */
    public void setLoginFailureCount(int count) { m_loginFailureCount = count; }

    /**
     * @return The number of times that a user has failed when attempting to login.
     * This value is reset when a user logs on successfully, or their account is
reset.
     * @hibernate.property
     */
    public int getLoginFailureCount() { return m_loginFailureCount; }

    /**
     * @param count The maximum number of times that a user may fail to login before
     * their account is locked out
     */
    public void setMaxLoginFailureCount(int count) { m_maxLoginFailureCount = count;
}

    /**
     * @return The maximum number of times that a user may fail to login before
their account
     * is locked out.
     * @hibernate.property
     */
    public int getMaxLoginFailureCount() { return m_maxLoginFailureCount; }

    /**
     * @param reset Whether this user's account will be unlocked after a specified
time when it is locked
     * due to login failure.
     * @see #setResetLockedOutAfterTime(boolean) setResetLockedOutAfterTime
     */

```

```

    public void setResetLockedOutAfterTime(boolean reset) {
m_resetLockedOutAfterTime = reset; }

    /**
     * @return Whether this user's account will be unlocked after a specified time
when it
     * is locked out due to login failure.
     * @see #getResetLockedOutAfterTime getResetLockedOutAfterTime
     * @hibernate.property
     */
    public boolean getResetLockedOutAfterTime() { return m_resetLockedOutAfterTime; }

    /**
     * @param time The time in millis between login attempts before login failure
count is reset. Login failure
     * count will only be reset if the Reset Locked Out After Time boolean is set to
true.
     */
    public void setResetLockedOutTime(long time) { m_resetLockedOutTime = time; }

    /**
     * @return Time in milliseconds before account is auto-reset after login
lockout.
     * @hibernate.property
     */
    public long getResetLockedOutTime() { return m_resetLockedOutTime; }

    /** @param lockedOut User's locked out status. */
    public void setLockedOut(boolean lockedOut) { m_lockedOut = lockedOut; }

    /**
     * @return Whether this user's account is locked out
     * @hibernate.property
     */
    public boolean isLockedOut() { return m_lockedOut; }

    /** @param disabled User's disabled status. */
    public void setDisabled(boolean disabled) { m_disabled = disabled; }

    /**
     * @return Whether this user's account disabled
     * @hibernate.property
     */
    public boolean isDisabled() { return m_disabled; }

    /** @param superUser True if user is super user */
    public void setSuperUser(boolean superUser) { m_isSuperUser = superUser; }

    /**
     * @return Whether this user is a super user
     * @hibernate.property
     */
    public boolean isSuperUser() { return m_isSuperUser; }

    /** @param expires True if user's password never expires */
    public void setPasswordNeverExpires(boolean expires) { m_passwordNeverExpires =
expires; }

    /**
     * @return Whether this user's password ever expires
     * @hibernate.property
     */

```

```

    public boolean getPasswordNeverExpires() { return m_passwordNeverExpires; }

    /** @param privs Set of privileges for this user */
    public void setPrivileges(Set privs) { m_privileges = privs; }

    /**
     * @return Set of Privileges for this User.
     * @hibernate.set
     *     table="user_priv"
     *     lazy="true"
     *     cascade="all"
     * @hibernate.collection-key
     *     column="userId"
     * @hibernate.collection-element
     *     column="priv"
     *     type="string"
     */
    public Set getPrivileges() { return m_privileges; }

    /** convenience method of OGNL */
    public void setPriv(String[] privs) {
        for (int i = 0; i < privs.length; i++) {
            m_privileges.add(privs[i]);
        }
    }

    // other methods

    /**
     * Changes the user's password. Password must meet criteria
     * defined in configuration. The user's password will be appended to
     * a random 20 byte salt and then hashed using MD5 to create the
     * value that will be stored in the DB. The current Hibernate Session
     * will be used to update pwd history.
     *
     * @param password The password to set
     * @return true if password is changed, false if password was not changed
     * because it did not meet password requirements.
     * @throws HibernateException if error updating password history
     */
    public boolean changePassword(String password) throws HibernateException {
        // check format
        if (!PasswordFormatValidator.checkPasswordFormat(password)) {
            return false;
        }

        // check history
        // first check current password
        String hashedPwd = hashPassword(password);
        LOG.debug("checking if password is same as current");
        if (hashedPwd.equals(m_hashedPassword)) {
            LOG.info("password is same as current password");
            return false;
        }

        LOG.debug("checking if password exists in history. History size is " +
            m_passwordHistory.size());
        for (Iterator i = getPasswordHistory().iterator(); i.hasNext(); ) {
            PasswordHistory ph = (PasswordHistory) i.next();
            if (hashedPwd.equals(ph.getHashedPassword())) {
                LOG.info("password already used as one of last "

```



```

        + ConfigManager.getConfig().getUser().getNoRepeatHistory());
        return false;
    }
}

// add current pwd to history and truncate history if it is too long now
PasswordHistory ph = new PasswordHistory(this, m_hashedPassword);
m_passwordHistory.add(ph);
LOG.debug("saving old password into password history");
HibernateUtil.currentSession().save(ph);
// compare to (noRepeat - 1) because we are checking current as part of
history
if (m_passwordHistory.size() >
    ConfigManager.getConfig().getUser().getNoRepeatHistory() - 1) {
    PasswordHistory toRemove = (PasswordHistory)m_passwordHistory.first();
    LOG.info("Removing password history object for user " + m_username
        + " created: " + toRemove.getCreateDate());
    m_passwordHistory.remove(toRemove);
    HibernateUtil.currentSession().delete(toRemove);
}

// now set password and date
m_hashedPassword = hashedPwd;
m_passwordChangeDate = UtcDate.createUtcDate();
return true;
}

/**
 * Hashes input pwd to see if it equals stored pwd hash value.
 * @param pwd Password to check
 * @return true if passwords are equal.
 */

public boolean passwordEquals(String pwd) {
    String hash = hashPassword(pwd);
    return m_hashedPassword.equalsIgnoreCase(hash);
}

/**
 * Hashes salt and password to produce hashed password.
 * @param pwd Password to hash
 * @return Hex encoding of MD5 hash of salt and pwd
 */
private String hashPassword(String pwd) {
    byte[] pwdBytes = pwd.getBytes(); //TODO: should an encoding be specified
    here?
    byte[] in = new byte[OS.m_saltBytes.length + pwdBytes.length];
    System.arraycopy(m_saltBytes, 0, in, 0, m_saltBytes.length);
    System.arraycopy(pwdBytes, 0, in, m_saltBytes.length, pwdBytes.length);
    byte[] out = s_md5.digest(in);
    return Hex.toString(out);
}

/** initialises salt */
private void initSalt() {
    m_saltBytes = new byte[OS.m_saltBytes.length];
    s_random.nextBytes(m_saltBytes);
    m_salt = Hex.toString(m_saltBytes);
}

/** @return String representation of User */
public String toString() {

```

```

        StringBuffer sb = newStringBuffer(500);
        sb.append("[").append("ID:").append(m_id)
        .append(",version:").append(m_version)
        .append(",hashedPassword:").append(m_hashedPassword)
        .append(",salt:").append(m_salt)
        .append(",createDate:").append(m_createDate)
        .append(",email:").append(m_email)
        .append(",locale:").append(m_locale)
        .append(",timeZone:").append(m_timeZone)
        .append(",telephone:").append(m_telephone)
        .append(",lastSuccessfulLogin:").append(m_lastSuccessfulLogin)
        .append(",lastSuccessfulLoginIp:").append(m_lastSuccessfulLoginIp)
        .append(",lastFailedLogin:").append(m_lastFailedLogin)
        .append(",lastFailedLoginIp:").append(m_lastFailedLoginIp)
        .append(",loginFailureCount:").append(m_loginFailureCount)
        .append(",maxLoginFailureCount:").append(m_maxLoginFailureCount)
        .append(",resetLockedOutAfterTime:").append(m_resetLockedOutAfterTime)
        .append(",resetLockedOutTime:").append(m_resetLockedOutTime)
        .append(",lockedOut:").append(m_lockedOut)
        .append(",disabled:").append(m_disabled)
        .append(",isSuperUser:").append(m_isSuperUser)
        .append(",passwordNeverExpires:").append(m_passwordNeverExpires)
        .append(",passwordChangeDate:").append(m_passwordChangeDate)
        .append(",privs:").append(m_privileges);
        return sb.toString();
    }

    /**
     * Copies editable data from this object to User object provided. This is used
     * in Edit actions. Not all fields are copied, only those that are editable
     * @param user Object to copy to
     */
    public void copy(User user) {
        user.setUsername(m_username);
        user.setEmail(m_email);
        user.setLocale(m_locale);
        user.setTimeZone(m_timeZone);
        user.setTelephone(m_telephone);
        user.setLockedOut(m_lockedOut);
        user.setDisabled(m_disabled);
        user.setPasswordNeverExpires(m_passwordNeverExpires);

        // do some smarts for privs removal. Clear all if more than half are
        removed
        if (m_privileges.size() <= user.getPrivileges().size() / 2) {
            LOG.debug("detected that many privs are removed, clearing all");
            user.setPrivileges(m_privileges);
        } else {
            // find which ones should be removed
            List toRemove = new ArrayList();
            for (Iterator i = user.getPrivileges().iterator(); i.hasNext(); ) {
                String priv = (String)i.next();
                if (!m_privileges.contains(priv)) {
                    toRemove.add(priv);
                }
            }

            // remove them
            for (int i = 0; i < toRemove.size(); i++) {
                user.getPrivileges().remove(toRemove.get(i));
            }
        }
    }

```

```
        // add all new privs
    for (Iterator i = m_privileges.iterator(); i.hasNext(); ) {
        user.getPrivileges().add(i.next());
    }
}
}
```

Using Checkboxes - Velocity and HTML

This page last changed on Jun 18, 2004 by [plightbo](#).

Velocity View - edit.vm:

```
<html>
<body onload="document.forms[0].elements[0].focus()">

<a href="home.vm">Home</a><br/>

#if ($fieldErrors)
  #foreach ($error in $fieldErrors)
    $error<br>
  #end
#end
#if ($actionErrors)
  #foreach ($error in $actionErrors)
    $error<br>
  #end
#end

<form name="edit" action="edit.action" method="post">
<table>
<tr><td>Name</td><td>$user.username</td></tr>
#formRowText("Password" "user.password"
$stack.findValue("@cash.validator.PasswordFormatValidator@PASSWORD_MASK"))
#formRowText("Repeat Password" "repeatPassword"
$stack.findValue("@cash.validator.PasswordFormatValidator@PASSWORD_MASK"))
#formRowText("Email" "user.email" $!user.email)
#formRowSelect("Language" "user.locale"
$stack.findValue("@cash.util.Html@getInstance()").getLocales($locale)
$!user.locale.toString())
#formRowSelect("Time Zone" "user.timeZone"
$stack.findValue("@cash.util.Html@getInstance()").getTimeZones($locale)
$!user.timeZone.ID)
#formRowText("Telephone" "user.telephone" $!user.telephone)
#formRowCheckbox("Locked Out" "user.lockedOut" "true" $user.lockedOut)
#formRowCheckbox("Disabled" "user.disabled" "true" $user.disabled)

#set ($privs = [OS:"boss", "admin", "early", "late", "train"])

#foreach ($priv in $privs)
  #set ($checked = $user.privileges.contains($priv))
  #formRowCheckbox($priv "user.priv" $priv $checked)
#end
<tr><td>&nbsp;</td><td><input type="submit" name="submit" value="submit"></td></tr>
</table>

<input type="hidden" name="user.username" value="$user.username">
</form>

</body>
</html>
```

Velocity Macros - macros.vm:

```

#macro (formRowText $label $name $value)
  <tr><td><label for="$name">$label</label></td><td><input id="$name" type="text"
name="$name" value="$!value"></td></tr>
#end

#macro (formRowSelect $label $name $options $selectedValue)
  <tr><td><label for="$name">$label</label></td><td><select id="$name" name="$name">
#foreach ($option in $options)
  <option#if ($option.get(0).equals($selectedValue)) selected#end
value="$!option.get(0)">$!option.get(1)</option>
#end
</select></td></tr>
#end

#macro (formRowCheckbox $label $name $value $checked)
  <tr><td><label for="$name.$value">$label</label></td><td><input id="$name.$value"
type="checkbox" name="$name" value="$value"#if ($checked) checked#end ></td></tr>
#end

```

Note that I don't use the webwork UI tags. (The HTML that comes out of them looks like vomit.)

The HTML generated from above looks like:

```

<html>
<body onload="document.forms[0].elements[0].focus()">

<a href="home.vm">Home</a><br/>

<form name="edit" action="edit.action" method="post">
<table>
<tr><td>Name</td><td>user</td></tr>
  <tr><td><label for="user.password">Password</label></td><td><input
id="user.password" type="text" name="user.password" value="*****"></td></tr>
  <tr><td><label for="repeatPassword">Repeat Password</label></td><td><input
id="repeatPassword" type="text" name="repeatPassword" value="*****"></td></tr>

  <tr><td><label for="user.email">Email</label></td><td><input id="user.email"
type="text" name="user.email" value="user@example.com"></td></tr>
  <tr><td><label for="user.locale">Language</label></td><td><select id="user.locale"
name="user.locale">
<option value="en">English</option>
<option selected value="en_AU">English (Australia)</option>
<option value="en_US">English (United States)</option>
<option value="en_GB">English (United Kingdom)</option>
<option value="es">Spanish</option>
<option value="fr">French</option>

<option value="de">German</option>
</select></td></tr>
  <tr><td><label for="user.timeZone">Time Zone</label></td><td><select
id="user.timeZone" name="user.timeZone">
<option selected value="America/Los_Angeles">(GMT-08:00) Los Angeles</option>
<option value="Europe/London">(GMT+00:00) London</option>
<option value="Australia/Brisbane">(GMT+10:00) Brisbane</option>

```

```

</select></td></tr>
  <tr><td><label for="user.telephone">Telephone</label></td><td><input
id="user.telephone" type="text" name="user.telephone" value="134"></td></tr>
  <tr><td><label for="user.lockedOut.true">Locked Out</label></td><td><input
id="user.lockedOut.true" type="checkbox" name="user.lockedOut" value="true"
></td></tr>

  <tr><td><label for="user.disabled.true">Disabled</label></td><td><input
id="user.disabled.true" type="checkbox" name="user.disabled" value="true"
></td></tr>

  <tr><td><label for="user.priv.boss">boss</label></td><td><input
id="user.priv.boss" type="checkbox" name="user.priv" value="boss" ></td></tr>
  <tr><td><label for="user.priv.admin">admin</label></td><td><input
id="user.priv.admin" type="checkbox" name="user.priv" value="admin" ></td></tr>
  <tr><td><label for="user.priv.early">early</label></td><td><input
id="user.priv.early" type="checkbox" name="user.priv" value="early" ></td></tr>
  <tr><td><label for="user.priv.late">late</label></td><td><input
id="user.priv.late" type="checkbox" name="user.priv" value="late" ></td></tr>

  <tr><td><label for="user.priv.train">train</label></td><td><input
id="user.priv.train" type="checkbox" name="user.priv" value="train" ></td></tr>
  <tr><td>&nbsp;</td><td><input type="submit" name="submit" value="submit"></td></tr>
</table>

<input type="hidden" name="user.username" value="user">
</form>

</body>
</html>

```

This page last changed on Aug 29, 2005 by [plightbo](#).

WW2/WX1 and its taglib is oriented towards OGNL, which is using a value stack for all action properties. These values are not directly available for the expression language of JSP2/JSTL1.1.

However, it's easy to populate the request attribute set, with all gettable properties of an action object. You need to provide an interceptor that does the job, by registering a `PreResultListener` which is invoked after the return of `Action.execute()` but before the rendering of the result.

The interceptor below is using Jakarta BeanUtils. It first extracts all getters of the current action, invokes them one at a time and stores the values into a map. Then it iterates over the map and populates the request attribute set. *The double iteration is not needed, it's just there for clarity.*

class ActionPropertyExportInterceptor

```
package com.whatever.interceptors;

import com.opensymphony.webwork.WebWorkStatics;
import com.opensymphony.xwork.Action;
import com.opensymphony.xwork.ActionInvocation;
import com.opensymphony.xwork.interceptor.AroundInterceptor;
import com.opensymphony.xwork.interceptor.PreResultListener;
import org.apache.commons.beanutils.PropertyUtils;
import javax.servlet.http.HttpServletRequest;
import java.beans.PropertyDescriptor;
import java.util.*;

/**
 * Populates HTTP Request Attributes with all gettable properties of the current
 * action.
 */
public class ActionPropertyExportInterceptor extends AroundInterceptor {
    protected void before(ActionInvocation invocation) throws Exception {
        invocation.addPreResultListener( new PropertyExporter() );
    }
    protected void after(ActionInvocation dispatcher, String result) throws
    Exception { }

    public static class PropertyExporter implements PreResultListener {
        private static final List ignore = Arrays.asList(new String[] { "class",
        "texts" }); //skip getClass,...
        //Invoked after Action.execute() but before Result
    }
}
```

```
//Calls all getters of the action and insert the values into the request
public void beforeResult(ActionInvocation invocation, String resultCode) {
    Map props = extractGetterPropertyValues(
invocation.getAction() );
    HttpServletRequest request = getRequest(invocation);
    for (Iterator it = props.entrySet().iterator(); it.hasNext();) {
        Map.Entry e = (Map.Entry) it.next();
        request.setAttribute((String) e.getKey(), e.getValue());
    }
}

public Map extractGetterPropertyValues(Object bean) {
    PropertyDescriptor[] descr =
PropertyUtils.getPropertyDescriptors(bean);
    Map props = new HashMap();
    for (int i = 0; i < descr.length; i++) {
        PropertyDescriptor d = descr[i];
        if (d.getReadMethod() == null) continue;
        if (ignore.contains(d.getName())) continue;

        try {
            props.put(d.getName(), PropertyUtils.getProperty(bean,
d.getName()));
        } catch (Exception e) { }
    }
    return props;
}

public HttpServletRequest getRequest(ActionInvocation invocation) {
    return (HttpServletRequest)
invocation.getInvocationContext().get(WebWorkStatics.HTTP_REQUEST);
}
}
```

Don't forget to *declare* the interceptor in your xwork.xml file and *insert* it into your interceptor stack.

xwork.xml snippet

```
<interceptor name="export"
class="com.whatever.interceptors.ActionPropertyExportInterceptor" />
. . .
<interceptor-stack name="standard-interceptors">
    <interceptor-ref name="timer" />
    <interceptor-ref name="logger" />
    <interceptor-ref name="params" />
    * <interceptor-ref name="export" /> *
    <interceptor-ref name="validateParams" />
    <interceptor-ref name="awarePluggger" />
</interceptor-stack>
```

Your action need to provide getters for all properties that should be exported into the

request attribute set.

class ViewUser

```
public class ViewUser extends ActionSupport {
    private int id;
    private User user;

    public String execute() throws Exception {
        user = findUser( getId() );
        return Action.SUCCESS;
    }

    public int getId() {return id;}
    public void setId(int id) {this.id = id;}
    * public User getUser() {return user;} *

    private User findUser(int id) {...}
}
```

The User class might look like this

class User

```
import java.util.Date;
public class User {
    private int id;
    private String firstName, lastName, email;
    private String street, zip, city;
    private Date date;

    public String getFirstName() {return firstName;}
    //..._getters and setters_...
}
```

Finally, using the samples above you can write your JSP2 page like this.

ViewUser.jsp

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<%@ taglib prefix="fn" uri="http://java.sun.com/jsp/jstl/functions" %>
<html>
<head>
```

```

    <title>Info about ${user.firstName}</title>
</head>
<body>
    <h1>Info about ${user.firstName} ${user.lastName} [OS:ID=${user.id}]</h1>
    <table border="1" cellspacing="0" cellpadding="2" width="90%" >
        <tr>
            <th>Name</th> <td>${user.firstName} ${user.lastName}</td>
        </tr>
        <tr>
            <th>Created</th> <td><fmt:formatDate value="${user.date}"
pattern="yyyy-MM-dd HH:mm"/></td>
        </tr>
        <tr>
            <th>Email</th> <td>${user.email}</td>
        </tr>
        <tr>
            <th>Address</th> <td>${user.street} ${user.zip}
${fn:toUpperCase(user.city)}</td>
        </tr>
    </table>
</body>
</html>

```

Displaying validation errors with JSTL

```

<c:if test="${!empty fieldErrors || !empty actionErrors}">
    <div class="red">
        <ul>
            <c:forEach items="${fieldErrors}" var="fieldError">
                <c:forEach items="${fieldError.value}" var="error">
                    <li>${error}</li>
                </c:forEach>
            </c:forEach>
            <c:forEach items="${actionErrors}" var="actionError">
                <li>${actionError}</li>
            </c:forEach>
        </ul>
    </div>
</c:if>

```

Using WebWork Components

This page last changed on Nov 30, 2004 by [jcarreira](#).

A simple example of using WebWork components is available in the webwork-example.war that comes with the WebWork 2.0 Beta 1 distribution. You can download the distribution from <https://webwork.dev.java.net/servlets/ProjectDocumentList> .

Components are defined `_/WEB-INF/classes/components.xml_`.

The example consists of one component, which is defined by

```
<component>
  <scope>session</scope>
  <class>com.opensymphony.webwork.example.counter.Counter</class>
  <enabler>com.opensymphony.webwork.example.counter.CounterAware</enabler>
</component>
```

`com.opensymphony.webwork.example.counter.Counter` is just a POJO.

`com.opensymphony.webwork.example.counter.CounterAware` is an interface which your `*Action` classes have to implement.

```
publicinterface CounterAware {
    public void setCounter(Counter counter);
}
```

Additionally, you need to tag your actions with the interceptor, for example,

```
<action name="SimpleCounter"
class="com.opensymphony.webwork.example.counter.SimpleCounter">
  <result name="success" type="dispatcher">
    <param name="location">/success.jsp</param>
  </result>
  <interceptor-ref name="defaultComponentStack"/>
</action>
```

WebWork will call the interface and set the Counter bean . The Counter bean would then be subsequently be available to be used by your `*Action` classes.

This page last changed on Nov 30, 2004 by [jcarreira](#).

As Matt Ho explained on the mailing list:

A value stack is essentially a List. Calling `[1]` on the stack, returns a substack beginning with the element at index 1. It's only when you call methods on the stack that your actual objects will be called.

Said another way, let's say I have a value stack that consists of a model and an action as follows:

```
[ model, action ]
```

here's how the following ognl would resolve:

`[0]` - a CompoundRoot object that contains our stack, `[model, action]`

`[1]` - another CompoundRoot that contains only `[action]`

`[0].toString()` - calls `toString()` on the first object in the value stack (excluding the CompoundRoot) that supports the `toString()` method

`[1].foo` - call `getFoo()` on the first object in the value stack starting from `[OS:action]` and excluding the CompoundRoot that supports a `getFoo()` method

I hope this doesn't sound too confusing :\

If you're using Velocity, this can most easily be written as:

```
$stack.findValue("[0]").peek()
```

Unfortunately, `<ww:property value="[0].peek()"/>` won't work as this

would translate into "starting at the top of the value stack (and excluding the CompoundRoot), find the first object that has a method called peek()"

-----thanks Matt!

here is the com.opensymphony.xwork.util.CompoundRoot class which Matt mentions:

```
public class CompoundRoot extends ArrayList {
    //~ Constructors //////////////////////////////////////
    public CompoundRoot() {
    }

    public CompoundRoot(List list) {
        super(list);
    }

    //~ Methods //////////////////////////////////////
    public CompoundRoot cutStack(int index) {
        return new CompoundRoot(subList(index, size()));
    }

    public Object peek() {
        return get(0);
    }

    public Object pop() {
        return remove(0);
    }

    public void push(Object o) {
        add(0, o);
    }
}
```

What's on the stack?

WebWork2 contains the following items by default in the ValueStack:

- req - the current HttpServletRequest
- res - the current HttpServletResponse
- stack - the current OgnlValueStack
- ognl - an instance of OgnlTool
- ui - a (now deprecated) instance of a ui tag renderer

HTML Form Buttons and Webwork2

This Howto will describe the usage of HTML form buttons to invoke different behavior in actions.

Determine which button was pressed

The trick is that the type conversion of [XW:XWork](#) can be used to test which button was pressed in a simple way. When a button is pressed, a parameter is set in webwork with the name and value that are specified as the *name* and *value* attributes of your HTML button. [XW:XWork](#) converts this automatically to boolean value if an appropriate property of the Action is found.

These boolean Properties can be tested to determine which button was pressed:

```
<form action="MyAction.action">
<input type="submit" name="buttonOnePressed" value="First option">
<input type="submit" name="buttonTwoPressed" value="Alternative Option">
</form>
```

```
public class MyAction extends Action {

    /**
     * Action implementation
     *
     * Sets the message according to which button was pressed.
     */
    public String execute() {
        if (buttonOnePressed) {
            message="You pressed the first button";
        } elseif (buttonTwoPressed) {
            message="You pressed the second button";
        } else {
            return ERROR;
        }
        return SUCCESS;
    }

    // Input parameters
    private boolean buttonOnePressed=false;
    private boolean buttonTwoPressed=false;

    public void setButtonOnePressed(boolean value) {
```

```

        this.buttonOnePressed = value;
    }

    public void setButtonTwoPressed(boolean value) {
        this.buttonTwoPressed = value;
    }

    // Output parameters
    privateString message;
    publicString getMessage() {
        return message;
    }
}

```

**Note_:* Do not use String properties with buttons and test for the value that's set. This will break as soon as the `_value` attribute of the HTML button changes! This is likely because the `value` attribute used as the text shown to the user.

Dynamic set of buttons

Consider a web page showing a shopping cart or similar tabular data. Often there is a button belonging to each row, in case of the shopping cart a delete button to remove the item from the cart. The number of buttons is dynamic and the id that couples the button to an item cannot go to the `value` attribute because all buttons should read "delete".

The solution is to `__name*` the buttons like `delete[123]`, `delete[594]`, `delete[494]` where 123, 594 and 494 are e.g. the items' ids:

```

<form action="UpdateCart.action">
  <ww:iterate value="items">
    <ww:property value="name">
      <input type="submit" name="delete[<ww:property value='id'>]" value="delete" />
    <br/>
  </ww:iterate>
</form>

```

When e.g. the button for the item with the property `id == "27"` is pressed, a parameter named `delete[27]` and value "delete" is set in your action. The trick is to us declare your action's property "delete" as `java.util.Map`. Then, a key will exist for the button that was pressed:

```

public void class UpdateCart implements Action {
    // must be initialized to be usable as a webwork input parameter
    private Map delete = new HashMap();

    /** This is somewhat counter intuitive. But a property like "delete[OS:27]"
     * that is set to "delete" by webwork will be interpreted by the underlying
     * OGNL expression engine as "set the property 27 of the action's property
     * "delete" to the value "delete". So we must provide a getter forthis
     */ action. A setter is not needed.
    public Map getDelete() {
        return delete;
    }

    public String execute() {
        for(Iterator i = delete.keySet().iterator(); i.hasNext(); ) {
            String id = (String) i.next();
            ...
            // do what ever you want
            ...
        }
        ...
    }
}

```

In this case it would not be necessary to iterate the whole keySet because it contains only one key but the same code can be use to handle sets of checkboxes if this is preferred later:

```

<form action="UpdateCart.action">
    <ww:iterate value="items">
        <ww:property value="name">
            <input type="checkbox" name="delete[<ww:property value='item'/>]"
value="delete"> <br/>
        </ww:iterate>
        <input type="submit" name="updateCart" value="Update the cart"/>
    </form>

```

The two implementations can even be combined two provide a quick "delete this item" button and a set of checkboxes for "mass updates". All with the above code, cool eh?

Webwork 2 skinning

This page last changed on Nov 30, 2004 by [jcarreira](#).

Skinning in Webwork 2 can be done more than one way. We will show how to use two skins called "html" and "wml", and we'll be working with the following directory structure:

```
/WEB-INF
  /web.xml
/html
  /index.jsp
  /Register.jsp
/wml
  /index.jsp
  /Register.jsp
/index.jsp
```

Classic Approach

If you want to go the Webwork 1.3 route, simply place all actions in the default namespace so that they are accessible from any URL path. When you create your views, place them in the sub-directory that corresponds with the skin's identifier.

Your action configuration would look like this (simplified, without defined interceptors):

```
<package name="default">
  <action name="registration" class="x.actionset.Register">
    <result name="success" type="dispatcher">
      <param name="location">Register.jsp</param>
    </result>
    <interceptor-ref name="defaultStack"/>
  </action>
</package>
```

If a user requested <http://yoursite/html/register.action>, he would see the JSP located at /html/Register.jsp.

Namespace Defined

If you require the use of namespaces, you can do the following:

Simplified configuration example:

```
<package name="user" extends="default">
  <action name="register" class="x.x.actionset.Register">
    <result name="success" type="dispatcher">
      <param name="location">Register.jsp</param>
    </result>

    <interceptor-ref name="defaultStack"/>
  </action>
</package>

<package name="user-html" extends="user" namespace="/user/html" />
<package name="user-wml" extends="user" namespace="/user/wml" />
```

The last two package definitions extend the first package, changing only the namespace. The view result defined in the "register" action has a relative path. Because of this, you'll get the same behavior as the Classic Approach, but with the security of knowing that ONLY those two paths can be accessed for the action, instead of ANY path.

File upload using WebWork

Webwork comes with built in file upload support. Uploading a file is simple. When `ServletDispatcher` begins it checks to see if the request contains multipart content. If it does the dispatcher creates a `MultipartWrapperRequest`. This wrapper handles receiving the file and saving to disk. It is important for the action programmer to check to see if any errors occurred during processing. Three properties can be set that effect file uploading.

Properties

Webwork properties can be set by putting a file 'webwork.properties' in WEB-INF/classes. Any property found there will override the default value.

1. `webwork.multipart.parser` - This should be set to a class that extends `MultiPartRequest`. Currently WebWork ships with two implementations. "`com.opensymphony.webwork.dispatcher.multipart.PellMultiPartRequest`" and "`com.opensymphony.webwork.dispatcher.multipart.CosMultiPartRequest`" If the property is not found the Pell parser is used.
2. `webwork.multipart.saveDir` - The directory where the uploaded files will be placed. If this property is not set it defaults to `javax.servlet.context.tempdir`.
3. `webwork.multipart.maxSize` - The maximum file size in bytes to allow for upload. This helps prevent system abuse by someone uploading lots of large files. The default value is 2 Megabytes and can be set as high as 2 Gigabytes (higher if you want to edit the Pell multipart source but you really need to rethink things if you need to upload files larger then 2 Gigabytes!) If you are uploading more than one file on a form the `maxSize` applies to the combined total, not the individual file sizes.

If you're happy with the defaults there is no need to put any of the properties in `webwork.properties`. Here is my current `webwork.properties`

```
# don't really need to set this but I put it here for testing
# various values
```

```
webwork.multipart.parser=com.opensymphony.webwork.dispatcher.multipart.PellMultiPartRequest

# put the uploaded files in /tmp. My application will move them to their
# final destination
webwork.multipart.saveDir=/tmp
```

Note, while you can set these properties to new values at runtime the MultiPartRequestWrapper is created and the file handled before your action code is called. So if you want to change values you must do so before this action.

Sample form

```
<%@ taglib uri="webwork" prefix="ww" %><html><head><title>File Upload
Test</title></head><body><h1>File Upload</h1><form action="FileUpload.action"
method="POST" enctype="multipart/form-data"><center><table width="350" border="0"
cellpadding="3" cellspacing="0"><tr><td colspan="2"><input type="file"
name="FileName" value="Browse..." size="50"/></td></tr><tr><td colspan="2"
align="center"><input type="submit"
value="Submit"></td></tr></table></center></form></body></html>
```

That's all you have to do to upload a file. No coding required, the file will be placed in the default directory. However, that leaves us with no error checking among other things. So let's add some code to the Action.

FileUploadAction.java

Before the action method is called the dispatcher will upload the file. Then we can get access to information about the file from MultiPartRequestWrapper.

```
MultiPartRequestWrapper multiWrapper =
    (MultiPartRequestWrapper) ServletActionContext.getRequest();
```

The first thing you should always do is check for errors. If there were any there's no point in continuing, most methods will return null. Unfortunately, currently there is no easy way to distinguish what error occurred making it more difficult to route to different error pages. (I have improving error handling for file uploads on my stack of things I'd like to do sometime).

```
if (multiWrapper.hasErrors()) {
    Collection errors = multiWrapper.getErrors();
```

```

    Iterator i = errors.iterator();
    while (i.hasNext()) {
        addActionError((String) i.next());
    }
    return ERROR;
}

```

Now get the input tag name for the uploaded file and use that to get information on the transfer. Since you can upload multiple files (just add multiple input tags) at a time `getFileNames` returns an Enumeration of the names.

```

Enumeration e = multiWrapper.getFileNames();

while (e.hasMoreElements()) {
    // get the value of this input tag
    String inputValue = (String) e.nextElement();

    // get the content type
    String contentType = multiWrapper.getContentType(inputValue);

    // get the name of the file from the input tag
    String fileName = multiWrapper.getFilesystemName(inputValue);

    // Get a File object for the uploaded File
    File file = multiWrapper.getFile(inputValue);

    // If it's null the upload failed
    if (file == null) {
        addActionError("Error uploading: " +
            multiWrapper.getFilesystemName(inputValue));
    }

    // Do additional processing/logging...
}

```

Further improvements.

Code above may be packed into one nice reusable component (Interceptor) that handles 90% of all typical file upload tasks. And Action does not know anything about web-app and just gets its files. Neat. See [WW:File Upload Interceptor](#)

Webwork reference to OGNL access

This page last changed on Nov 30, 2004 by jcarreira.

Webwork uses a standard naming context to evaluate OGNL expressions. The top level object dealing with OGNL is a map (usually referred as a context map). OGNL has a concept of a root object (in webwork terms, this is the OGNLValueStack). Along with the root, other objects are placed in the context map (referred as in the context) including your session/application/request/attr maps. These objects have nothing to do with the root, they just exist along side it in the context map. So, to access these objects, the # is used telling ognl not to look in the root object, but within the rest of the context

```
|
|
|
|
context map---|-- request
|              |--application
|              |--OgnlValueStack(root)
|              |--session
|              |--attr
|              |--parameters
```

Note that there are other objects in the context map, I'm just referring to a few for this example. Now, your actions instances are placed in the OGNLValueStack so you can refer to your bean properties without the #.

```
<ww: property value="myBean.myProperty"/>
```

for sessions, request, and the rest that lie in the context map

```
ActionContext.getContext().getSession().put("mySessionPropKey", mySessionObject);
```

```
<ww: property value="#session.mySessionPropKey"/> or
```

```
<ww: property value="#session['mySessionPropKey']"/>
```

```
<ww: property value="#attr.mySessionPropKey"/>
```

WebworkVelocity and Sitemesh velocity combined

This page last changed on Mar 11, 2005 by [sutter2k](#).

```
/*
 * Copyright (c) 2002-2003 by OpenSymphony
 * All rights reserved.
 */
package com.diamondip.common.views.velocity;

import java.io.IOException;
import java.io.StringWriter;
import java.io.UnsupportedEncodingException;
import java.io.Writer;

import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.jsp.JspFactory;
import javax.servlet.jsp.PageContext;

import org.apache.velocity.Template;
import org.apache.velocity.context.Context;
import org.apache.velocity.exception.MethodInvocationException;
import org.apache.velocity.exception.ParseErrorException;
import org.apache.velocity.exception.ResourceNotFoundException;
import org.apache.velocity.runtime.RuntimeSingleton;
import org.apache.velocity.servlet.VelocityServlet;
import org.apache.velocity.tools.view.context.ChainedContext;
import org.apache.velocity.tools.view.servlet.VelocityViewServlet;

import com.opensymphony.module.sitemesh.*;
import com.opensymphony.module.sitemesh.util.OutputConverter;
import com.opensymphony.webwork.ServletActionContext;
import com.opensymphony.webwork.config.Configuration;
import com.opensymphony.webwork.views.velocity.VelocityManager;
import com.opensymphony.xwork.ActionContext;

/**
 * @author $Author: Payne_m $
 * @author Matthew Payne
 * @version $Revision: 2 $
 */
public class WebWorkVelocityServlet extends VelocityViewServlet {
    //~ Instance fields ////////////////////////////////////////
    private VelocityManager velocityManager;

    //~ Constructors ////////////////////////////////////////
    public WebWorkVelocityServlet() {
        velocityManager = VelocityManager.getInstance();
    }

    //~ Methods ////////////////////////////////////////
    public void init(ServletConfig servletConfig) throws ServletException {
        super.init(servletConfig);

        // initialize our VelocityManager
        velocityManager.init(servletConfig.getServletContext());
        servletConfig.getServletContext().setAttribute("webwork.servlet", this);
    }
}
```

```

    }

    protected Context createContext(HttpServletRequest request, HttpServletResponse
response) {
        // first get manager's context from www
        Context ctx =
velocityManager.createContext(ActionContext.getContext().getValueStack(), request,
response);
        ChainedContext chained = new ChainedContext(ctx, request, response,
getServletContext());

        /* if we have a toolbox manager, get a toolbox from it */
        if (toolboxManager != null)
        {
chained.setToolbox(toolboxManager.getToolboxContext(chained));
        }
        return chained;
    }

    protected Template handleRequest(HttpServletRequest httpServletRequest,
HttpServletResponse httpServletResponse, Context context) throws Exception {
        HttpServletRequest request = httpServletRequest;

        HTMLPage htmlPage = (HTMLPage) request.getAttribute(RequestConstants.PAGE);
        String template;

        context.put("base", request.getContextPath());
        if (htmlPage == null) {
            context.put("title", "Title?");
            context.put("body", "<p>Body?</p>");
            context.put("head", "<!-- head -->");
            template = (String)
httpServletRequest.getAttribute("javax.servlet.include.servlet_path");

            if (template == null) {
                template = httpServletRequest.getServletPath();
            }
        }
        else {
            context.put("title", OutputConverter.convert(htmlPage.getTitle()));
            {
                StringWriter buffer = new StringWriter();
                htmlPage.writeBody(OutputConverter.getWriter(buffer));
                context.put("body", buffer.toString());
            }
            {
                StringWriter buffer = new StringWriter();
                htmlPage.writeHead(OutputConverter.getWriter(buffer));
                context.put("head", buffer.toString());
            }
            context.put("page", htmlPage);
            Factory factory = Factory.getInstance(new Config(getServletConfig()));
            Decorator decorator = factory.getDecoratorMapper().getDecorator(request,
htmlPage);
            template = decorator.getPage();
        }

        return getTemplate(template, getEncoding());
    }

    /**

```



```

        * create a PageContext and render the template to PageContext.getOut()
        *
        * @see VelocityServlet#mergeTemplate(Template, Context, HttpServletResponse)
for additional documentation
        */
        protected void mergeTemplate(Template template, Context context,
HttpServletResponse response) throws ResourceNotFoundException, ParseException,
MethodInvocationException, IOException, UnsupportedEncodingException, Exception {
            // save the old PageContext
            PageContext oldPageContext = ServletActionContext.getPageContext();

            // create a new PageContext
            JspFactory jspFactory = JspFactory.getDefaultFactory();
            HttpServletRequest request = (HttpServletRequest)
context.get(VelocityManager.REQUEST);

            PageContext pageContext = jspFactory.getPageContext(this, request, response,
null, true, 8192, true);

            // put the new PageContext into ActionContext
            ActionContext actionContext = ActionContext.getContext();
            actionContext.put(ServletActionContext.PAGE_CONTEXT, pageContext);

            try {
                Writer writer = pageContext.getOut();
                template.merge(context, writer);
                writer.flush();
            } finally {
                // perform cleanup
                jspFactory.releasePageContext(pageContext);
                actionContext.put(ServletActionContext.PAGE_CONTEXT, oldPageContext);
            }
        }

        privateString getEncoding() {
            // todo look into converting this to using XWork/WebWork2 encoding rules
        try {
            return Configuration.getString("webwork.il8n.encoding");
        } catch (IllegalArgumentException e) {
            return RuntimeSingleton.getString(RuntimeSingleton.OUTPUT_ENCODING,
DEFAULT_OUTPUT_ENCODING);
        }
    }
}

```

FAQ

This page last changed on Oct 30, 2005 by [plightbo](#).



Each question should be a new page. Typically answers should link to content in the [Reference](#). If the answer isn't in the [Reference](#), then it should probably be added there and then linked to from the FAQ. Also note that some of the questions are current'y too verbose and should be broken down given that they have already been categorized (ie: Validation, Internationalization, etc).

General

- [How do I get the latest version of WebWork](#)
- What are the default variables in the value stack
- How do I get access to the session
- How can I see all parameters passed into the action
- How can I get the HttpServletRequest
- Can I break up my large XWork.xml file into smaller pieces
- I'm trying to run the webwork example in the tutorial on Tomcat, and it can't instantiate the VelocityEngine

Tags

- How can I put a String literal in a Javascript call, for instance in an onChange attribute
- Why won't the 'if' tag evaluate a one char string

Inversion of Control

- How can I integrate WebWork IoC in to an object that is not an action

Validation

- How do I use messages from within the validator

Internationalization

- How do I set a global resource bundle?
- How do I decouple XWork LocalizedTextUtil global resource bundle loading from serlvets
- How do I add I18N to a UI tag, like the textfield tag
- Can I add I18N outside the Action's context

Type Conversion

- How do I change the error message for invalid inputted fields

What are the default variables in the value stack? (accessible using #foo)

attr (scans the request, session, and application attributes, in that order)

request (request attributes)

session (session attributes)

application (application attributes)

parameters (request params)

How do I get the latest version of Webwork and XWork from CVS?

```
cvs -d :pserver:guest@cvs.dev.java.net:/cvs login
```

(Use an empty password, just hit enter..)

```
cvs -d :pserver:guest@cvs.dev.java.net:/cvs checkout webwork
```

```
cvs -d :pserver:guest@cvs.dev.java.net:/cvs checkout xwork
```

Note: WebWork from the CVS does not compile with the latest 1.5 J2sdk. Use the stable J2sdk 1.4.2.

How do I build the latest versions XWork and Webwork?

Just go into the XWork or WebWork directories and run 'ant' (you must have ant installed and have the jars of junit and clover inside \$ANT_HOME/lib)

Once you have built the xwork.jar copy it into the webwork/lib/core folder, and delete the old one.

How do I use messages from within the validator?

```
<validators><field name="name"><field-validator type="requiredstring"><message
key="template.name.errors.required">A default message in case the key is not
found</message></field-validator></field></validators>
```

How do I set a global resource bundle?

In webwork.properties(as of Webwork 2.1.1),
you can now use:

```
webwork.custom.i18n.resources=global-messages
```

Several resource bundles can be specified by comma separating them.
for example see webwork.properties :

<http://wiki.opensymphony.com/display/WW/webwork.properties>

Java class (thanks Drew McAuliffe):

```
public class WebworkGlobalMessagesListener implements ServletContextListener {
    private static Logger log =
        Logger.getLogger(WebworkGlobalMessagesListener.class);
    private static final String DEFAULT_RESOURCE = "global-messages";

    /**
     * Uses the LocalizedTextUtil to load messages from the global
     * message bundle.
     * @see
     * javax.servlet.ServletContextListener#contextInitialized(javax.servlet.Servle
     * tContextEvent)
     */
    public void contextInitialized(ServletContextEvent arg0) {
        log.info("Loading global messages from " + DEFAULT_RESOURCE);
        LocalizedTextUtil.addDefaultResourceBundle(DEFAULT_RESOURCE);
        log.info("Global messages loaded.");
    }

    /**
     * @see
     * javax.servlet.ServletContextListener#contextDestroyed(javax.servlet.ServletContextEvent)
     */
    public void contextDestroyed(ServletContextEvent arg0) {

        // do nothing
    }
}
```

web.xml:

(under listeners section)

```
<listener><listener-class>mypackageName.WebworkGlobalMessagesListener</listener-class></listener>
```

How do I change the error message for invalid inputted fields?

You need to create a message for that field, for example if you have a user.dob field you would use this in your messages file (see above for example on setting a global messages file):

invalid.fieldvalue.user.dob=Please enter Date of Birth in the correct format.

How do I get access to the Session?

ActionContext.getContext().getSession() (returns Map, works internally using a ThreadLocal)

How can I see all parameters passed into the Action?

ActionContext.getParameters() (returns Map, works internally using a ThreadLocal)

How can I get the HttpServletRequest?

ServletActionContext.getRequest() (works internally using a ThreadLocal)

How can I use the IOC container to initialize a component in another object that isn't an action?

Obtain the ComponentManager from the request: ComponentManager cm = (ComponentManager)

ServletActionContext.getRequest().getAttribute("DefaultComponentManager");

then you need to initialize it using: cm.initializeObject(Object)

How do I decouple XWork LocalizedTextUtil global resource bundle loading from servlets (ServletContextListener)?

If you're using XWork outside a Web context, then use whatever startup hooks you have in that context (i.e. application start for a desktop app) to add the global resource bundle. This is a startup activity, so use whatever mechanisms are provided in the context you're running in.

What I need to do to put values in a combobox. If I am using webwork2?

If i have :

```
#tag(Select "label='xxx '" "name='xxx'" "list=?")
or
#tag(combobox "label='Prioridade'" "name='inavis.avisTpPrioridade'" "list=?")
```

the values in this combobox, what i need to do?

Exemple:

```
html tag i use to do:

<select..>
  <otpion value="" selected>XXX</option>
</selct>
```

so...i need to do this using Webwork tags from Velocity...how can i do this??

How do I add I18N to a UI tag, like ww:textfield?

```
<ww:textfield label="'i18n.label'" name="'label1'" value="''">
```

This will get the localized text message for the key "i18n.label" and put it in the label.

```
<ww:textfield label="getText('i18n.label')" name="'label1'" value="''">
```

Alternatively, you could modify controlheader.vm and copy it to /template/xhtmll. There you could make it so that it automatically does a call to `$stack.findValue("getText($parameters.label)")`, making the first example actually work for i18n.

Can I add I18N outside the Action's context? i.e. adding i18n to some JSP using the ww taglib?

Yes, use the `<ww:i18n>` tag to push a resource bundle on to the stack. Now calls with `<ww:text/>` or `<ww:property value="getText(...)"/>` will read from that resource bundle.

Can I break up my large XWork.xml file into smaller pieces?

Sure, that's what the `<include>` element is for. Most xwork.xml files already have one:

```

<xwork>
  <include file="webwork-default.xml"/>
  <include file="config-browser.xml"/>
  <package name="default" extends="webwork-default">
    ....
  </package>
  <include file="other.xml"/>
</xwork>

```

This tells it to load the webwork-default.xml from the webwork jar file to get all of those interceptor and result definitions.

You can put your own <include> in your xwork.xml interchangeably with <package> elements... They will be loaded in the same order as it reads from top to bottom and adds things as it reads them.

How can I put a String literal in a Javascript call, for instance in an onChange attribute?

The problem is in escaping quotes and getting the double quotes around the final value, like we expect in HTML attributes. Here's an example of the right way to do this (thanks to John Brad):

```

onChange=' "someFunc(this.form, \'abc\')"'

```

Notice here that there are single quotes surrounding the double quotes, and then the single quotes inline in the Javascript are escaped. This produces this result:

```

onChange="someFunc(this.form, 'abc')"

```

Why won't the 'if' tag evaluate a one char string?

```

<ww:if test="#myObj.myString == 'A'">
  Why doesn't this work when myString is equal to A?
</ww:if>

```

OGNL will interpret 'A' as a char type and not a string. Simple solution - flip the double and single quotes.

```
<ww:if test='#myObj.myString == "A"'>
This works!
</ww:if>
```

Alternatively, you can escape the double quotes in the String:

```
<ww:if test="#myObj.myString == \"A\"">
This works!
</ww:if>
```

I'm trying to run the webwork example in the tutorial on Tomcat, and it can't instantiate the VelocityEngine

Tomcat says:

```
javax.servlet.ServletException: Servlet.init() for servlet webwork threw exception at
org.apache.catalina.core.StandardWrapper.loadServlet(StandardWrapper.java:963)
```

...

root cause

```
java.lang.RuntimeException: Unable to instantiate VelocityEngine!
```

at

```
com.opensymphony.webwork.views.velocity.VelocityManager.newVelocityEngine(VelocityManager
```

at

```
com.opensymphony.webwork.views.velocity.VelocityManager.init(VelocityManager.java:146)
```

at

```
com.opensymphony.webwork.dispatcher.ServletDispatcher.init(ServletDispatcher.java:177)
```

at

```
org.apache.catalina.core.StandardWrapper.loadServlet(StandardWrapper.java:935)
```

Solution: (thanks to Keith Lea)

It turns out Velocity's Avalon logging system was trying to write to my tomcat folder.

So that it's on file somewhere for other people, I will describe the solution:

I created a file "velocity.properties" and placed it in my WEB-INF/classes folder. Inside

the file I wrote:

```
runtime.log.logsystem.class=org.apache.velocity.runtime.log.NullLogSystem
```

This stops velocity from logging, and makes webwork work again.

How do I get the latest version of WebWork

This page last changed on Oct 30, 2005 by [plightbo](#).

Overview

This page last changed on Oct 30, 2005 by [plightbo](#).

1. [What is WebWork](#)
2. [Comparison to other web frameworks](#)
3. [Articles and press](#)
4. [Projects using WebWork](#)
5. [Testimonials](#)

WebWork is a very popular framework and community. As such, there are many articles, presentations, and books about WebWork. Here is just a sample.

Books

- [WebWork in Action](#) - Patrick Lightbody, Jason Carreira; Manning; September 2005
- [Java Open Source Programming](#) - Joseph Walnes, Ara Abrahamian, Mike Cannon-Brookes, Patrick Lightbody; Wiley; November 2003
- [Art of Java Web Development](#) - Neal Ford; Manning; November 2003
- [WebWork Live](#) - Matthew Porter; SourceBeat; N/A

Presentations

- [WebWork + AJAX: A winning combination](#) - Patrick Lightbody; JavaZone; August 2005
- [WebWork in Action: An introduction to WebWork](#) - Patrick Lightbody; JavaZone; August 2005
- [WebWork 2.0: Strutting the OpenSymphony way](#) - Jason Carreira; TheServerSide Symposium; April 2004
- [Strutting the OpenSymphony way](#) - Mike Cannon-Brookes; TheServerSide Symposium; July 2003
- [WebWork 2.0 Overview](#) - Rick Salsa; [Groove Systems](#)

Articles

- [Working wit the WebWork Framework](#) - Vlad Kofman
- [Building with WebWork](#) - Kris Thompson; TheServerSide; November 2003
- [Tutorial and article in Brazilian Portuguese](#) - January, 2004

Blogs

The official WebWork Blog can be found [here](#).

Additionally, the blogs of the developers of WebWork may provide some useful information:

- [Blogbody](#) - Patrick Lightbody
- [Jason Carreira](#)

Blog entries:

- [A summary of what is new in WebWork 2.0](#) - Mike Cannnon-Brookes

WebWork: Strutting the OpenSymphony way – Mike Cannon-Brookes

- [An overview of Mike's thoughts from the conference](#)
- [Mike's original PPT](#)

[WebWork](#) is a pull-based MVC framework focused on componentization and code reuse. It is currently in beta, but is being used by several opensource projects and a few commercial projects in development. This is the second generation of WebWork, which was originally developed by Rickard Oberg, and in this release, what was WebWork has been broken into two projects, [XW:XWork](#) and [WebWork](#).

XWork is a generic command pattern implementation with absolutely NO ties to the web. XWork provides many core services, including interceptors, meta-data based validation, type conversion, a very powerful expression language (OGNL – the Object Graph Notation Language) and an Inversion of Control (IoC) container implementation.

WebWork provides a layer on top of XWork to do HTTP request / response handling. It includes a ServletDispatcher to turn HTTP requests into calls to an Action, session and application scope mapping, request parameter mapping, view integration with various web view technologies (JSP, Velocity, FreeMarker, JasperReports), and user interface components in the form of JSP tags and Velocity macros wrapped around reusable UI components.

An Action is the basic unit of work in WebWork. It is a simple command object that implements the Action Interface, which has only one method: `execute()`. Action implementers can extend the ActionSupport class, which provides i18n localization of messages (with one ResourceBundle per Action class and searching up the inheritance tree) and error message handling including class level and field level messages.

Actions can be developed in one of two styles: Model driven or field driven. Model driven Actions expose a model class via a `get` method, and the form fields refer directly to the model properties using expressions like `"pet.name"`. XWork uses Ognl (the Object Graph Notation Language) as its expression language, and when

rendering the page, this expression will translate to `getPet().getName()`. When setting properties, this will translate to `getPet().setName()`. This style of development allows for a great deal of model reuse and can allow you to directly edit your domain objects in your web pages, rather than needing a translation layer to form beans. Field driven Actions have their own properties which are used in the view. The action's `execute()` method collates the properties and interacts with the model. This can be very useful when your form and model are not parallel. Even in this case, the powerful expression language in WebWork can allow you to compose your form fields into aggregate beans, such as an address bean, which you can reuse to simplify your action classes.

WebWork allows you to build your own reusable UI components by simply defining a Velocity template. This is how the pre-built components of WebWork are built for common components such as text fields, buttons, forms, etc. and made available from any view type (either JSP or Velocity at the moment). These components are skinnable by defining multiple templates for the same component in different paths. If your components include the default header and footer templates that are used in the pre-built templates, then they will inherit the ability to automatically handle displaying error messages beside the problem form field. These custom UI components are especially handy for reusing templates which handle your custom model types or for things like date pickers, which Mike showed as an example.

Interceptors in XWork allow common code to be applied around (before and/or after) action execution. This is what Mike calls "Practical AOP". Interceptors help to decouple and componentized your code. Interceptors can be organized into stacks, which are lists of interceptors to be applied in sequence, and can be applied to actions or whole packages. Much of the core functionality of XWork and WebWork is implemented as Interceptors. The common basic examples of Interceptors are timing and logging, and these are built in with XWork. Mike went through an example of an interceptor to identify users of events via email. This interceptor has its own external configuration file which specifies which users are interested in which events, and it compares this configuration with the action invocations passing through it to determine if any messages should be sent.

XWork's validation framework allows for decoupled validation of action properties. It is implemented as an Interceptor and reads external XML files which define the validations to be applied to the Action. Error messages are loaded from the Action's localized messages and flow through to the UI. Validator classes can be plugged in to

add to the set of bundled validators. The bundled validators include required field and required String validators, range validators for Dates and numbers, and email and URL validators. XWork also includes expression validators at both the Action and field level which allow you to use any Ognl expression as the validation.

Inversion of Control (IoC) removes the burden of managing components from your code and puts it on the container. The container takes care of managing component lifecycle and dependencies. EJB is an example of IoC, but with limited services. IoC promotes simplicity and decoupling of your components and encourages your classes to be smaller and more focused. Unit testing is also simplified, as you can just supply MockObject instances of the services your code depends upon during testing. XWork and WebWork provide a web-native IoC container which manages component dependencies. In WebWork IoC is implemented as lifecycle managers (SessionLifecycleListener, etc) and an Interceptor. There are 4 component scopes in WebWork IoC: Application, HTTP Session, HTTP Request, and Action invocation. IoC in XWork / WebWork is purely optional, so you can use it if you want it.

XWork / WebWork allows for sets of Actions and their views to be bundled as a jar file and reused. Your main xwork.xml file can include the xml configuration file of the jar file because they are included from the classpath. Similarly, if your views are [Velocity](#) templates, you can bundle your views in the jar file and they will be loaded from the classpath when rendering. This allows for componentization of your application and reuse of bundled Actions across applications.

I have to admit, when Mike mentioned this feature, I thought he was crazy. I didn't say anything at the session, but asked him about it later, and he said "didn't you write the package include stuff?" I'll take it as a good sign that things can be used in a different way than was imagined. 😊

Mike finished up with a comparison of WebWork vs. [Struts](#) . Struts is obviously the 500 lb gorilla in the web MVC space, so why use WebWork? WebWork's pros include being a smaller, simpler framework, not having to build ActionForm beans, making it very simple to test your Actions, having multiple well-supported view technologies, simpler views with less JSP tags and a more powerful expression language, not having to make your Actions thread-safe, not having your Actions tied to the web, and not being part of [Jakarta](#) . WebWork also adds many new features such as Interceptors,

packages, IoC, etc. WebWork's cons include being a smaller project with fewer books and less tool support, having less standards support for specs like JSTL and JSF, and not being part of [Jakarta](#) .

Comparison to other web frameworks

This page last changed on Aug 03, 2005 by [plightbo](#).

- [Comparison to Struts](#)
- [Comparison to JSF](#)
- [Comparison to Tapestry](#)
- [Comparison to Spring MVC](#)
- [Comparison to Ruby on Rails](#)

Comparison to JSF

This page last changed on Oct 30, 2005 by [plightbo](#).

TODO: a brief write-up comparing WebWork to JSF

Comparison to Ruby on Rails

This page last changed on Oct 30, 2005 by [plightbo](#).

WebWork's architecture is very similar to Ruby on Rails. The biggest difference between WebWork and Rails is actually more of a difference between Java and Ruby than anything. Using [FreeMarker](#) and [QuickStart](#), developers can achieve the same level of productivity as developers who use Rails and the fact that Ruby is a scripting language.

Comparison to Spring MVC

This page last changed on Oct 30, 2005 by [plightbo](#).

TODO: a brief write-up comparing WebWork to Spring MVC

Feature Comparison

Feature	Struts	WebWork 1.x	WebWork 2.x
Action classes	Struts requires Action classes to extend an Abstract base class. This shows a common problem in Struts of programming to abstract classes instead of interfaces.	Action classes must implement the webwork.Action Interface. There are other Interfaces which can be implemented for other services, such as storing error messages, getting localized texts, etc. The ActionSupport class implements many of these Interfaces and can act as a base class. WebWork is all written to Interfaces, which allows for plugging in your own implementations.	An Action must implement the com.opensymphony.xwork.Action Interface, with a series of other Interfaces for other services, like in WebWork 1.x. WebWork2 has its own ActionSupport to implement these Interfaces.
Threading Model	Struts Actions must be thread-safe because there will only be one instance to handle all requests. This places restrictions on what can be done with Struts Actions as any	WebWork Actions are instantiated for each request, so there are no thread-safety issues. In practice, Servlet containers generate many throw-away objects per request, and one	ditto


	resources held must be thread-safe or access to them must be synchronized.	more Object does not prove to be a problem for performance or garbage collection.	
Servlet Dependency	Struts Actions have dependencies on Servlets because they get the ServletRequest and ServletResponse (not HttpServletRequest and HttpServletResponse, I've been told) when they are executed. This tie to Servlets (although not Http*) is a defacto tie to a Servlet container, which is an unneeded dependency. Servlets may be used outside a Web context, but it's not a good fit for JMS, for instance.	WebWork Actions are not tied to the web or any container. WebWork actions CAN choose to access the request and response from the ActionContext, but it is not required and should be done only when ABSOLUTELY necessary to avoid tying code to the Web.	ditto
Testability	Many strategies have sprung up around testing Struts applications, but the major hurdle is the fact that Struts Actions are so tightly tied to the web (receiving a Request and Response object). This often	WebWork actions can be tested by instantiating your action, setting the properties, and executing them	ditto, but the emphasis on Inversion of Control makes testing even simpler, as you can just set a Mock implementation of your services into your Action for testing, instead of having to set up

	<p>leads people to test Struts Actions inside a container, which is both slow and NOT UNIT TESTING.</p> <p>There is a Junit extension : Struts TestCase</p> <p>(http://strutstestcase.sourceforge.net/)</p>		<p>service registries or static singletons</p>
FormBeans	<p>Struts requires the use of FormBeans for every form, necessitating either a lot of extra classes or the use of DynaBeans, which are really just a workaround for the limitation of requiring FormBeans</p>	<p>WebWork 1.x allows you to have all of your properties directly accessible on your Action as regular Javabeans properties, including rich Object types which can have their own properties which can be accessed from the web page. WebWork also allows the FormBean pattern, as discussed in "WW1:Populate Form Bean and access its value"</p>	<p>WebWork 2 allows the same features as WebWork 1, but adds ModelDriven Actions, which allow you to have a rich Object type or domain object as your form bean, with its properties directly accessible to the web page, rather than accessing them as sub-properties of a property of the Action.</p>
Expression Language	<p>Struts 1.1 integrates with JSTL, so it uses the JSTL EL. This EL has basic object graph traversal, but relatively weak collection and indexed property support.</p>	<p>WebWork 1.x has its own Expression language which is built for accessing the ValueStack. Collection and indexed property support are basic but good. WebWork can also be made to work directly with</p>	<p>WebWork 2 uses XW:Ognl which is a VERY powerful expression language, with additions for accessing the value stack. Ognl supports very powerful collection and indexed property support. Ognl also</p>

		JSTL using the Filter described in WW1:Using JSTL seamlessly with WebWork	supports powerful features like projections (calling the same method on each member of a collection and building a new collection of the results), selections (filtering a collection with a selector expression to return a subset), list construction, and lambda expressions (simple functions which can be reused). Ognl also allows access to static methods, static fields, and constructors of classes. WebWork2 may also use JSTL as mentioned in WW1:Using JSTL seamlessly with WebWork
Binding values into views	Struts uses the standard JSP mechanism for binding objects into the page context for access, which tightly couples your view to the form beans being rendered	WebWork sets up a ValueStack which the WebWork taglibs access to dynamically find values very flexibly without tightly coupling your view to the types it is rendering. This	ditto

		allows you to reuse views across a range of types which have the same properties.	
Type Conversion	<p>Struts FormBeans properties are usually all Strings. Struts uses Commons-Beanutils for type conversion. Converters are per-class, and not configurable per instance. Getting a meaningful type conversion error out and displaying it to the user can be difficult.</p>	<p>WebWork 1.x uses PropertyEditors for type conversion. PropertyEditors are per type and not settable per Action, but field error messages are added to the field error map in the Action to be automatically displayed to the user with the field.</p>	<p>WebWork2 uses Ognl for type conversion with added converters provided for all basic types. Type converters default to these converters, but type conversion can be specified per field per class. Type conversion errors also have a default error message but can be set per field per class using the localization mechanism in WW2 and will be set into the field error messages of the Action.</p>
Modular Before & After Processing	<p>Class hierarchies of base Actions must be built up to do processing before and after delegating to the Action classes, which can lead deep class hierarchies and limitations due to the inability to have multiple inheritance</p> <p>WW:Comparison to</p>	Class hierarchies	<p>WebWork 2 allows you to modularize before and after processing in Interceptors. Interceptors can be applied dynamically via the configuration without any coupling between the Action classes and the Interceptors.</p>

	Struts#1		
Validation	<p>Struts calls validate() on the FormBean. Struts users often use Commons Validation for validation. I don't know a lot about this, so I'll put some questions here:</p> <p>Because FormBean properties are usually Strings, some types of validations must either be duplicated (checking type conversion) or cannot be done?</p> <p>Can Commons Validation have different validation contexts for the same class? (I've been told yes, so that's a good thing)</p> <p>Can Commons Validation chain to validations on sub-objects, using the validations defined for that object properties class?</p>	<p>WebWork1.x calls the validate() method on Actions, which can either do programmatic validations or call an outside validation framework (this is apparently the same as Struts)</p>	<p>WebWork2 can use the validate() method of WebWork and Struts and / or use the XW:Validation Framework, which is activated using an XWork Interceptor. The Xwork Validation Framework allows you to define validations in an XML format with default validations for a class and custom validations added for different validation contexts. The Xwork Validation Framework is enabled via an Interceptor and is therefore completely decoupled from your Action class. The Xwork Validation Framework also allows you to chain the validation process down into sub-properties using the VisitorFieldValidator which will use the validations defined for the properties</p>

			class type and the validation context.
Control Of Action Execution	As far as I know Struts sets up the Action object for you, and you have very little control over the order of operations. To change them I think  you need to write your own Servlet to handle dispatching as you want	The ActionFactory chain controls the order in which an Action is constructed and initialised, but this requires writing a class	The interceptor stacks in WebWork 2 are hugely powerful in this regard. All aspects of Action setup have been moved into Interceptor implementations (ie setting parameters from the web, validation etc), so you can control on a per action basis the order in which they are performed. For example you might want your IOC framework to setup the action before the parameters are set from the request or vice versa - you can thusly control this on a per package or per action basis with interceptor stacks.

References

- <http://www.mail-archive.com/opensymphony-webwork@lists.sourceforge.net/msg00995.htm>
- compares Struts development to WebWork 1.x development from the point of view of a Struts developer who switched to WebWork
- <http://www.mail-archive.com/opensymphony-webwork@lists.sourceforge.net/msg04700.htm>
- Kind of the first draft of this comparison

Footnotes

1. Some Struts users have built the beginnings of an Interceptor framework for Struts (<http://struts.sourceforge.net/saif/>). It currently has some serious limitations (no "around" processing, just before and after) and is not part of the main Struts project

Comparison to Tapestry

This page last changed on Oct 30, 2005 by [plightbo](#).

TODO: a brief write-up comparing WebWork to Tapestry

Projects Using WebWork

This page last changed on Jun 11, 2005 by [plightbo](#).

- [Atlassian Confluence](#) - Commercial Wiki and knowledge management system using WebWork 2.0, Hibernate, Spring, and Velocity
- [Jive Software](#) - With over 1100 customers, Jive Software's Forums and Knowledge Base products both use WebWork 2.1+ and are some of the largest deployments of WebWork
- [Midwinter](#) - an open source rapid web application develop system using WebWork 2.0, Hibernate, Spring, and Velocity
- [Chemist Australia](#) - Online Pharmacy
- [DriveNow](#) - last minute Australian car rentals
- [OpenReports](#) - an open source web based reporting application that uses WebWork 2.0, Velocity, and Hibernate
- [eSage Group](#) is a consulting company that uses it for all their client engagements. Additionally, its used for their internal systems
- [Filmweb](#) - Polish Film Portal
- [TeraMEDICA](#) - WebWork is used in TeraMEDICA's commercial TI2m product, which performs intelligent image management for the healthcare enterprise. Specifically, WebWork is a key component of the system's management interface.
- [EBIA COBRA and 401K Benefits Site](#) provides law reviews for employee benefits like COBRA and 401K. The site moved from all struts to a current architecture of about 50% WebWork and 50% Struts. We are trying to move it all over to WebWork. This site is also a great example of porting from Tiles to SiteMesh.
- [Valtira Rolecall](#) - Identity Management Framework that provides single sign-on for J2EE and .NET web applications.
- [Orange Blossom Indian River Citrus](#) - Retail site of shipper of fresh Florida (USA) citrus.
- [JavaEye Reporting Tool](#) - An open-source, web-based database reporting tool. It allows you to create reports without any programming (though you'll need SQL

knowledge). It's a lightweight reporting environment, the report can be created to quickly share information via web.

- [Dating-site](#) - Commercial dating site (BE, dutch) ww2.1.7, hibernate
- [No Fluff, Just Stuff](#) - Java/Open Source Conferences delivered locally.

Testimonials

This page last changed on Oct 30, 2005 by [plightbo](#).

WebWork rocks! We use it for our [Bug Tracking](#) and for several of our clients. We have moved several sites from Struts to WebWork. I love it. Another site we work with for [Survey Software](#) is also moving off of Struts to WebWork. Everything is easier in WW, especially with the power of Interceptors!!!

Again... another site in 1 week with Site Mesh and WebWork. Its a [blog community](#) site. The one I manage... being a cyclist is the [Cycling Community](#)

Mike Porter, Architect, eSage Group

<http://www.esagegroup.com>

Two years ago we dediced to use WebWork instead of Struts because of it's technical superiority and it proved to be an excellent decision. WebWork is successfully used by productive customer applications running with WebLogic and Tomcat. A major project will be migrated to the newest XWork/WebWork versions in the next 6 months. Besides it's technical advantages, XWork/WebWork has a smart and extremely skilled developer team and a healthy community.

Lars Fischer, Project Manager, Computata AG Switzerland

<http://www.compudata.ch>

WebWork is a very versatile web framework. After using solutions ranging from home-grown to Struts, WebWork is truly a breath of fresh air. XWork/WebWork not only used advanced techniques and technology, but brought concepts to the table that actually made development easier. These include built-in IOC, easy to use Spring integration, and null-property handling, and of course, type conversion.

I'm definitely looking forward to utilizing the newest features in my future projects. Keep up the good work!

Jay Bose, Sr. Engineer, Notiva Corporation

<http://www.notiva.com>

WebWork is a powerful web-based MVC framework built on top of a command pattern framework API called [XWork](#). The true power of Webwork is its underlying concept of simplicity and interoperability. Using WebWork will help minimize code and allow developers to concentrate more on business logic and modeling, rather than the plumbing often required when building web-based applications.

Features

- A flexible [Validation](#) framework allowing you to decouple validation rules from your action code.
- [Type Conversion](#) allowing you to easily convert objects from one class to another, solving one of the most tedious efforts when creating web apps.
- A powerful **Expression Language** based on [OGNL](#) allowing dynamic object graph traversal and method execution and transparent access to properties from multiple beans using a ValueStack. Webwork also has the ability to use [JSTL](#).
- [Inversion of Control](#) integration that manages component lifecycle and dependencies without the need to build registry classes that clients must call to obtain a component instance. WebWork recommends [Spring](#) for IoC.
- Reusable [Tags and UI Components](#) that allow for easy and reusable component-oriented web development.
- Advanced [Interceptors](#) that provide for various rich functionality, including preventing multiple form submissions and executing long running queries in the background.
- Hierarchical and pluggable support for [Internationalization](#).
- Easy integration with third party software including [Hibernate](#), [Spring](#), [Sitemesh](#), and [JSTL](#).
- Support for many view technologies such as [JSP](#), [FreeMarker](#), and [JasperReports](#).
- Modular [Configuration](#) using packages and namespaces to manage hundreds of actions.

Project Information

This page last changed on Oct 30, 2005 by [plightbo](#).

1. [License](#)
2. WebWork versions
 - Current release - [WebWork 2.2](#)
 - [Previous releases](#)
 - Migrating from WebWork 1.x
3. [Dependencies](#)
4. [WebWork Team](#)
5. [How to contribute](#)
6. Services
 - [User forums / mailing list](#)
 - [Developer forums / mailing list](#)
 - [Reporting issues](#)
 - [Documentation and wiki](#)

Dependencies

This page last changed on Oct 30, 2005 by [plightbo](#).

WebWork has only a few required dependencies but has many optional dependencies. You can determine what these dependencies are by looking in the **docs/dependencies** directory of the distribution, or by [clicking here](#). The only required dependencies are those in the **default** configuration. If you plan to use the UI tags, you will also need the jars in the **freemarker** configuration (unless you plan to write all your UI templates from scratch in a different language, such as Velocity or JSP, which is not recommended).

Deployment Notes

This page last changed on Aug 30, 2005 by [plightbo](#).

TODO: shouldn't this be merged with the stuff in the [Cookbook](#)?

WebWork runs on most application servers without any problems. However, you may need to do a few modifications in order to get it running in your environment.

WebLogic 6.1

A subproject has been added to the WebWork CVS repository that vastly simplifies getting WebWork to work under BEA Weblogic Server 6.1. Documentation is included. Look for the subproject under the main folder "misc".

WebLogic 8.1

Seems to have some difficulty loading the Velocity templates. If you run into this problem, the work-around is documented [here](#).

SunONE 7.0

You need to grant permissions to WebWork:

```
grant {  
    permission java.security.AllPermission;  
};
```

or more specifically,

- Give Write Permissions to java.util.PropertyPermission.
- Add java.lang.reflect.ReflectPermission "suppressAccessChecks"
- OgnlInvoke Permission

```
grant {  
    permission java.util.PropertyPermission "*", "read, write";  
    permission java.lang.reflect.ReflectPermission "suppressAccessChecks";  
};
```

```
    permission ognl.OgnlInvokePermission "*" ;  
};
```

Previous releases

This page last changed on Sep 21, 2005 by [plightbo](#).

- Release Notes
 - [WebWork 2.2](#) - Upcoming
 - [WebWork 2.3](#) - Upcoming
 - [WebWork 2.1.7](#)

Old format of release notes and upgrade guides

- Release Notes
 - [Release Notes - 2.1.6](#)
 - [Release Notes - 2.1.5](#)
 - [Release Notes - 2.1.4](#)
 - [Release Notes - 2.1.3](#)
 - [Release Notes - 2.1.2](#)
 - [Release Notes - 2.1.1](#)
 - [Release Notes - 2.1](#)
- Upgrading from previous versions
 - [Upgrading from 2.1.5](#)
 - [Upgrading from 2.1.4](#)
 - [Upgrading from 2.1.3](#)
 - [Upgrading from 2.1.2](#)
 - [Upgrading from 2.1.1](#)
 - [Upgrading from 2.1](#)
 - [Upgrading from 2.0](#)
 - [Upgrading from 1.4](#)

Key Changes

- JavaScript client validation support - not totally complete, but basic validators work well. Look at the validators.xml file include in src/example to see how you can configure your validators to do client side validation on top of their normal duties
- The label attribute in UI tags are no longer required
- The themes and templates in UI tags behave like they did in 1.x
- A new theme, in addition to the existing "xhtml" one, called "simple" is included that doesn't have any of the labels, error reporting, or table rows that the "xhtml" template has. This is more in line with the tags included with Struts.
- New UI tags for CSS styles and classes added: cssStyle and cssClass
- Old action!command URL support works again. This means you can invoke a doCommand() method like in 1.x
- ww:param tag no longer requires the name attribute (for ordered params, like with ww:text). It also evaluates the the body as the value if no value is given.
- UI tags now have access to the FormTag parameter map using the "form" key. This means \$parameters.form.name would return the form name, for example. The result is that complex JavaScript-based components can be built.


Migration Notes

Version	Description	Old Code	New Code
2.0	WebWorkUtil has been refactored into a number of classes, and the constructor has changed. If you were using it for Velocity support before, look at VelocityWebWorkUtil now		

2.0	The <i>webwork.ui.templateDir</i> configuration property has been broken into <i>webwork.ui.templateDir</i> and <i>webwork.ui.theme</i>	<pre>webwork.ui.templateDir = /webwork/mytheme</pre>	<pre>webwork.ui.templateDir = /webwork webwork.ui.theme = mytheme</pre>
2.0	"namespace" attribute of the <code>ww:action</code> tag is now evaluated; those upgrading from 2.0 will need to place single quotes around the attribute value	<pre><ww:action namespace="/foo" .../></pre>	<pre><ww:action namespace="'/foo'" .../></pre>
2.0, but not 1.x	theme and template attributes in UI tags have changed are now evaluated; those upgrading from 2.0 will need to place single quotes around the attribute value	<pre><ww:xxxx theme="/template/foo" template="bar.vm"/></pre>	<pre><ww:xxxx theme="'foo'" template="'bar.vm'"/></pre>
1.x, 2.0	label UI tag evaluates the value attribute now instead of the name attribute	<pre><ww:label name="'Foo'"/></pre>	<pre><ww:label value="'Foo'"/></pre>

Changelog

OpenSymphony JIRA (25 issues)		
T	Key	Summary
	WW-592	Upgrade commons-logging
	WW-560	SessionMap holds on to requests when it doesn't need to
	WW-546	Make the config-browser show validators applied via the XML validation files
	WW-544	Velocity result hardcodes contenttype and encoding
	WW-541	Webpage link for download
	WW-537	Velocity tag outputs to the response, not the velocity writer
	WW-530	Config Browser doesn't work after lates ActionConfig refactoring
	WW-519	ActionTag should evaluate namespace attribute
	WW-518	Label attribute shouldn't be required
	WW-517	Themes and templates should behave like 1.x
	WW-516	Simple theme that has no tables and xhtml extends from
	WW-515	Class attribute is illegal
	WW-514	Form tag double evaluates name attribute
	WW-503	Fix tag libraries
	WW-502	foo!default.action should work
	WW-501	JavaScript-based client side validation

	WW-500	ww:param tag fixes
	WW-499	UI tags should have access to form
	WW-488	Check QuickStart Guide to make sure it works
	WW-487	WebWorkConversionErrorInterceptorTe in wrong branch
	WW-484	label tag problems
	WW-478	URLTag tld entry does not correspond with actual property
	WW-476	WebWork needs a simple changelog for each release
	WW-475	Multipart encoding still not fixed
	WW-474	Ability to dynamically create array of Objects from a given request

WebWork 2.1.1 Release Notes

Key Changes

- Improved integration with Sitemesh
 - WebWork taglibs can be used in Sitemesh decorators to access Action properties
- Validator short-circuiting to allow validation to stop on first invalid data
- Improved class hierarchy resource bundle searching
- File upload support has been rebuilt to allow for multiple files with the same HTTP parameter name. Besides "cos" and "pell" support, "jakarta" support has been added, utilizing the Commons-FileUpload library. Only "jakarta" supports multiple files with the same HTTP parameter name. In future versions "jakarta" may become the default upload library, replacing "pell".











Migration Notes

Version	Description	Old Code	New Code
2.1	There is a new validator DTD: xwork-validator-1.0.2.dtd. You aren't required to use this, but you will need to if you wish to use the new short-circuiting validation	N/A	N/A
2.1	File upload support has been rebuilt, although we don't see any compatibility problems with 2.1.	N/A	N/A

	However, many of the methods in MultiPartRequest have become deprecated in favor of new ones. Please switch to these as soon as possible.		
--	---	--	--

Changelog





WebWork 2.1.1

OpenSymphony JIRA (25 issues)		
T	Key	Summary
	WW-926	Type conversion fails with ModelDriven actions
	WW-925	Document the config-browser in Related Tools
	WW-923	Fix up validation documentation
	WW-922	Add CaveatEmptor example
	WW-921	Defect in com.opensymphony.webwork.views.ut
	WW-920	Missing attribute for ww:form tag
	WW-919	Bug in DefaultActionMapper (could be Weblogic specific)
	WW-917	HttpServletRequest locale/encoding problem
	WW-916	Complete Architecture section of Documentation
	WW-915	Complete Introduction

		section of Documentation
	WW-913	Alt Syntax Migration - Page Specific altSyntax Change
	WW-912	ww:a support for preInvokeJS
	WW-911	Fix velocity code bug
	WW-910	Update the IOC section to reflect Spring intergration.
	WW-909	Document FreeMarker simple map change
	WW-908	Client side validation?
	WW-907	ww:a does not support nested param tags
	WW-905	Update Scope Interceptor Documentation
	WW-904	Update Execute and Wait Interceptor Documentation
	WW-903	Update File Upload Interceptor Documentation
	WW-902	Update Chaining Interceptor Documentation
	WW-901	Complete Conversion Error Interceptor Documentation
	WW-900	Complete Prepare Interceptor Documentation
	WW-899	Complete Servlet Config Interceptor Documentation
	WW-898	Complete Workflow Interceptor Documentation

Xwork 1.0.2

OpenSymphony JIRA (15 issues)		
T	Key	Summary
	XW-210	Make default type conversion message a localized text that can be overridden
	XW-205	missing xwork 1.0.2 dtd in jar and website and typo in ValidationInterceptor
	XW-204	TextProvider.getText() should look in child property files
	XW-203	Add "trim" parameter to string validators
	XW-202	Integer and Float conversion dont work in CVS HEAD
	XW-200	i18n broken when the name of the text to find starts with a property exposed by the action
	XW-195	Add interface XWorkStatics which contains XWork-related constants from WebWorkStatics
	XW-194	Patch to help LocalizedTextUtil deal with messages for indexed fields (collections)
	XW-193	InstantiatingNullHandler and Typeconversion fails
	XW-192	Create a version 1.0.2 of the XWork validation DTD with short circuit
	XW-191	Type conversion improvement.

	XW-190	Provide a xwork-default.xml.
	XW-189	Improve ActionValidationManager's short circuit behaviour
	XW-179	Optimise OgnlUtil.copy method
	XW-172	XWorkBasicConverter doesn't care about the current locale

WebWork 2.1.2 Release Notes




Key Changes

- This version ships with XWork 1.0.3 – we recommend you make sure you are running this version (or later) of XWork.
- Minor bug fixes for file upload support with Jakarta
- New StreamResult type which allows you to stream content directly back from an action
- UI tags may now be written in languages other than Velocity. JSP is supported, though you currently must write your own templates similar to the Velocity templates. Future versions of WebWork will include more languages supported as well as templates shipped out of the box.

Migration Notes

Migration should require nothing more than copying over the new libs. Specifically note that XWork 1.0.3 and WebWork 2.1.2 should be copied over.

Changelog

OpenSymphony JIRA (14 issues)		
T	Key	Summary
	WW-642	Allow the 'name' attribute of the TextTag to be evaluated at runtime
	WW-639	"Could not open template ", possible a bug
	WW-634	File Upload Interceptor stack



	WW-633	Jakarta File Upload fails with mixed content (normal and file)
	WW-630	upload newest webwork files to ibiblio
	WW-629	checkboxlist doesn't have a disabled attribute tag
	WW-628	Bug with request parameter handling with WebLogic 8.1sp3
	WW-624	If/Else tag do not render body
	WW-622	The changelog for WW2.1.1 shows open issues not the closed ones!
	WW-616	ww:label and ww:textarea problem with null values
	WW-612	WebworkStatistics.SERVLET DISPATCH is spelled incorrectly
	WW-611	Error in Freemarker docs
	WW-602	Stream Result Type
	WW-485	Add docs for WebWork2 tags

WebWork 2.1.3 Release Notes

Key Changes

WebWork version 2.1.3 resolves a critical problem preventing UI tags from working under certain circumstances. It is recommended that all users use 2.1.3 in place of 2.1.1 or 2.1.2

Changelog

OpenSymphony JIRA (2 issues)		
T	Key	Summary
	WW-659	VelocityTemplateEngine broken
	WW-321	ActionMessage as the companion of ActionError

WebWork 2.1.4 Release Notes

Key Changes

This release is the first release to include the re-vamped JSP tag support. Specifically, there is an option (off by default) that now lets you use an alternative syntax for JSP tags.

When **webwork.tag.altSyntax** is set to true in `webwork.properties`, all attributes in the JSP tags (both UI and non-UI) that evaluate to a String (as opposed to a Boolean, Integer, Collection, or anything else) shall not be evaluated like it normally is.

Rather, the string will be parsed for the pattern `"%{...}"` and only the text between those braces shall be evaluated. This should make using all the tags, but especially the UI tags, much easier.

The only exception to this rule of parsing String attributes is for the `<ww:property/>` tag. That is because the usage for `<ww:property/>` is so commonly used for pulling values from the stack, enforcing the `"%{..}"` syntax on it would be overly tedious.

Migration Notes


There is nothing to migrate for now. Because this new syntax is optional and turned off by default, you don't need to do anything to migrate as long as you don't plan to use this new syntax. If you DO plan to use this new syntax, you must modify all your tag attributes that previously had the pattern of `"..."` to just be `"..."`. Basically, you must remove single quotes where they once were.

Also, any place where an attribute did not have single quotes, you must check to see if the attribute is expected to be a String attribute. If so, you should replace the pattern of `"..."` with `"%{...}"` so that the expression is still being evaluated.

Note that attributes such as disabled, maxlength, etc do not need the new syntax. **That is because the syntax is only applied to attributes that are expected to be strings.** This is very important to remember!

Finally, please note that this release is a preview release for the new syntax. The new syntax will not be finalized and turned on by default until 2.2.0. You are free to use it, but it is not guaranteed to be stable and may change in the coming releases.

Changelog

OpenSymphony JIRA (1 issues)		
T	Key	Summary
	WW-581	JSP Tags should support better syntax

WebWork 2.1.5 Release Notes





Key Changes

- All UI tags now support the complete set of JavaScript event listeners now, such as onChange, onBlur, etc.
- ExecuteAndWaitInterceptor is easier to use
- Several important bug fixes for Velocity integration

Migration Notes

Version	Description	Old Code	New Code
2.1.4 and below	TLD updated – be sure you are using the latest webwork.tld file	N/A	N/A

Changelog

OpenSymphony JIRA (18 issues)		
T	Key	Summary
	WW-666	ExecAndWaitInterceptor should put executing action on the stack
	WW-663	VelocityResult doesn't initialize VelocityManager
	WW-660	xhtml's checkbox.vm vertical alignment
	WW-658	JSP Tags do not support onFocus, onBlur js handlers





	WW-653	url taglib does not support 'page' attribute but the webwork-example.war uses it all over the place.
	WW-651	IfTag does not convert to Boolean
	WW-644	Xhtml generated by the ui tags is (still) invalid
	WW-632	OnClick for radiotag
	WW-627	select.vm requires htmlEncode for name parameter
	WW-626	Principal Interceptor
	WW-614	Cannot set velocity macro autoreloading
	WW-554	Bad value for IMAGES_URI in JasperReportsResult.java
	WW-526	Velocity using include ignores character encoding
	WW-479	getValueClassType() in ComboboxTag returns Boolean.class
	WW-425	Refactoring to decouple the UI Tag dependency on velocity
	WW-392	Update doubleselect tag
	WW-223	Multipart & SaveDir
	WW-161	ServletDispatcher could be a lot simpler to extend

WebWork 2.1.6 Release Notes

Key Changes

This release includes a few bug fixes for the URL tag, a new base class to make type conversion easier, and better documentation. It also includes the latest XWork release: version 1.0.4

Changelog

OpenSymphony JIRA (4 issues)		
T	Key	Summary
	WW-673	Create a WebWorkTypeConverter for extension
	WW-671	URLTag does not include parameters when value is specified
	WW-664	Document WebFlow
	WW-585	ComboBoxTag should subclass TextFieldTag

Package changes

Webwork1.x was separated into two projects, XWork and Webwork. From this, several classes have been moved to different package names.

- ActionSupport has moved from **webwork.ActionSupport** to **com.opensymphony.xwork.ActionSupport**
 - doExecute() no longer exists, override execute()
 - the methods addError and addErrorMessage are now addFieldError and addActionError respectively

Configuration changes

- **actions.xml/views.properties needs to be converted to xwork.xml**

If you're using an actions.xml file to configure your webwork 1, you can use the attached XSLT to convert the actions.xml file to a vanilla xwork.xml file.

To apply this XSLT, you'll need to do the following:

Get a copy of the XSLT. You can find the latest version in CVS in webwork/src/etc/actions.xsl . Next, find yourself an XSLT rendering engine. Xalan is a good choice and can be found at <http://xml.apache.org/xalan-j/index.html>

Finally, do the conversion.

```
java org.apache.xalan.xslt.Process -IN actions.xml -XSL actions.xsl -OUT xwork.xml
```

Remember that you'll need to Xalan libraries in your classpath to run the above command.

If you want to look at these pages directly in your browser, I recommend user Internet Explorer as it automagically formats XML documents reasonably. There one caveat though. WW1 had a way to shorten the declaration of actions by allowing you to specify a package prefix in webwork.properties file. Since this information is outside the actions.xml file, the XSLT is unable to take advantage of it. Consequently, you might need to edit the xwork.xml file to update the class names.

WebWork 1.x configuration used a pull paradigm to load action configurations when

they are asked for, whereas WebWork2 builds the configuration up-front to make the configuration queryable. The `webwork.MigrationConfiguration` must therefore act as an adapter between these two paradigms. It does this by returning a custom `RuntimeConfiguration` which first tries the default `XWork Configuration` (which, by default, loads configuration information from a file named "xwork.xml" in the root of the classpath) and then attempts to load action configuration using the `Configuration` classes from WebWork 1.x. In this way, an application can be slowly converted over to WebWork2 while reusing the configuration and Actions from a WebWork 1.x application. One caveat in this is that your migrated application **MUST** be rebuilt against the WebWork2 and migration jar files, as the classloader will rightly recognize that the `webwork.Action` and `webwork.ActionSupport` in WebWork 1.x are not the same as those provided by the migration jar files. Other than that, it should be seamless (and let us know if it isn't).

If the `webwork.MigrationRuntimeConfiguration` does not find the action configuration using the `RuntimeConfiguration` from the supplied `RuntimeConfiguration` (which is acquired from the `Xwork DefaultConfiguration` and will load configurations from all of the sources configured for `XWork / WebWork2`), it will build an `ActionConfiguration` by instantiating an Action using the `ActionFactories` from WebWork 1.x. The `ActionFactory` stack used is a subset of the default `ActionFactory` stack used in WebWork 1.x:

```
factory = new JavaActionFactory();
factory = new ScriptActionFactoryProxy(factory);
factory = new XMLActionFactoryProxy(factory);
factory = new PrefixActionFactoryProxy(factory);
factory = new JspActionFactoryProxy(factory);
factory = new CommandActionFactoryProxy(factory);
factory = new AliasingActionFactoryProxy(factory);
factory = new CommandActionFactoryProxy(factory);
factory = new ContextActionFactoryProxy(factory);
```

Some of the `ActionFactory` classes have been left out as they are handled by `Interceptors` in WebWork2. If the Action instance is created (meaning that the configuration has been found in the `webwork.properties` or `actions.xml` files used by the WebWork 1.x configuration classes) a parameter `Map` is created by introspecting the Action instance. A `Map` is needed for results and, again, WebWork 1.x used a pull paradigm to find results when they were needed, so a `LazyResultMap` is created which extends `HashMap` and overrides `get()` to look up the `Result` configuration if it has not previously been loaded. If the result ends in the Action suffix (defaulting to ".action"), then a `ChainingResult` is created, otherwise a `ServletDispatcherResult` is created.

Using the Action class of the instantiated Action, the Map of parameters introspected from the Action instance, and the LazyResultMap, a new ActionConfig is created. The ActionConfig is saved into a special Package, "webwork-migration", so that it will pick up the default Interceptor stack defined for that package. The "webwork-migration" package is defined in a webwork-migration.xml file which is included in the migration jar file and which is automatically added to the Xwork configuration providers when the MigrationConfiguration is used:

```
<!DOCTYPE xwork PUBLIC "-//OpenSymphony Group//XWork
1.0//EN" "http://www.opensymphony.com/xwork/xwork-1.0.dtd"><xwork><include
file="webwork-default.xml"/><package name="webwork-migration" abstract="true"
extends="webwork-default"><interceptors><interceptor-stack
name="migrationStack"><interceptor-ref name="timer"/><interceptor-ref
name="logger"/><interceptor-ref name="chain"/><interceptor-ref
name="static-params"/><interceptor-ref name="prepare"/><interceptor-ref
name="params"/><interceptor-ref
name="workflow"/></interceptor-stack></interceptors><default-interceptor-ref
name="migrationStack"/></package></xwork>
```

Here we can see that a number of the functions previously performed by ActionFactories in WebWork 1.x are now replaced by Interceptors in WebWork2, including the parameters, the chaining, calling prepare(), and the workflow (which was formerly implemented in ActionSupport in WebWork 1.x).

By creating and saving the ActionConfig in the PackageConfig for the "webwork-migration" package, the ActionConfig for the migrated Action is available for future calls, obviating the need to re-parse the old configuration files and making the configuration for the Action available for querying via the configuration API for tools such as the configuration browser.

Tag Changes

The biggest change is the use of OGNL for accessing object properties. Properties are no longer accessed with a forward slash "/" but with a dot "." Also, rather than using ".." to traverse down the stack, we now use "[n]" where n is some positive number. Lastly, in WebWork 1.x one could access special named objects (the request scope attributes to be exact) by using "@foo", but now special variables are accessed using "#foo". However, it is important to note that "#foo" does NOT access the request attributes. "#foo" is merely a request to another object in the OgnlContext other than

the root. See [OGNL](#) reference for more details.

Also see [JSP Expression Language Comparison with WebWork 1.x](#) for a table of the expression language changes.

[property](#)

The property tag is now only used to print out values from the stack. In WW1, it was also used to set a variable in the scope, and to push properties to the top of the stack. These functions are now performed by the [set](#) and [push](#) tags.

[action tag](#)

The action tag does not evaluate the body section any more and does not push the executed action onto the ValueStack. Instead, use the "**id**" attribute to assign a name to the action and reference it as "**#id**".

Examples

Lets enumerate some examples of differences between code snips using [WW:WebWork](#) and [WW:WebWork](#).

- *New JSP syntax*

There are numerous changes in syntax. First of all there are new tags and secondly there is a new expression language. Here's a small example:

Webwork 1

```
<ww:property value="a/b"><ww:property value="foo" /></ww:property>
```

Webwork 2

```
<ww:push value="a.b"><ww:property value="foo" /></ww:push>
```

One can note that the "push" tag doesn't just push it pops too at the end of the tag. Surprise! Also note the "." instead of the "/" for traversing object properties.

- *List errors posted by an Action*

Webwork 1

```
<webwork:if test="hasErrorMessages == true">
  ERROR:<br /><font color="red"><webwork:iterator
value="errorMessages"><webwork:property/><br
/></webwork:iterator></font></webwork:if>
```

Webwork 2

```
<webwork:if test="hasErrors()">
  ERROR:<br /><font color="red"><webwork:iterator
value="actionErrors"><webwork:property/><br
/></webwork:iterator></font></webwork:if>
```

Update your web.xml file

- If you're using Velocity for views, you'll need to make sure you have the following snippet. Specifically note that the `<load-on-startup>` tag is now required so that the servlet can initialize some important Velocity properties.

```
<servlet>
  <servlet-name>velocity</servlet-name>
  <servlet-class>com.opensymphony.webwork.views.velocity.WebWorkVelocityServlet</servlet-class>
  <load-on-startup>1</load-on-startup>
</servlet>
```

- Set the property **webwork.velocity.configfile** in your `_webwork.properties_`. For example:

```
webwork.velocity.configfile=velocity.properties
```

WebWork will use this file to initialize the Velocity engine. The search path for the file is:

1. context root (web root)
2. WEB-INF/

3. classpath

- Additional Steps:

1. If you used the `<ww:action` taglib in 1.3... you used to reference the java Action classname. In 2.x this reference is now the action name not the class. you will need to change all your old references in your view.

ResultException doesn't exist anymore

It might be possible to copy WW1's `ResultException`, and write an `Interceptor` that catches the `ResultExceptions` and add the result of `getMessage()` to the `actionErrors` of the executed Action and return `ResultException.getResult()`.

Maybe it would be possible to include `ResultException` in WW2 too to make migration easier?!

DateFormatter doesn't exist anymore

It can be replaced by directly using **`java.text.DateFormat`**

`addError(String, String)` in `webwork.action.ActionSupport` has been removed

The new method to use is **`addFieldError(String, String)`**.

`addErrorMessage(String)` in `webwork.action.ActionSupport` has been removed

The new method is now **`addActionError(String)`**.

`webwork.util.ValueStack` has been removed

The ValueStack is **com.opensymphony.xwork.util.OgnlValueStack**

The old methods **pushValue** and **popValue** are renamed to simply **push** and **pop**.

An instance of the ValueStack can be obtained by using **ActionContext.getContext().getValueStack** instead of the old **ValueStack.getStack()**.

***Aware-Interfaces have been removed**

Instead of implementing **ServletRequestAware** etc the **[Servlet]ActionContext.getXXX**-methods can be used to obtain application-map, request, response etc.

CommandDriven interface removed

The **CommandDriven** interface is removed. It is not necessary to implement a special interface when working with commands anymore. Use the **method** attribute in your **action**-Element in xwork.xml to tell xwork which method to invoke on your action.

isCommand(String) method has been removed

You can see which alias you're accessing by doing this:
ActionContext.getContext().getActionInvocation().getProxy().getActionName()

JSP Expression Language Comparison with WebWork 1.x

This page last changed on Apr 12, 2005 by [plightbo](#).

Situation	Previous (WW-1.4)	Current (WW-2.1)
Referring to an object in the PageContext scope	@itemIdOrName	#attr['itemIdOrName']
Referring to an object in the Request scope	itemIdOrName	Same, but use #request['itemIdOrName'] if nested in an iteration.
Referring to an object in the Session scope	@itemIdOrName	#session['itemIdOrName']
Referring to an object in the Application scope	@itemIdOrName	#application['itemIdOrName']
Property Setters	foo/bar translates to getFoo().setBar()	foo.bar translates to getFoo().setBar()
Property Getters	foo/bar translates to getFoo().getBar()	foo.bar translates to getFoo().getBar()
Boolean/boolean Property Getters	foo/bar translates to getFoo().getBar() if bar is java.lang.Boolean, if primitive bar translates to getFoo().isBar()	Same, except uses dot notation instead of a slash (i.e. foo.bar)
Collections as Properties	N/A	Collections (including arrays) are similar to other objects, except they allow indexing: foo.bar[indexOrKeyName] translates to getFoo().getBar().get(indexOrKeyName)
curly braces - {}, evaluates contents of braces first, and use the result as the property to then evaluate.	<webwork:property value="{ 'name' }"/> translates to getName() on the curent object.	No longer used.
Reference a static variable	@com.fully.qualified.class.name.staticVariable	Still The Actual Class@STATIC_ATTRIBUTE

Originally written by Jay Bose and sent to the mailing list

Upgrading from 2.0

This page last changed on Dec 14, 2004 by [plightbo](#).

Upgrading from Webwork 2.0 is rather trivial. This version of webwork adds enhancements and bug fixes with hardly any configuration or syntax changes. Follow these two simple steps and you should be on your way with the latest and greatest from the OS crew.

1. Update/Replace your current binaries with the new binaries located in the distribution download under the `lib/core`. You may also want to grab any related jars from the `lib/optional` folder. Don't forget the webwork binary `webwork-2.1.jar` in the base directory of the distribution download. Review the [Dependencies](#) for Webwork.
2. Check out the [Release Notes - 2.1](#) to see if any of changes need to be applied to your code base.

Upgrading from 2.1

This page last changed on Dec 14, 2004 by [plightbo](#).

Upgrading from 2.1 to 2.1.1 is very easy. Simply copy over the new webwork.jar file and make sure you are running all the correct [Dependencies](#).

Upgrading from 2.1.1

This page last changed on Dec 14, 2004 by [plightbo](#).

Upgrading from 2.1.1 to 2.1.2 is very easy. Simply copy over the new webwork.jar file and make sure you are running all the correct [Dependencies](#) – especially make sure you are using XWork 1.0.3.

Upgrading from 2.1.2

This page last changed on Dec 14, 2004 by [plightbo](#).

Upgrading from 2.1.2 to 2.1.3 is very easy. Simply copy over the new webwork.jar. There have been no new dependencies.

Upgrading from 2.1.3

This page last changed on Dec 14, 2004 by [plightbo](#).

Since this release is merely a preview release of the new tag syntax (see [Release Notes - 2.1.4](#)) and no other changes have been made, simply copying over the new webwork jar file is all that is needed to upgrade. No other libraries have changed.

Upgrading from 2.1.4

This page last changed on Dec 14, 2004 by [plightbo](#).

Upgrading from 2.1.4 is pretty easy as no new libraries have been introduced. However, the TLD (taglib definition file) has been modified to support more JavaScript events, such as onBlur, onChange, etc. Be sure to check that you are using the latest TLD. This is automatic if you point web.xml to **/WEB-INF/webwork.jar**.

Upgrading from 2.1.5

This page last changed on Dec 14, 2004 by [plightbo](#).

Upgrading from 2.1.5 to 2.1.6 requires that you copy over the new webwork-2.1.6.jar and xwork-1.0.4.jar. Note that WebWork 2.1.6 includes a new version of XWork, version 1.0.4.

WebWork 2.1.7 release notes

Key changes

- XWork upgraded to 1.0.5
 - Using `i18n`, especially the `ww:text` tag, is now possible in SiteMesh decorators. See [SiteMesh](#).
 - Issues where the `ActionContext` wasn't properly cleaned up (after `ww:include` and `ww:action`) have been resolved.
 - Configuration
 - If the action class is not specified, it now defaults to `com.opensymphony.xwork.ActionSupport`.
 - If the result name is not specified, it is assumed to be "success".
- Non-UI tags
 - `ww:text` and `ww:url` have an `id` attribute that, when specified, causes the tag not to print out the localized text but rather store the result in the `ActionContext` to be referenced later. See [SiteMesh](#), [URL tag](#), and [Text Tag](#).
- UI tags
 - `ww:form` has an added `target` attribute.
 - `ww:form` exposes "namespace" in the parameters Map in templates.
 - `ww:form` tag defaults the `id` and `name` attribute to the action name that it is submitting to.
 - Form elements default the `id` attribute to "`[formId]_[elementName]`".
 - Form elements default the "required" attribute to true if there is any validator associated with that field and the form's `validate` attribute is enabled.
 - `ww:textarea` has an added "wrap" attribute.
 - XHTML theme: added a parameter called "after" and that will add text to the right of the form element.
- Replaced old prototype client-side validation with XMLHttpRequest-based solution (STILL IN PROTOTYPE PHASE, works in XHTML theme only).
- Interceptors
 - `ServletConfigInterceptor` now checks for actions implementing `PrincipalAware` and adds a `Principal` that ties in to the `HttpServletRequest`'s `Principal`.
 - Minor bug fixes in the `ExecuteAndWaitInterceptor` and `FileUploadInterceptor`.
- Results
 - `JasperReportsResult` and `StreamResult` both support the `Content-disposition` header.
 - `JasperReports` integration has been upgraded to 0.6.3.

- `WebWorkResultSupport` has a simple `conditionalParse()` method making support for `${}` notation in your results easier (it will parse only if the `parse` attribute is true).





Upgrade steps















1. Copy over the new `webwork.jar` and `xwork.jar` files.
2. If you are using `JasperReports`, you will need to upgrade to the latest `JasperReports` jars of 0.6.3 because the package name has changed.
3. Read the migration notes for anything that you should be aware of.

Migration notes

- If you were using the old client-side validation code, it is still possible to keep using it but you will likely need to modify `form-close.vm` and `controlheader.vm` in the `XHTML` theme and `form.vm` in the `simple` theme. We recommend supporting the new prototype as it is a lot easier to use and doesn't require special validators.
- It is possible, though very unlikely, that the default ids for the UI tags and the default name for the form tag might cause you some trouble. Please be aware of this change.

Changelog

OpenSymphony JIRA (26 issues)		
T	Key	Summary
	WW-706	Text tag and URL tag should have option to store contents to context
	WW-704	Integrate PrincipalInterceptor with ServletConfigInterceptor
	WW-703	Add wrap attribute to textarea
	WW-702	Required (*) should be on by default if a validator

		exists
	WW-701	UI tags should have default ids and names
	WW-698	webwork.i18n.encoding does not get set by the ServletDispatcher
	WW-696	StreamResult should accept file option
	WW-695	JSP Form Tags does not have a target attribute
	WW-694	ExecuteAndWaitInterceptor can return null results
	WW-693	FileUploadInterceptor don't check the allowed files's Enumeration which is null.
	WW-692	Webwork package of jasper reports packaged as dori.jasper. Latest release of Jasper Reports packaged as net.sf.jasperreports
	WW-691	JasperReports 0.6.3 support
	WW-687	Freemarker - add method buildUrl to FreemakerWebworkUtil
	WW-686	JasperReport result - Content-Disposition management
	WW-684	Default action class and result name.
	WW-681	getText() doesn't work in Sitemesh filters
	WW-677	ww:include replaces existing value stack with new one
	WW-676	Freemarker Support:TemplatePath in

		web.xml has no effect
	WW-668	Externalise the JavaScript validation support from the JSP taglibs
	WW-649	Field errors should be displayed in the order they're in the POJO
	WW-638	Freemarker result encoding error
	WW-637	textarea does not have maxlength attribute
	WW-617	Stale Action Invocation left in Stack Context
	WW-490	Is WW ignoring webwork.locale setting?
	WW-455	Select tag template does not work properly for Object like BigDecima
	WW-351	Forwarding unknown tag attributes

WebWork 2.2 Release Notes

Key Changes

Productivity enhancements

Tools

- Fully functional WebFlow support for JSP, FreeMarker, and Velocity (Patrick)

Documentation

- Improved documentation, including detailed information for every interceptor (Everyone)
- ~~Totally new example application: now the example application is a set of tutorials and only teaches best practices of WebWork, rather than every single feature. (Patrick)~~

Enhanced framework feedback

- ~~Better and more intelligent error reporting (Patrick)~~
- ~~"Developer" mode where inline errors are displayed when possible (Patrick)~~
- ~~Common interceptor stack issues, such as validation+workflow with no "input" result, are now reported in a more obvious way (Jason)~~

Other

- ~~Deprecated WebWork IoC container in favor of Spring~~
- ~~Built-in support for Spring~~
- ~~Official support for wizards/workflows using the [Scope Interceptor](#) and a pre-release of [Continuations](#) (Patrick)~~
- ~~Removed support for WebWork 1.x migration jar (Patrick)~~

User interface improvements

UI tag overhaul

- ~~FreeMarker is now the default UI tag implementation (Patrick)~~
- ~~Refactored UI tag base classes such that they are no longer tied to JSP (Patrick)~~
- ~~New native Velocity and FreeMarker UI tag support, built on top of new base classes (Patrick)~~
- ~~UI tags now use "altSyntax" (available since 2.1.4) as the default syntax (the 2.0 - 2.1 syntax is deprecated but still available) (Patrick)~~

Velocity Support Improvements

- ~~Upgraded support to velocity 1.4 (Andres)~~
- ~~webwork.velocity.contexts now chains contexts on each request, i.e. contexts do not need to be thread-safe (Andres)~~

AJAX support

- ~~Official support for client-side validation using DWR (Patrick)~~
- ~~New tabbed panel widget (Ian)~~
- ~~Built-in support for Dojo widgets~~

Result changes

- ~~Velocity and FreeMarker Servlets are now deprecated in favor of direct results (Patrick)~~

Other

- ~~Easy way to invoke different action name or command, making forms with multiple buttons easy to use (Patrick)~~
- ~~Initial JSR168 integration (Henry Hu)~~

Core API changes

Type conversion

- Typing support in Maps, Sets, and Lists is now supported even when the collection is not null (Gabe)
- Map type conversion support for keys and values (Gabe)
- Support for Java 5 generics and annotations for Collections and enums (Gabe?)





Other


- Improved exception handling, with support for exception-to-result mapping in xwork.xml (Matthew Porter)
- Parameters interceptor updated to let you include and/or exclude certain parameters, thereby providing a simple way to secure what data can be changed from the web (Bob)



Migration Notes

WebWork 2.2 is the most significant release since the 2.0 release two years ago. There are some significant changes, deprecated items, and various issues to be aware of when upgrading or if you're just curious what is new. **Please see the [WebWork 2.2 Migration Notes](#) for more info.**

Changelog

OpenSymphony JIRA (25 issues)		
T	Key	Summary
	WW-926	Type conversion fails with ModelDriven actions
	WW-925	Document the config-browser in Related Tools
	WW-923	Fix up validation documentation
	WW-922	Add CaveatEmptor example

	WW-921	Defect in com.opensymphony.webwork.views.utl
	WW-920	Missing attribute for ww:form tag
	WW-919	Bug in DefaultActionMapper (could be Weblogic specific)
	WW-917	HttpServletRequest locale/encoding problem
	WW-916	Complete Architecture section of Documentation
	WW-915	Complete Introduction section of Documentation
	WW-913	Alt Syntax Migration - Page Specific altSyntax Change
	WW-912	ww:a support for preInvokeJS
	WW-911	Fix velocity code bug
	WW-910	Update the IOC section to reflect Spring intergration.
	WW-909	Document FreeMarker simple map change
	WW-908	Client side validation?
	WW-907	ww:a does not support nested param tags
	WW-905	Update Scope Interceptor Documentation
	WW-904	Update Execute and Wait Interceptor Documentation
	WW-903	Update File Upload Interceptor Documentation
	WW-902	Update Chaining Interceptor Documentation
	WW-901	Complete Conversion Error Interceptor Documentation
	WW-900	Complete Prepare

		Interceptor Documentation
	WW-899	Complete Servlet Config Interceptor Documentation
	WW-898	Complete Workflow Interceptor Documentation

This page last changed on Oct 27, 2005 by [jcarreira](#).

This document covers a step-by-step guide for upgrading to WebWork 2.2 from 2.1.x, as well as a list of the key individual changes for reference.

Upgrade Guide

1. Get the [latest 2.2 release](#)
2. Check out the [dependencies](#) to see what the required libraries are. One change of note is the dependency on Rife-Continuations. Click through the tabs for the dependencies for different usage profiles. If you use FreeMarker for instance, click on that tab to see those dependencies. Note that if you use the JSP tags you are now using FreeMarker by default for the UI component templates.
3. Check the **Individual Changes** section below to see if any of those changes affect your code
4. Update to use the **FilterDispatcher** instead of the **ServletDispatcher**. Check out the [web.xml 2.1.x compatibility](#) page for some compatibility discussions, and see [web.xml](#) for what needs to go in the *web.xml* file.

Individual Changes

Version	Description	Old Code	New Code
2.1.x	<p>If you implemented your own ObjectFactory or ActionInvocation classes, you will notice that there have been some minor changes to make an "extraContext" Map available for the build* methods. This allows, for instance, access to the Session map during object creation, even before the ActionContext ThreadLocal has been set.</p>	ObjectFactory.getObjectFactory().buildObject(extraContext);	ObjectFactory.buildObject(extraContext);
2.0+	<p>If you've used the WebWork base classes for building templated tags, you'll run into the refactoring of the UI tags to use common Component classes as the templated back-end. The tags now use these Component classes, as do</p>	...your code..	See the existing UI tags in the 2.2 source

	Velocity and FreeMarker. This allows Velocity and FreeMarker to use the same UI components directly, without pretending to be a JSP page, but it also means you need to refactor your custom tags to use the new API's		
2.1.x	If you were <i>not</i> using the altSyntax , it is now enabled by default. You can either upgrade or change the Tag Syntax	<ww:url value=""http://www.yahoo.com	<ww:url value=""http://www.yahoo.com
2.1.x	If you are using FreeMarker and your code uses psuedo properties on collections and maps, you need to modify the code to call methods instead.	\${parameters?size}	\${parameters.size()}}
2.1.x	The defaultStack has been renamed to the basicStack.	<interceptor-ref name="defaultStack"/>	<interceptor-ref name="basicStack"/>
2.1.x	The completeStack has been renamed to the defaultStack.	<interceptor-ref name="completeStack"/>	<interceptor-ref name="defaultStack"/>
2.1.x	The defaultStack	N/A	N/A

	(previously the completeStack) is now the default interceptor stack in webwork-default.xml.		
2.1.x	The component interceptor has been deprecated (along with all WebWork IOC features) and has been removed from the basicStack and completeStack. You'll need to add it back by hand if you wish to use this deprecated feature.	N/A	N/A
2.0+	The include tag's page attribute has been deprecated since 1.x and is now removed from 2.2. Please use the value attribute.	<ww:include page="..."/>	<ww:include value="..."/>
2.0+	The text tag's value0, value1, value2, and value3 attributes have been deprecated since 1.x and are now removed from 2.2. Please use the param tag instead.	<ww:text value0="..." />	<ww:text><ww:param>...</w
2.0+	The VUI tags have been removed from	N/A	N/A

	WebWork. They haven't been actively worked on in over 4 years are not used in the community.		
--	--	--	--

WebWork 2.3 Release Notes

Key Changes

Productivity enhancements

Tools

- IDE plugins for Eclipse and IDEA (Ricardo)
- Enhanced WebFlow support using new features introduced in [WebWork 2.2](#) (Patrick)

Other

- Improved support for action chaining and how it interacts with validation (Jason)

User interface improvements

AJAX support

- New AJAX widgets?

Other

- Better JSP 2.0/JSTL integration
- Native support for checkboxes, even when they aren't checked
- Improved JSR 168 support

Core API changes

Configuration

- API updated to allow for multiple deployments in a single classloader (Jason)
- Removal of many statics and overall cleaner API (Jason)
- All configuration-related files (xwork-conversion.properties, xwork.xml, etc) can easily be looked up from any path (Jason)
- Support for Java 5 annotations (Rainer, Nils)

Type conversion

- Support for Java 5 generics and annotations for Collections and enums (Gabe?)
- Support for Java 5 annotations (Rainer, Nils)






Validation








- Easier support for typical validation GET/POST lifecycle (see [AppFuse](#)) (Jason)
- Support for Java 5 annotations (Jason, Rainer, Nils)







Migration Notes

Version	Description	Old Code	New Code
---------	-------------	----------	----------

Changelog

OpenSymphony JIRA (25 issues)		
T	Key	Summary
	WW-914	Create new ww:errors tag
	WW-838	Create a new ww:css tag
	WW-830	PROTOTYPE: Add configuration options for flexibility
	WW-809	Expression Support Properteis Configuration
	WW-805	ww:date tag

	WW-803	JFreeReport Result
	WW-799	OgnlValueStackDataSource Field Name X Description Bug
	WW-796	SetTag "scope" attribute do not put the object to stack if the scope is given.
	WW-795	Can't override default webwork messages
	WW-794	Issue with ServletDispatcherResponse after a JasperException
	WW-793	Message with key "webwork.internal.invalid.token" is never used
	WW-785	Freemarker JSP Taglibs fails if action has param named "Request"
	WW-772	Allow to radio tag to generate only one radio button
	WW-771	Add an "errorStyleClass"-like attribute to form element tags
	WW-770	Validation should be aware of namespace that action lives in
	WW-769	UITags do not evaluate id attribute
	WW-764	URLBean improvements
	WW-739	Action tag TLD missing ignoreContextParams
	WW-733	Unmapped action results in a Server 500 error on Tomcat 5.x

	WW-731	Add the capability to automatically save messages between Actions
	WW-728	Provide a Status object in the select tag like the one in the Iterator tag
	WW-727	profiling in ww2
	WW-721	<ww:subset/> problem
	WW-711	combobox tag does not support listKey and listValue
	WW-688	Refactor TokenInterceptor and TokenHelper (and affected classes)

Reference

This page last changed on Oct 30, 2005 by [plightbo](#).

The Basics

1. [Architecture](#)
2. [Configuration](#)
3. [Action Configuration](#)
4. [Interceptors](#)
5. [Result Types](#)

UI-related Topics

1. [Tags and UI Components](#)
2. [OGNL](#)
3. View technologies:
 - a. [JSP](#)
 - b. [Velocity](#)
 - c. [FreeMarker](#)
 - d. [JasperReports](#)

Advanced Functionality

1. [Action Chaining](#)
2. [Inversion of Control](#) (IOC)
3. [Type Conversion](#)
4. [Validation](#)
5. [Internationalization](#)
6. [Continuations](#)
7. [ActionMapper](#)

Other

1. [Related Tools](#)
2. [J2SE 5 Support](#)
3. [3rd Party Integration](#)

3rd Party Integration

This page last changed on Oct 30, 2005 by [plightbo](#).

1. [Sitemesh](#)
2. [Spring](#)
3. [Pico](#)
4. [Hibernate](#)
5. [JSTL](#)
6. [JUnit](#)
7. [Quartz](#)

Hibernate

This page last changed on Jun 18, 2004 by [plightbo](#).

There's nothing more that you have to do use Hibernate with WebWork than with other Web framework. Just setup Hibernate according to the <http://www.hibernate.org/5.html>. However, there're a number of good patterns that people have used successfully in the following projects:

- AdminApp <http://www.hibernate.org/159.html#a5>
- Petsoar <http://www.wiley.com/legacy/compbooks/walnes>

JSTL

This page last changed on Oct 30, 2005 by [plightbo](#).

JSTL integration is built in to WebWork 2.2+ - there are no steps required to enable it. Simply refer to your JSTL expressions just as you would with a normal WebWork JSP tag, such as the property tag. This is accomplished a request wrapper called For more info, see the javadocs of the WebWorkRequestWrapper object:

Content pulled from external source. Click here to refresh.

All WebWork requests are wrapped with this class, which provides simple JSTL accessibility. This is because JSTL works with request attributes, so this class delegates to the value stack except for a few cases where required to prevent infinite loops. Namely, we don't let any attribute name with "#" in it delegate out to the value stack, as it could potentially cause an infinite loop. For example, an infinite loop would take place if you called: <code>request.getAttribute("#attr.foo")</code> .
--

There's a number of approaches you can take to unit-test your WebWork actions.

The simplest is to instantiate your actions, call setters then execute(). This allows you to bypass all the complicated container setup.

Taken from Petsoar:

```
package org.petsoar.actions.inventory;

import com.mockobjects.constraint.IsEqual;
import com.mockobjects.dynamic.C;
import com.mockobjects.dynamic.Mock;
import com.opensymphony.xwork.Action;
import junit.framework.TestCase;
import org.petsoar.pets.Pet;
import org.petsoar.pets.PetStore;

public class TestViewPet extends TestCase {
    private Mock mockPetStore;
    private ViewPet action;

    protected void setUp() throws Exception {
        mockPetStore = new Mock(PetStore.class);
        PetStore petStore = (PetStore) mockPetStore.proxy();

        action = new ViewPet();
        action.setPetStore(petStore);
    }

    public void testViewPet() throws Exception {
        Pet existingPet = new Pet();
        existingPet.setName("harry");
        existingPet.setId(1);

        Pet expectedPet = new Pet();
        expectedPet.setName("harry");
        expectedPet.setId(1);

        mockPetStore.expectAndReturn("getPet", C.args(new IsEqual(new Long(1))),
existingPet);
        action.setId(1);

        String result = action.execute();

        assertEquals(Action.SUCCESS, result);
        assertEquals(expectedPet, existingPet);
    }
}
```



```

        mockPetStore.verify();
    }

    public void testViewPetNoId() throws Exception {
        mockPetStore.expectAndReturn("getPet", C.ANY_ARGS, null);

        String result = action.execute();

        assertEquals(Action.ERROR, result);
        assertEquals(1, action.getActionErrors().size());
        assertEquals("Invalid pet selected.",
            action.getActionErrors().iterator().next());
        assertNull(action.getPet());
        mockPetStore.verify();
    }

    public void testViewPetInvalidId() throws Exception {
        action.setId(-1);
        testViewPetNoId();
    }
}

```

Test interceptors and/or result types

Check out the test suites in XWork/WebWork. These are pretty comprehensive and provide a good starting point. For example, this is how the **ParametersInterceptor** is tested:

```

public void testDoesNotAllowMethodInvocations() {
    Map params = new HashMap();
    params.put("@java.lang.System@exit(1).dummy", "dumb value");

    HashMap extraContext = new HashMap();
    extraContext.put(ActionContext.PARAMETERS, params);

    try {
        ActionProxy proxy = ActionProxyFactory.getFactory().
            createActionProxy("",
                MockConfigurationProvider.MODEL_DRIVEN_PARAM_TEST, extraContext);
        assertEquals(Action.SUCCESS, proxy.execute());

        ModelDrivenAction action = (ModelDrivenAction) proxy.getAction();
        TestBean model = (TestBean) action.getModel();

        String property = System.getProperty("webwork.security.test");
        assertNull(property);
    } catch (Exception e) {
        e.printStackTrace();
        fail();
    }
}

```

Note: these are not the ONLY ways so make your own judgement.

Pico

This page last changed on Dec 22, 2004 by [plightbo](#).

Pico is an Inversion of Control container available at <http://picocontainer.codehaus.org>. There have been several reports of integration between WebWork and Pico.

<http://www.nanocontainer.org/NanoWar+WebWork> contains more information on integrating WebWork and Pico/Nano. Note that the documentation here doesn't require you to create your own ObjectFactory.

The following class performs the glue between Quartz and WebWork:

```
package com.trantek.sit.action;

import com.opensymphony.xwork.ActionProxy;
import com.opensymphony.xwork.ActionProxyFactory;
import com.opensymphony.xwork.interceptor.component.ComponentInterceptor;
import org.quartz.Job;
import org.quartz.JobExecutionContext;
import org.quartz.JobExecutionException;
import java.util.HashMap;

public class WebWorkJob implements Job
{
    public void execute(JobExecutionContext context) throws JobExecutionException
    {
        try
        {
            HashMap ctx = new HashMap();
            ctx.put(ActionContext.PARAMETERS,
context.getJobDetail().getJobDataMap());
            ctx.put(ComponentInterceptor.COMPONENT_MANAGER, ???);
            ctx.put(???, ???);
            ServletDispatcher.createContextMap()
            ActionProxy proxy = ActionProxyFactory.getFactory().
                createActionProxy("", context.getJobDetail().getName(), ctx);

            proxy.execute();
        }
        catch (Exception e)
        {
            thrownew JobExecutionException(e);
        }
    }
}
```

To schedule webwork actions you simply create a job where

- the name of your job is the name of the WW action to execute (no ".action" suffix).
- all the parameters you want to send to the WW action is contained in the JobDataMap of the JobDetail

(the Quartz scheduler is setup as a servlet according to the javadocs of `org.quartz.ee.servlet.QuartzInitializerServlet`.)

The following code schedules an e-mail action:

```

Scheduler scheduler = StdSchedulerFactory.getDefaultScheduler();

JobDetail jobDetail = new JobDetail("email.send",
                                     scheduler.DEFAULT_GROUP, WebWorkJob.class);

Map m = jobDetail.getJobDataMap();
m.put("to", "me@bogusdomain.com");
m.put("subject", "quartz test");
m.put("body", "This is a quartz test, Hey ho");
m.put("smtpServer", "smtp.bogusdomain.com");
m.put("from", "quartz@bogusdomain.com");

SimpleTrigger trigger = new SimpleTrigger("myTrigger",
                                           scheduler.DEFAULT_GROUP,
                                           new Date(), null, 0, 0L);

scheduler.deleteJob("email.send", scheduler.DEFAULT_GROUP);
scheduler.scheduleJob(jobDetail, trigger);

```

This example is based on [WW1: Integrating Webwork and Quartz](#)

SiteMesh can be found at <http://www.opensymphony.com/sitemesh>

Integrating WebWork with SiteMesh is amazingly simple: you don't have to do anything in fact. WebWork stores all its value stack information in the request attributes, meaning that if you wish to display data that is in the stack (or even the ActionContext) you can do so by using the normal tag libraries that come with WebWork. That's it!

Passing data around

One thing to note is when you want to pass a value from a decorated page to a decorator using the `<ww:set>` tag, you need to specify a scope (request, session, application) if the decorated page is invoked directly (not a result of an action). By default if no action has been executed and no scope was specified, the set value will only be available from the same PageContext.

Localization

In WebWork 2.1.7, support was added that makes using i18n in decorators much easier. Now using the `<ww:text/>` tag works seamlessly. In the rare event where you need to reference an i18n string, use the id attribute as documented in the [Text tag](#).

An example of such situation is given below. Typically embedding i18n in to form elements would be done using the following:

```
<ww:textfield label="getText('com.acme.login.text')" name="'login'"/>
```

However, due to the way WebWork and SiteMesh work, you would need to separate the above code into two tags.

```
<ww:text id="login" name="'com.acme.login.text'"/><ww:textfield label="#login" name="'login'"/>
```

SiteMesh, Velocity, and WebWork

If you are using Velocity for your SiteMesh decorators, we recommend not using the WebWorkVelocityServlet or the SiteMeshVelocityServlet. In fact, we don't recommend you use any servlet at all. Instead, try creating a servlet that extends PageFilter and then using that servlet in place of PageFilter in web.xml. In your custom servlet, override the applyDecorator() method and then use WebWork's VelocityManager to access your decorator templates. You can use VelocityManager to create a default Context as well, which will include all the WebWork variables such as \$stack. Then simply add the Page object to the context and then you can access the page parts using \$page.title, \$page.body, etc.

Spring

This page last changed on Oct 03, 2005 by [jcarreira](#).

Spring is an, among other things, an Inversion of Control framework. As of WebWork 2.2, it is the only supported IoC container. You can find out more about Spring at <http://www.springframework.org>.



This section covers the only *supported* Spring integration technique. However, there are many other ways to tie in to Spring with WebWork. Please see [Other Spring Integration](#) for more info. Note that *none* of these other methods are currently supported and could change at any time!

Enabling Spring Integration

Turning on Spring support in WebWork is simply a matter of installing the latest Spring jars in to your classpath and then adding the following entry to [webwork.properties](#):

```
webwork.objectFactory = spring
```

If you want to change from the default autowiring mode, which is to auto-wire by name (i.e. to look for beans defined in Spring with the same name as your bean property), then you'll also need a setting for this in your [webwork.properties](#):

```
webwork.objectFactory.spring.autoWire = type
```

Options for this setting are:

name	Auto-wire by matching the name of the bean in Spring with the name of the property in your action. This is the
------	---

	default
type	Auto-wire by looking for a bean registered with Spring of the same type as the property in your action. This requires you to have only one bean of this type registered with Spring
auto	Spring will attempt to auto-detect the best method for auto-wiring your action
constructor	Spring will auto-wire the parameters of the bean's constructor

At this point, all objects will at least try to get created by Spring. If they cannot be created by Spring, then WebWork will create the object itself. Next, you'll need to turn on the Spring listener in web.xml:

```
<listener><listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
```

Sample Spring Configuration

At this point, you can add the standard Spring configuration at **WEB-INF/applicationContext.xml**. An example of this configuration is:

```
<?xml version="1.0" encoding="UTF-8"?><!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN" "http://www.springframework.org/dtd/spring-beans.dtd"><beans
default-autowire="autodetect"><bean id="personManager"
class="com.acme.PersonManager"/>
...
</beans>
```

Initializing Actions from Spring

Normally, in xwork.xml you specify the class for each action. When using the SpringObjectFactory (configured as shown above), this means that WebWork will ask Spring to create the action and wire up dependencies as specified by the default

auto-wire behavior. The SpringObjectFactory will also apply all bean post processors to do things like proxy your action for transactions, security, etc. which Spring can automatically determine without explicit configuration. For most usages, this should be all you need for configuring your actions to have services and dependencies applied.



We **strongly** recommend that you find declarative ways of letting Spring know what to provide for your actions. This includes making your beans able to be autowired by either naming your dependent properties on your action the same as the bean defined in Spring which should be provided (to allow for name-based autowiring), or using autowire-by-type and only having one of the required type registered with Spring. It also can include using JDK5 annotations to declare transactional and security requirements rather than having to explicitly set up proxies in your Spring configuration. If you can find ways to let Spring know what it needs to do for your action without needing any explicit configuration in the Spring *applicationContext.xml*, then you won't have to maintain this configuration in both places.

However, sometimes you might want the bean to be completely managed by Spring. This is useful, for example, if you wish to apply more complex AOP or Spring-enabled technologies, such as Acegi, to your beans. To do this, all you have to do is configure the bean in your Spring **applicationContext.xml** and then *change* the class attribute from your WebWork action in the *xwork.xml* to use the bean name defined in Spring instead of the class name.

Your *xwork.xml* file would then have the action class attributes changed, leaving it like

this:

```
<!DOCTYPE xwork PUBLIC "-//OpenSymphony Group//XWork
1.0//EN" "http://www.opensymphony.com/xwork/xwork-1.1.dtd"><xwork><include
file="webwork-default.xml"/><package name="default"
extends="webwork-default"><action name="foo"
class="com.acme.Foo"><result>foo.ftl</result></action></package><package
name="secure" namespace="/secure" extends="default"><action name="bar"
class="bar"><result>bar.ftl</result></action></package></xwork>
```

Where you have a Spring bean defined in your **applicationContext.xml** named "bar". Note that the *com.acme.Foo* action did not need to be changed, because it can be autowired.

Remember: **this is not required**. This is only needed if you wish to override the default behavior when the action is created in WebWork by decorating it with Spring-enabled interceptors and IoC that cannot be automatically determined by Spring. Keep in mind that WebWork's Spring integration will do standard IoC, using whatever auto-wiring you specify, even if you don't explicitly map each action in Spring. So typically you don't need to do this, but it is good to know how this can be done if you need to.

Other Spring Integration

This page last changed on Sep 21, 2005 by [plightbo](#).



This document is provided by the WebWork user community and does not represent the *supported* Spring integration methods. Please refer to [Spring](#) for documentation on the recommended integration.

I started out using the original WebWork documentation to get Spring to initialize Webwork actions, but it appears that a lot has changed since the days of WebWork 1.x. This will be my attempt to clarify some of those changes and to list the steps necessary to get the two to play nicely. Please comment with clarifications or corrections!

WebWork 1.x

(these are assumptions based on the 1.x [WW1:Spring Framework Integration](#) documentation)

It seems that the way you got Spring to initialize WebWork 1.x action classes was to add this line into your webwork.properties file:

```
webwork.action.factory=webwork.action.factory.SpringActionFactory
```

so that WebWork would know to use Spring's view of the world to create actions. The WebWork action classes would then need to be declared in the Spring applicationContext.xml file so that Spring would know directly of the action objects. Upon invocation of an action, WebWork would know to first use the SpringActionFactory to try and create an instance of the requested action which would ask Spring to create the object using its configuration. If there was no Spring definition of that action object, then WebWork would use it's normal instantiation methods to create that action. Well, things have changed slightly since WebWork 1.x.

WebWork 2

In WebWork 2 (the functionality actually exists in XWork), you specify relationships from action classes to other objects in XWork's `xwork.xml` file instead of Spring's `applicationContext.xml` file. So if you have an action class that utilizes a DAO, instead of having a bean definition like so in `applicationContext.xml`:

```
<bean id="myAction" class="com.ryandaigle.web.actions.MyAction"
singleton="false"><property name="DAO"><ref bean="myDAO"/></property><bean
id="myDAO" class="com.ryandaigle.persistence.MyDAO" singleton="true" />
```

you move the action definition to `xwork.xml` and keep the DAO definition in `applicationContext.xml` so that `xwork.xml` looks like:

```
<action name="myAction" class="com.ryandaigle.web.actions.MyAction"><external-ref
name="DAO">myDAO</external-ref><result name="success" type="dispatcher"><param
name="location">/success.jsp</param></result></action>
```

and `applicationContext.xml` looks like:

```
<bean id="myDAO" class="com.ryandaigle.persistence.MyDAO" singleton="true" />
```

Notice how there is the `external-ref` element in the action definition that points to an object that Spring is managing. There are several things that need to be in place for the `external-ref` to work, but I just wanted to give an overview of what has changed before going into the specific steps.

Steps for Configuring Spring/WebWork2 (XWork) Integration:

Get the files you need to externally resolve Spring beans. I've bundled them all here: <http://www.ryandaigle.com/pebble/images/webwork2-spring.jar> . They were originally spread between two JIRA issues filed against XWork 1.0 (see references below). This zip includes the source, the class files (so you can just include it in your classpath) and my example configuration files. Either extract the source files into your application, or put the file onto your classpath. (You may want to take the `applicationContext.xml` and `xwork.xml` files out, I don't know if they'll override your files... They're just there as an example configuration).

Now, let's get your XWork configuration file (xwork.xml) to resolve external references. XWork resolves external references (using the external-ref element) by utilizing an external reference resolver per package. You specify your external reference resolver as an attribute of the package element:

```
<package name="default" extends="webwork-default"
  externalReferenceResolver="com.atlassian.xwork.ext.SpringServletContextReferenceResolver">
```

This SpringServletContextReferenceResolver class reference is a class not part of the XWork distribution written as an extensions for XWork/Spring that I got from this JIRA issue filed against XWork addressing this Spring integration effort. (I have bundled it with the rest of the necessary files later on down for your convenience). This class will intercept all external-refs and resolve the references using Spring's context. There is also a SpringApplicationContextReferenceResolver included in the zip file that will allow you to resolve Spring references for applications not executing within the web context. But as this is a WebWork/Spring article, the servlet resolver is what we need to use.

Now we need to add the XWork reference resolver as part of the interceptor stack you're using. This will allow any references to be resolved (using the reference resolver you specified in the externalReferenceResolver attribute). This is how I've added that interceptor:

```
<interceptors><interceptor name="reference-resolver"
class="com.opensymphony.xwork.interceptor.ExternalReferencesInterceptor"/><interceptor-stack
name="myDefaultWebStack"><interceptor-ref name="defaultStack"/><interceptor-ref
name="reference-resolver"/></interceptor-stack></interceptors><default-interceptor-ref
name="myDefaultWebStack"/>
```

As I briefly outlined before, you can now reference Spring beans that your action classes need in xwork.xml:

```
<action name="myAction" class="com.ryandaigle.web.actions.MyAction"><external-ref
name="DAO">myDAO</external-ref><result name="success" type="dispatcher"><param
name="location">/success.jsp</param></result></action>
```

And that's all we have to do to xwork.xml to let XWork know how to resolve references to Spring's managed beans.

Now let's setup our web environment to properly notify Spring and our external reference resolver of the web context. We do this by adding two context listeners to your application's web.xml file:

```
<listener><listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
```

The first listener is Spring's that you would need independent of whether or not you were integrating with WebWork 2. The second listener is our external resolver's that will use the servlet context to retrieve Spring's application context. This is the link between WebWork and Spring.

At this point, we've set up Spring and our XWork reference resolver to work within a web context, and we've told XWork how to resolve external references to Spring. We're done! Fire it up and let me know if there are some steps I've missed or assumptions I've made that I shouldn't have.

References

- Here is my bundled source, class and example configuration files (that contains all the needed referenced files below); <http://www.ryandaigle.com/pebble/images/webwork2-spring.jar> .
- My searching started with the original WebWork 1.x + Spring documentation and comments on the Wiki; [WW1:Spring Framework Integration](#)
- The Wiki pointed me to the two JIRA issues that contained the source files for the reference resolvers:
- <http://jira.opensymphony.com/browse/XW-122> The "SpringExternalResolver.zip" attachment is the one needed for externally resolving Spring objects.
- <http://jira.opensymphony.com/browse/XW-132> The "xwork-springServletImpl.zip" attachment is the one needed for externally resolving Spring objects. It just contains some files missing from the original source.

Credits

Judging by the comments etc... of the JIRA issues filed against XWork, it appears that Ross Mason (of Atlassian?) is the man to thank for the external reference resolver code. And of course we have to thank the people of Spring and WebWork 2 for making this all possible.

Using the SpringObjectFactory

Rather than using an external reference resolver with releases of XWork from 1.0.1 and onwards, it's possible to use the SpringObjectFactory from the [xwork-optional](#) package. This uses Spring to wire up the dependencies for an Action before passing it to XWork. Each action should be configured within a Spring application context as a prototype (because XWork assumes a new instance of a class for every action invocation):

```
<bean name="some-action" class="fully.qualified.class.name"
singleton="false"><property name="someProperty"><ref
bean="someOtherBean" /></property></bean>
```

Within xwork.xml:

```
<action name="myAction" class="some-action"><result
name="success">view.jsp</result></action>
```

Notice that the XWork Action's class name is the bean name defined in the Spring application context.

The 1.1.3 release of the Spring/XWork integration library allows the user to configure everything in the **xwork.xml** file without needing to add extra entries to the **applicationContext.xml**. This is done by configuring the actions with the fully qualified class name (as if not using the SpringObjectFactory) It also added the ability to make use of constructor-based dependency injection without any further changes. The major caveat when using constructor-based DI is that objects passed in to the constructor must be unambiguous within the applicationContext (as is normally required by Spring) If there is any ambiguity, then you can still configure things the more traditional way, splitting the configuration of the action between **xwork.xml** and **applicationContext.xml** as described above.

One other advantage of the SpringObjectFactory approach is that it can also be used to load interceptors using the same sort of logic. If the interceptor is stateless, then it's possible to create the interceptor as a singleton instance, but otherwise it's best to create it as a Spring prototype.

In order to be used, the default ObjectFactory that XWork uses should be replaced with an instance of the SpringObjectFactory. The xwork-optional package ships with a ContextListener that does this, assuming that the Spring application context has already been configured.

ActionAutowiringInterceptor

Another alternative to using the SpringObjectFactory is to use the ActionAutowiringInterceptor. The interceptor will autowire any action class based on the autowire strategy defined. An advantage to using the interceptor over the SpringObjectFactory is that the action classes do not have to be defined in the Spring's application context. The following is an example of how it can be configured in xwork.xml:

```
<interceptors><interceptor name="autowire"
class="com.opensymphony.xwork.spring.interceptor.ActionAutowiringInterceptor"><param
name="autowireStrategy">1</param></interceptor><interceptor-stack
name="autowireDefault"><interceptor-ref name="autowire"/><interceptor-ref
name="defaultStack"/></interceptor-stack></interceptors>
```

Note that the autowireStrategy parameter is optional. If you do not define it, then the SpringObjectFactory will default to autowiring by name. The interceptor looks for Spring's application context in the XWork's application context. To initialize the application context, add the following listener to your web.xml:

```
<listener><listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
```

You do not have to configure the SpringObjectFactory separately unless you plan on instantiating results, interceptors, or validators as Spring beans. As a convenience method to get access to the application context for other uses, it is placed in the ActionContext map under the key ActionAutowiringInterceptor.APPLICATION_CONTEXT for each Action.

Action Chaining

This page last changed on Nov 29, 2004 by [plightbo](#).

The `ActionChainResult` in `WebWork2` provides the ability to compose multiple Actions together to execute in a defined sequence or workflow. By applying the `ActionChainResult` as the result of your Action, like so:

```
<!-- simple chain example to an action in same namespace --><result name="success" type="chain"><param name="actionName">Bar</param></result>
```

```
<!-- example of chaining to an action in a different namespace/package --><result name="success" type="chain"><param name="actionName">viewFoo</param><param name="namespace">/foo</param></result>
```

another Action in the same namespace (or the default "" namespace) can be executed after this Action (see [XW:Configuration](#)). An optional "namespace" parameter may also be added to specify an Action in a different namespace. The original parameters from the request and the `ValueStack` are passed in when this Action is chained to, so the chained to Action will be added on the `ValueStack` above the chained from Action. This allows the chained to Action to access the properties of the preceding Action(s) using the `ValueStack`, and also makes these properties available to the final result of the chain, such as the JSP or Velocity page.

If you need to copy the properties from your previous Actions in the chain to the current Action, you should apply the `ChainingInterceptor` (see [XW:Interceptors](#)) which copies the properties of all objects on the `ValueStack` to the current target.

One common use of Action chaining is to provide lookup lists (like for a dropdown list of states, etc). Since these Actions get put on the `ValueStack`, these properties will be available in the view. This functionality can also be done using the `ActionTag` to execute an Action from the display page. In `WW1.x` Action chaining is often used to chain to a `RedirectAction` to redirect to another page after processing (in `WW2` we have a redirect result).

Basically it's good when you have some reusable code you want to encapsulate... In `WW2` if you use it a lot, you could make it an `Interceptor`, or use it as an Action with chaining. If you need to set up and use some properties from it, it needs to be an Action.

Action Configuration

This page last changed on Aug 30, 2005 by [plightbo](#).

TODO: this whole section overlaps a lot with [xwork.xml](#).

All action configuration is done from within [xwork.xml](#) (see [Configuration](#) for more info). In this section we discuss the various elements that make up the action configuration, such as actions, interceptors, results, and package.

1. [Package Configuration](#)
2. [Namespace Configuration](#)
3. [Result Configuration](#)
4. [Interceptor Configuration](#)

Interceptor Configuration

This page last changed on Aug 30, 2005 by [plightbo](#).

TODO: describe how interceptors are configured. Refer to [Interceptors](#) for descriptions of the ones included with WebWork.

Namespaces

The namespace attribute allows you to segregate action configurations into namespaces, so that you may use the same action alias in more than one namespace with different classes, parameters, etc. This is in contrast to Webwork 1.x, where all action names and aliases were global and could not be re-used in an application. The default namespace, which is "" (an empty string) is used as a "catch-all" namespace, so if an action configuration is not found in a specified namespace, the default namespace will also be searched. This allows you to have global action configurations outside of the "extends" hierarchy, as well as to allow the previous Webwork 1.x behavior by not specifying namespaces. It is also intended that the namespace functionality can be used for security, for instance by having the path before the action name be used as the namespace by the Webwork 2.0 ServletDispatcher, thus allowing the use of J2EE declarative security on paths to be easily implemented and maintained.

Namespace example

```
<package name="default">

  <action name="foo" class="mypackage.simpleAction">
    <result name="success" type="dispatcher">greeting.jsp</result>
  </action>
  <action name="bar" class="mypackage.simpleAction">
    <result name="success" type="dispatcher">bar1.jsp</result>
  </action>

</package>

<package name="mypackage" namespace="/barspace">

  <action name="bar" class="mypackage.simpleAction">
    <result name="success" type="dispatcher">bar2.jsp</result>
  </action>

</package>
```

If a request for /barspace/bar.action is made, then the package named mypackage is searched and the bar action is executed. If success is returned, then bar2.jsp is

displayed.

Note: If a request is made to `/barspace/foo.action`, the action `foo` will be searched for in a namespace of `/barspace`. If the action is not found, the action will then be searched for in the default namespace. Unless specified, the default namespace will be `""`. In our example above, there is no action `foo` in the namespace `/barspace`, therefore the default will be searched and `/foo.action` will be executed.

Overview

Packages are a way to group Actions, Results, Result Types, Interceptors and Stacks into a logical unit that shares a common configuration. Packages are similar to objects in that they can be extended and have individual parts overridden by "sub" packages.

Packages

The package element has one required attribute, "name", which acts as the key for later reference to this package. The "extends" attribute is optional and allows one package to inherit the configuration of one or more previous packages including all interceptor, interceptor-stack, and action configurations. Note that the configuration file is processed sequentially down the document, so the package referenced by an "extends" should be defined above the package which extends it. The "abstract" optional attribute allows you to make a package abstract, which will allow you to extend from it without the action configurations defined in the abstract package actually being available at runtime.

Attribute	Required	Description
name	yes	key to for other packages to reference
extends	no	inherits package behavior of the package it extends
namespace	no	see Namespace Configuration
abstract	no	declares package to be abstract (no action configurations required in package)

Sample usage of packages in xwork.xml

```

<package name="bar" extends="webwork-default"
namespace="/foo/bar"><interceptors><interceptor-stack
name="barDefaultStack"><interceptor-ref name="debugStack"/><interceptor-ref
name="defaultStack"/></interceptor-stack></interceptors><action name="Bar"
class="com.opensymphony.xwork.SimpleAction"><interceptor-ref
name="barDefaultStack"/></action><action name="TestInterceptorParamInheritance"
class="com.opensymphony.xwork.SimpleAction"><interceptor-ref name="test"><param
name="expectedFoo">expectedFoo</param></interceptor-ref></action><action
name="TestInterceptorParamInheritanceOverride"
class="com.opensymphony.xwork.SimpleAction"><interceptor-ref name="test"><param
name="foo">foo123</param><param
name="expectedFoo">foo123</param></interceptor-ref></action></package><package
name="abstractPackage" namespace="/abstract" abstract="true"><action name="test"
class="com.opensymphony.xwork.SimpleAction"/></package><package
name="nonAbstractPackage" extends="abstractPackage"
namespace="/nonAbstract"/><package name="baz" extends="default"
namespace="baz"><action name="commandTest"
class="com.opensymphony.xwork.SimpleAction"><param name="foo">123</param><result
name="error" type="chain"><param
name="actionName">bar</param></result><interceptor-ref
name="static-params"/></action><action name="myCommand"
class="com.opensymphony.xwork.SimpleAction" method="commandMethod"><param
name="bar">456</param><result name="success" type="chain"><param
name="actionName">foo</param></result><interceptor-ref
name="logger"/></action></package><package name="multipleInheritance"
extends="default,abstractPackage,bar" namespace="multipleInheritance"><action
name="testMultipleInheritance" class="com.opensymphony.xwork.SimpleAction"><result
name="success" type="chain"><param
name="actionName">foo</param></result><interceptor-ref
name="barDefaultStack"/></action></package>

```

Overview

Results are string constants that Actions return to indicate the status of an Action execution. A standard set of Results are defined by default: error, input, login, none and success. Developers are, of course, free to create their own Results to indicate more application specific cases. Results are mapped to defined [Result Types](#) using a name-value pair structure.

- [Global results](#)
- [Default results](#)

Result tags

Result tags tell WebWork what to do next after the action has been called. There are a standard set of result codes built-in to WebWork, (in the Action interface) they include:

```
String SUCCESS = "success";
String NONE    = "none";
String ERROR   = "error";
String INPUT   = "input";
String LOGIN   = "login";
```

You can extend these as you see fit. Most of the time you will have either **SUCCESS** or **ERROR**, with **SUCCESS** moving on to the next page in your application;

```
<result name="success" type="dispatcher"><param
name="location">/thank_you.jsp</param></result>
```

...and **ERROR** moving on to an error page, or the preceding page;

```
<result name="error" type="dispatcher"><param
name="location">/error.jsp</param></result>
```


Results are specified in a xwork xml config file(xwork.xml) nested inside <action>. If the `location` param is the only param being specified in the result tag, you can simplify it as follows:

```
<action name="bar" class="myPackage.barAction"><result name="success"
type="dispatcher"><param name="location">foo.jsp</param></result></action>
```

or simplified

```
<action name="bar" class="myPackage.barAction"><result name="success"
type="dispatcher">foo.jsp</result></action>
```

Default results

This page last changed on Dec 13, 2004 by [casey](#).

Webwork has the ability to define a default result type for your actions. Thus, you don't have to specify the result-type for results using the default. If a package extends another package and you don't specify a new default result type for the child package, then the parent package default type will be used when the type attribute is not specified in the result tag.

```
<!-- parts of xwork.xml -->
....

<result-types><result-type name="dispatcher"
class="com.opensymphony.webwork.dispatcher.ServletDispatcherResult"
default="true"/><result-type name="redirect"
class="com.opensymphony.webwork.dispatcher.ServletRedirectResult"/><result-type
name="velocity"
class="com.opensymphony.webwork.dispatcher.VelocityResult"/></result-types>

....

<action name="bar" class="myPackage.barAction"><!-- this result uses dispatcher, so
you can omit the type="dispatcher" if you want --><result
name="success">foo.jsp</result><!-- this result uses velocity result, so the type
needs to be specified --><result name="error"
type="velocity">error.vm</result></action>

....
```

Global results

This page last changed on Dec 13, 2004 by [casey](#).

Global results allows you to define result mappings which will be used as defaults for all action configurations and will be automatically inherited by all action configurations in this package and all packages which extend this package. In other words, if you have the same result specified within multiple actions, then you can define it as a global result.

global results example

```
<package name="default">
....
<global-results><result name="login" type="dispatcher"><param
name="location">login.jsp</param></result></global-results><action name="foo"
class="mypackage.fooAction"><result name="success"
type="dispatcher">bar.jsp</result></action><action name="submitForm"
class="mypackage.submitFormAction"><result name="success"
type="dispatcher">submitSuccess.jsp</result></action>
...
</package>
```

Same thing

```
<package name="default">
....
<action name="foo" class="mypackage.fooAction"><result name="success"
type="dispatcher">bar.jsp</result><result name="login"
type="dispatcher">login.jsp</result></action><action name="submitForm"
class="mypackage.submitFormAction"><result name="success"
type="dispatcher">submitSuccess.jsp</result><result name="login"
type="dispatcher">login.jsp</result></action>
...
</package>
```

ActionMapper

This page last changed on Oct 30, 2005 by [plightbo](#).

Content pulled from external source. Click [here](#) to refresh.

The ActionMapper is responsible for providing a mapping between HTTP requests and action invocation requests and vice-versa. When given an `HttpServletRequest`, the ActionMapper may return null if no action invocation request maps, or it may return an `ActionMapping` that describes an action invocation that WebWork should attempt to try. The ActionMapper is not required to guarantee that the `ActionMapping` returned be a real action or otherwise ensure a valid request. This means that most ActionMappers do not need to consult WebWork's configuration to determine if a request should be mapped.

Just as requests can be mapped from HTTP to an action invocation, the opposite is true as well. However, because HTTP requests (when shown in HTTP responses) must be in String form, a String is returned rather than an actual request object.

By default, the `DefaultActionMapper` is used:

Content pulled from external source. Click [here](#) to refresh.

Default action mapper implementation, using the standard `*.[ext]` (where ext usually "action") pattern. This implementation does not concern itself with parameters. The extension is looked up from the WebWork configuration key **`webwork.action.exaction`**.

You can define your own ActionMapper by configuring the `ActionMapperFactory`:

Content pulled from external source. Click [here](#) to refresh.

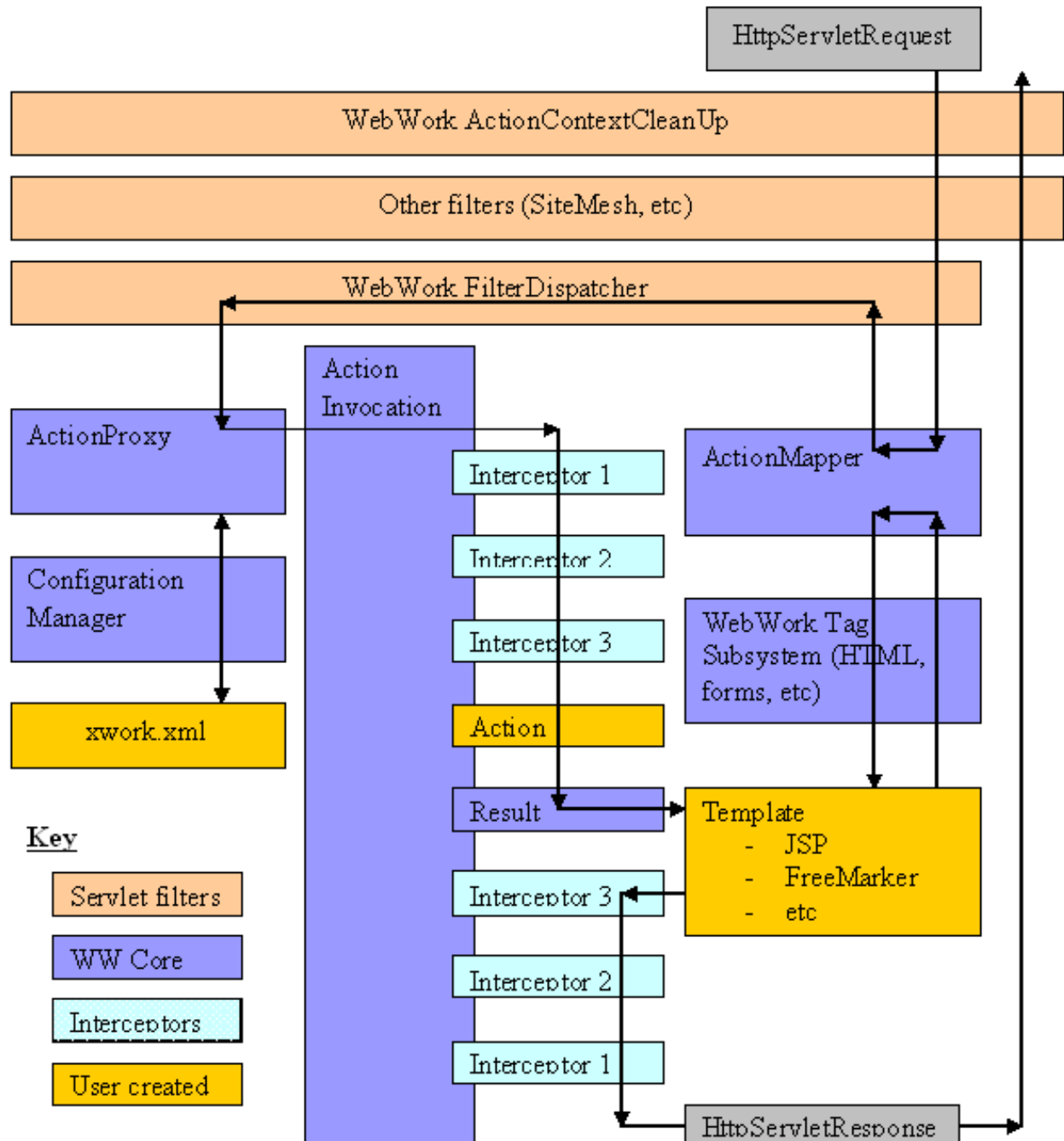
Factory that creates ActionMappers. This factory looks up the class name of the ActionMapper from WebWork's configuration using the key **`webwork.mapper.class`**.

Possible uses of the ActionMapper include defining your own, cleaner namespaces, such as URLs like `/person/1`, which would be similar to a request to `/getPerson.action?personID=1` using the `DefaultActionMapper`.

Architecture

This page last changed on Oct 30, 2005 by [plightbo](#).

The WebWork architecture can best be explained with a diagram:



In the diagram, an initial request goes to the Servlet container (such as Tomcat or Resin), the request goes through the standard filter chain. This includes the (optional)

ActionContextCleanUp filter, which is required if you wish to integrate in with technologies such as [SiteMesh](#). Next, the required **FilterDispatcher** is called, which in turn consults the [ActionMapper](#) to determine if the request should invoke an action.

If the ActionMapper determines that an action should be invoked, the FilterDispatcher then delegates to the **ActionProxy**, which in turn consults the WebWork [Configuration](#) manager, which finally reads your [xwork.xml](#) file. Next, the ActionProxy creates an **ActionInvocation**, which is responsible for the command pattern implementation. This includes invoking any **interceptors** (the *before()* method) before finally invoking the **action** itself.

Once the action returns, the ActionInvocation is responsible for looking up the proper **result** associated with the **action result code** mapped in xwork.xml. The result is then executed, which often (but not always, as is the case for [Action Chaining](#)) involves a template written in [JSP](#) or [FreeMarker](#) to be rendered. While rendering, the templates can utilize the [Tags and UI Components](#) provided by WebWork. Some of those components will work with the ActionMapper to render proper URLs for additional requests.



All objects in this architecture (action, result, interceptor, etc) are created by an ObjectFactory. This ObjectFactory is pluggable and is how frameworks like [Spring](#) and [Pico](#) integrate. You can also provide your own ObjectFactory for any reason that requires knowing when objects in WebWork are created.

Finally, the interceptors are executed again (in reverse order, calling the *after()* method) and finally returning back through the filters configured in web.xml. If the **ActionContextCleanUp** filter is present, the FilterDispatcher will *not* clean up the ThreadLocal **ActionContext**. If the **ActionContextCleanUp** filter is not present, the FilterDispatcher will cleanup all ThreadLocals.

Main Configuration Files

WebWork has two main configuration files you need to be aware of: `web.xml` and `xwork.xml`. Here you will find out all the information you need for both WebWork's required and optional configuration files.

Below are all the files that you may need to be aware of. Some of this configuration files can be reloaded dynamically, making development much easier. See [Reloading configuration](#) for more information.

File	Optional	Location (relative to webapp)	Purpose
web.xml	no	/WEB-INF/	Web deployment descriptor to include all necessary WebWork components
xwork.xml	no	/WEB-INF/classes/	Main configuration, contains result/view types, action mappings, interceptors, etc
webwork.properties	yes	/WEB-INF/classes/	WebWork properties
webwork-default.xml	yes	/WEB-INF/lib/webwork-default.xml	Default configuration that should be included in <code>xwork.xml</code>
velocity.properties	yes	/WEB-INF/classes/	Override the default velocity configuration

validators.xml	yes	/WEB-INF/classes/	Define input validators to be used later
components.xml	yes	/WEB-INF/classes/	Define IOC components
taglib.tld	no	/WEB-INF/lib/webwork-webwork.jar	WebWork tag library descriptor

Static Content

Common static content that is needed by webwork (JavaScript and CSS files, etc.) is served automatically by the FilterDispatcher filter. Any request starting with "/webwork/" denotes that static content is required, and then mapping the value after "/webwork/" to common packages in WebWork and, optionally in your class path.

By default, the following packages are searched:

- com.opensymphony.webwork.static
- template

Additional packages can be specified by providing a comma separated list to the configuration parameter named "packages" (configured in web.xml for the FilterDispatcher filter). When specifying additional static content, you should be careful not to expose sensitive configuration information (i.e. database password).

Reloading configuration

This page last changed on May 14, 2004 by [mgreer](#).

Webwork allows for dynamic reloading of xml configuration file (ie, reloading actions.xml).

This allows you to reconfigure your action mapping during development. There may be a slight performance penalty, so this is not recommended for production use.

In order to enable this feature, add the following to your webwork.properties file:

```
webwork.configuration.xml.reload=true
```

velocity.properties

This page last changed on Aug 30, 2005 by [plightbo](#).

TODO: this should be explained and linked to from the velocity docs.

This file if provided (/WEB-INF/classes) will be loaded by Velocity. It can be used to load custom macros:

```
# Velocity Macro libraries.  
velocimacro.library = webwork.vm, tigris-macros.vm, myapp.vm
```

Check Velocity documentation for other parameters.

web.xml

This page last changed on Aug 30, 2005 by [plightbo](#).

For those using all the latest features of WebWork and have no requirement for backwards compatibility, configuring web.xml is a matter of adding a single filter and, if you're using JSP, a taglib. However, those upgrading from version 2.1.7 or earlier may need to do a bit more work to get everything in order. See [web.xml 2.1.x compatibility](#) for more information.

The filter is configured as:

Content pulled from external source. Click [here](#) to refresh.

```
<filter><filter-name>webwork</filter-name><filter-class>com.opensymphony.webwork.dispatcher.Filter
```

For those using JSP, you may also configuration the tag library as:

Content pulled from external source. Click [here](#) to refresh.

```
<!-- this typically isn't required, as the taglib is include in webwork.jar  
--><taglib><taglib-uri>webwork</taglib-uri><taglib-location>/WEB-INF/webwork.tld</taglib-location>
```

web.xml 2.1.x compatibility

This page last changed on Aug 30, 2005 by [plightbo](#).

Before WebWork 2.2, a `ServletDispatcher` was used to handle action requests. In addition, JSP tags were emulated from within Velocity. WebWork 2.2 made a key changes in this area: The `ServletDispatcher` was deprecated and replaced with a `FilterDispatcher`. This generally works perfectly for users who follow the best practices of WebWork, which is what version 2.2 is pushing. However, due to some small behavioral changes in WebWork 2.2, older applications may require the `ServletDispatcher`.

The biggest change to note is that any application that was including another action, either via a result dispatcher or `jsp/ww:include` tag, no longer works with the `FilterDispatcher`. This is because Servlet containers don't support `RequestDispatchers` out to filter mappings – only servlet mappings are supported. To get around this, you can either change your code to use action chaining in liue of a result dispatcher and the `ww:action` tag in liue of a `jsp/ww:include`.

As a consequence of switching the `FilterDispatcher`, JSP tag emulation from within Velocity does not work. While this feature was never fully robust and supported, we recognize that many users take advantage of the features. As of WebWork 2.2, native Velocity tags are supplied and are the only supported tags within WebWork/Velocity integration.

However, we do provide a deprecated way to avoid changing your code. We recommend that when possible you update your code as suggested. In the meantime, you may add the following Servlets to [web.xml](#):

Content pulled from external source. Click [here](#) to refresh.

```
<servlet><servlet-name>JspSupportServlet</servlet-name><servlet-class>com.opensymphony.webwork.
```

webwork-default.xml

This page last changed on Jun 22, 2005 by [plightbo](#).

A base configuration file named webwork-default.xml is included in the webwork jar file. This file may be included at the top of your xwork.xml file to include the standard configuration settings without having to copy them, like so:

```
<!DOCTYPE xwork PUBLIC "-//OpenSymphony Group//XWork
1.0//EN" "http://www.opensymphony.com/xwork/xwork-1.0.dtd"><xwork><include
file="webwork-default.xml"/><package name="default" extends="webwork-default">
...
</package></xwork>
```

The contents of webwork-default.xml are here:

Content pulled from external source. Click [here](#) to refresh.

```
<xwork><package name="webwork-default"><result-types><result-type name="chain"
class="com.opensymphony.xwork.ActionChainResult"/>
  <result-type name="dispatcher"
class="com.opensymphony.webwork.dispatcher.ServletDispatcherResult"
  default="true"/>
  <result-type name="freemarker"
class="com.opensymphony.webwork.views.freemarker.FreemarkerResult"/><result-type
name="httpheader"
class="com.opensymphony.webwork.dispatcher.HttpHeaderResult"/><result-type
name="jasper"
class="com.opensymphony.webwork.views.jasperreports.JasperReportsResult"/><result-type
name="redirect" class="com.opensymphony.webwork.dispatcher.ServletRedirectResult"/>
  <result-type name="redirect-action"
    class="com.opensymphony.webwork.dispatcher.ServletActionRedirectResult"/>
  <result-type name="stream"
class="com.opensymphony.webwork.dispatcher.StreamResult"/><result-type
name="velocity"
class="com.opensymphony.webwork.dispatcher.VelocityResult"/><result-type name="xslt"
class="com.opensymphony.webwork.views.xslt.XSLTResult"/></result-types><interceptors><interceptor
name="alias" class="com.opensymphony.xwork.interceptor.AliasInterceptor"/>
  <interceptor name="autowiring"
    class="com.opensymphony.xwork.spring.interceptor.ActionAutowiringInterceptor"/>
  <interceptor name="chain"
class="com.opensymphony.xwork.interceptor.ChainingInterceptor"/><interceptor
name="component"
class="com.opensymphony.xwork.interceptor.component.ComponentInterceptor"/>
  <interceptor name="conversionError"
    class="com.opensymphony.webwork.interceptor.WebWorkConversionErrorInterceptor"/>
  <interceptor name="external-ref"
class="com.opensymphony.xwork.interceptor.ExternalReferencesInterceptor"/><interceptor
name="execAndWait"
class="com.opensymphony.webwork.interceptor.ExecuteAndWaitInterceptor"/><interceptor
name="exception"
class="com.opensymphony.xwork.interceptor.ExceptionMappingInterceptor"/><interceptor
name="fileUpload"
class="com.opensymphony.webwork.interceptor.FileUploadInterceptor"/><interceptor
name="i18n" class="com.opensymphony.xwork.interceptor.I18nInterceptor"/><interceptor
```

```

name="logger"
class="com.opensymphony.xwork.interceptor.LoggingInterceptor"/><interceptor
name="model-driven"
class="com.opensymphony.xwork.interceptor.ModelDrivenInterceptor"/><interceptor
name="params"
class="com.opensymphony.xwork.interceptor.ParametersInterceptor"/><interceptor
name="prepare"
class="com.opensymphony.xwork.interceptor.PrepareInterceptor"/><interceptor
name="static-params"
class="com.opensymphony.xwork.interceptor.StaticParametersInterceptor"/><interceptor
name="servlet-config"
class="com.opensymphony.webwork.interceptor.ServletConfigInterceptor"/>
    <interceptor name="sessionAutowiring"
        class="com.opensymphony.webwork.spring.interceptor.SessionContextAutowiringInterceptor"/>
    <interceptor name="timer"
class="com.opensymphony.xwork.interceptor.TimerInterceptor"/><interceptor
name="token" class="com.opensymphony.webwork.interceptor.TokenInterceptor"/>
    <interceptor name="token-session"
        class="com.opensymphony.webwork.interceptor.TokenSessionStoreInterceptor"/>
    <interceptor name="validation"
class="com.opensymphony.xwork.validator.ValidationInterceptor"/><interceptor
name="workflow"
class="com.opensymphony.xwork.interceptor.DefaultWorkflowInterceptor"/><!-- Basic
stack --><interceptor-stack name="basicStack"><interceptor-ref
name="exception"/><interceptor-ref name="servlet-config"/><interceptor-ref
name="prepare"/><interceptor-ref name="static-params"/><interceptor-ref
name="params"/><interceptor-ref name="conversionError"/></interceptor-stack><!--
Sample validation and workflow stack --><interceptor-stack
name="validationWorkflowStack"><interceptor-ref name="basicStack"/><interceptor-ref
name="validation"/><interceptor-ref name="workflow"/></interceptor-stack><!-- Sample
file upload stack --><interceptor-stack name="fileUploadStack"><interceptor-ref
name="fileUpload"/><interceptor-ref name="basicStack"/></interceptor-stack>

    <!-- Sample WebWork Inversion of Control stack
        Note: WebWork's IoC is deprecated - please
        look at alternatives such as Sprint -->
    <interceptor-stack name="componentStack"><interceptor-ref
name="component"/><interceptor-ref name="basicStack"/></interceptor-stack><!--
Sample model-driven stack --><interceptor-stack
name="modelDrivenStack"><interceptor-ref name="model-driven"/><interceptor-ref
name="basicStack"/></interceptor-stack><!-- Sample action chaining stack
--><interceptor-stack name="chainStack"><interceptor-ref
name="chain"/><interceptor-ref name="basicStack"/></interceptor-stack><!-- Sample
i18n stack --><interceptor-stack name="chainStack"><interceptor-ref
name="i18n"/><interceptor-ref name="basicStack"/></interceptor-stack>

    <!-- Sample execute and wait stack.
        Note: execAndWait should always be the *last* interceptor. -->
    <interceptor-stack name="executeAndWaitStack"><interceptor-ref
name="basicStack"/><interceptor-ref name="execAndWait"/></interceptor-stack>

    <!-- A complete stack with all the common interceptors in place.
        Generally, this stack should be the one you use, though it
        may process additional stuff you don't need, which could
        lead to some performance problems. Also, the ordering can be
        switched around (ex: if you wish to have your components
        before prepare() is called, you'd need to move the component
        interceptor up -->
    <interceptor-stack name="defaultStack"><interceptor-ref
name="exception"/><interceptor-ref name="alias"/><interceptor-ref
name="prepare"/><interceptor-ref name="servlet-config"/><interceptor-ref
name="i18n"/><interceptor-ref name="chain"/><interceptor-ref

```

```
<interceptor-ref name="fileUpload"/><interceptor-ref
name="static-params"/><interceptor-ref name="params"/><interceptor-ref
name="conversionError"/><interceptor-ref name="validation"/><interceptor-ref
name="workflow"/></interceptor-stack>

    <!-- The completeStack is here for backwards compatibility for
         applications that still refer to the defaultStack by the
         old name -->
    <interceptor-stack name="completeStack"><interceptor-ref
name="defaultStack"/></interceptor-stack></interceptors><default-interceptor-ref
name="defaultStack"/></package></xwork>
```

This file defines all of the default bundled results and interceptors and many interceptor stacks which you can use either as-is or as a basis for your own application-specific interceptor stacks. **Notice the name of the package is "webwork-default".**

webwork.properties

This page last changed on Aug 30, 2005 by [plightbo](#).

WebWork uses a number of properties that can be changed to fit your needs. To change them, specify your values in `webwork.properties` in the classpath (typically `/WEB-INF/classes`). The list of properties can be found in `default.properties` (inside `webwork.jar`):

Content pulled from external source. Click [here](#) to refresh.

```
###
### Webwork default properties (can be overridden by a weework.properties file in
the root of the classpath)
###

### This can be used to set your default locale and encoding scheme
#webwork.locale=en_US
webwork.il8n.encoding=ISO-8859-1

# if specified, the default object factory can be overridden here
# Note: short-hand notation is supported in some cases, such as "spring"
#     alternatively, you can provide a class name here
#webwork.objectFactory = spring

# specifies the autoWiring logic when using the SpringObjectFactory.
# valid values are: name, type, auto, and constructor (name is the default)
webwork.objectFactory.spring.autoWire = name

### Parser to handle HTTP POST requests, encoded using the MIME-type
multipart/form-data
#webwork.multipart.parser=cos
#webwork.multipart.parser=pell
webwork.multipart.parser=jakarta
# uses javax.servlet.context.tempdir by default
webwork.multipart.saveDir=
webwork.multipart.maxSize=2097152

### Load custom property files (does not override weework.properties!)
#webwork.custom.properties=application,com/webwork/extension/custom

# extension for actions
webwork.mapper.class=com.opensymphony.webwork.dispatcher.mapper.DefaultActionMapper
webwork.action.extension=action

# use beta alternative syntax that requires %{} in most places
# to evaluate expressions for String attributes for tags
webwork.tag.altSyntax=true

# when set to true, WebWork will act much more friendly for developers. This
# includes:
# - weework.il8n.reload = true
# - weework.configuration.xml.reload = true
# - raising various debug or ignorable problems to errors
#   For example: normally a request to foo.action?someUnknownField=true should
#                 be ignored (given that any value can come from the web and it
#                 should not be trusted). However, during development, it may be
```



```

#             useful to know when these errors are happening and be told of
#             them right away.
webwork.devMode = false

# when set to true, resource bundles will be reloaded on _every_ request.
# this is good during development, but should never be used in production
webwork.il8n.reload=false

### Standard UI theme
# Change this to reflect which path should be used for JSP control tag templates by
default
webwork.ui.theme=xhtml
webwork.ui.templateDir=template
#sets the default template type. Either ftl, vm, or jsp
webwork.ui.templateSuffix=ftl

### Configuration reloading
# This will cause the configuration to reload xwork.xml when it is changed
webwork.configuration.xml.reload=false

### Location of velocity.properties file. defaults to velocity.properties
#webwork.velocity.configfile = velocity.properties

### Comma separated list of VelocityContext classnames to chain to the
WebWorkVelocityContext
#webwork.velocity.contexts =

# used to build URLs, such as the UrlTag
webwork.url.http.port = 80
webwork.url.https.port = 443

### Load custom default resource bundles
#webwork.custom.il8n.resources=testmessages,testmessages2

# workaround for some app servers that don't handle
HttpServletRequest.getParameterMap()
# often used for WebLogic, Orion, and OC4J
webwork.dispatcher.parametersWorkaround = false

```

TODO: There is a lot of overlapping info at [Action Configuration](#).

Example

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE xwork
    PUBLIC
    "-//OpenSymphony Group//XWork
1.0//EN" "http://www.opensymphony.com/xwork/xwork-1.0.dtd">
<xwork><include file="webwork-default.xml"/><package name="default"
extends="webwork-default"><interceptors><interceptor-stack
name="defaultComponentStack"><interceptor-ref name="component"/><interceptor-ref
name="defaultStack"/></interceptor-stack></interceptors><default-interceptor-ref
name="defaultStack"/><action name="SimpleCounter"
class="com.opensymphony.webwork.example.counter.SimpleCounter"><result
name="success" type="dispatcher"/>success.jsp</result><interceptor-ref
name="defaultComponentStack"/></action>

    <!--
    - Velocity implementation of the SimpleCounter. Also demonstrate a more
    verbose version of result element
    -->
    <action name="VelocityCounter"
class="com.opensymphony.webwork.example.counter.SimpleCounter"><result
name="success" type="velocity"><param
name="location"/>success.vm</param></result><interceptor-ref
name="defaultComponentStack"/></action>

    <!--
    - Different method can be used (processForm).
    -->
    <action name="formTest" class="com.opensymphony.webwork.example.FormAction"
method="processForm" ><result name="success"
type="dispatcher"/>formSuccess.jsp</result><result name="invalid.token"
type="dispatcher"/>form.jsp</result><interceptor-ref
name="defaultStack"/><interceptor-ref name="token"/></action></package></xwork>
```

Actions

```
<action name="formTest" class="com.opensymphony.webwork.example.FormAction"
method="processForm">
```

Actions are the basic "unit-of-work" in WebWork, they define, well, actions. An action will usually be a request, (and usually a button click, or form submit). The main action

element (tag is too synonymous with JSP) has two parts, the friendly name (referenced in the URL, i.e. saveForm.action) and the corresponding "handler" class.

The optional "**method**" parameter tells WebWork which method to call based upon this action. If you leave the method parameter blank, WebWork will call the method **execute()** by default. If there is no execute() method and no method specified in the xml file, WebWork will throw an exception.

Also, you can tell WebWork to invoke "**doSomething**" method in your action by using the pattern "**actionName!something**" in your form. For example, "**formTest!save.action**" will invoke the method "**save**" in FormAction class. The method must be public and take no arguments:

```
publicString save() throws Exception
{
    ...
    return SUCCESS;
}
```

All the configuration for "**actionName**" will be used for "actionName!something" (interceptors, result types, etc...)

Results

```
<result name="missing-data" type="dispatcher"><param
name="location">/form.jsp</param><param name="parameterA">A</param><param
name="parameterB">B</param></result>
```

Result tags tell WebWork what to do next after the action has been called. The "name" attribute maps to the result code returned from the action execute() method. The "type" attribute indicates what result type class to use (see [Result Types](#)). The "param" elements allow you to pass parameters to the view:

```
<result-types>
....
<result-type name="header"
    class="com.opensymphony.webwork.dispatcher.HttpHeaderResult"/>
```

```
</result-types><result name="no-content" type="header"><param
name="status">204</param><param name="headers.customHeaderA">A</param><param
name="headers.customHeaderB">B</param></result>
```

There are a standard set of result codes built-in to WebWork, (in the Action interface) they include:

```
Action.SUCCESS = "success";
Action.NONE    = "none";
Action.ERROR   = "error";
Action.INPUT   = "input";
Action.LOGIN   = "login";
```

You can extend these result codes as you see fit (i.e "missing-data"). Most of the time you will have either SUCCESS or ERROR, with SUCCESS moving on to the next page in your application.

If you only need to specify the "location" parameter, you can use the short form:

```
<result name="missing-data" type="dispatcher">/form.jsp</result>
```

See [webwork-default.xml](#) or [Result Types](#) for standard result types.

Interceptors

Interceptors allow you to define code to be executed before and/or after the execution of an action. Interceptors can be a powerful tool when writing web applications. Some of the most common implementations of an Interceptor might be:

- Security Checking (ensuring the user is logged in)
- Trace Logging (logging every action)
- Bottleneck Checking (start a timer before and after every action, to check bottlenecks in your application)

You can also chain Interceptors together to create an interceptor **stack**. If you wanted to do a login check, security check, and logging all before an Action call, this could easily be done with an interceptor package.

Interceptors must first be defined (to give name them) and can be chained together as a stack:

```
<interceptors><interceptor name="security"
class="com.mycompany.security.SecurityInterceptor"/><interceptor-stack
name="defaultComponentStack"><interceptor-ref name="component"/><interceptor-ref
name="defaultStack"/></interceptor-stack></interceptors>
```

To use them in your actions:

```
<action name="VelocityCounter"
class="com.opensymphony.webwork.example.counter.SimpleCounter"><result
name="success">...</result><interceptor-ref name="defaultComponentStack"/></action>
```

NOTE: Reference name can be either the name of the interceptor or the name of a stack

For more details, see [Interceptors](#) reference.

Views

WebWork supports JSP and Velocity for your application presentation layer. For this example we will use a JSP file. Webwork comes packaged with a tag library (taglibs). You can use these taglibs as components in your JSP file. Here is an section of our form.jsp page:

```
<%@ taglib prefix="ww" uri="webwork" %><html><head><title>Webwork Form
Example</title></head><body><ww:form name="myForm" action="'formTest'" namespace="/"
method="POST"><table><ww:textfield label="First Name" name="'formBean.firstName'"
value="formBean.firstName"/><ww:textfield label="Last Name"
name="'formBean.lastName'" value="formBean.lastName"/><ww:submit value="Save
Form"/></table></ww:form></body>
```

The process of events will go as follows:

1. WebWork will take notice since the URI ends in **.action** (defined in **web.xml** files)
2. WebWork will look up the action **formTest** in its action hierarchy and call any Interceptors that might have been defined.
3. WebWork will translate **formTest** and decide to call the method **processForm** in the class **com.opensymphony.webwork.example.FormAction** as defined in **xwork.xml** file.
4. The method will process successfully and give WebWork the **SUCCESS** return parameter.

5. WebWork will translate the **SUCCESS** return parameter into the location **formSuccess.jsp** (as defined in **xwork.xml**) and redirect accordingly.

Include

To make it easy to manage large scale development (lots of actions + configuration), WebWork allows you to include other configuration files from xwork.xml :

```
<xwork><include file="webwork-default.xml"/><include file="user.xml"/><include  
file="shoppingcart.xml"/><include file="product.xml"/>  
....  
</xwork>
```


The included files must be the same format as xwork.xml (with the doctype and everything) and be placed on classpath (usually in /WEB-INF/classes or jar files in /WEB-INF/lib).

Most of the content here provided by Matt Dowell <matt.dowell@notiva.com>

Continuations

This page last changed on Oct 17, 2005 by [plightbo](#).

Continuations are a feature in WebWork, borrowed from the [RIFE project](#), that allow for extremely simple state management and wizard-like functionality.

	Continuations are currently experimental, and as such, we cannot recommend they be used for heavy-use production deployments at this time. We will continue to work with the community to stabilize and enhance this feature until we are confident it can be used in the most extreme site traffic and use-cases.
---	--

Setting it Up

Setting up continuation support requires identifying the base package that your classes are in. This is done in [webwork.properties](#) using the key **webwork.continuations.package**. Typically, this can be the root package that your classes are found in, such as **com.acme**.

Once you've done this, WebWork will analyze your classes and automatically apply continuation support for any class that uses the continuation features - specifically a class that extends `ActionSupport` that has an **execute()** method that calls a **pause()** method.

URL Concerns

Because continuations require the state of your flow be managed by WebWork, it is up to you to make sure your application inform WebWork what the flow's ID is. This is done via a **continue** parameter that provides a unique ID for every request in the flow. Assuming you are generating your URLs using the [URL](#) tag or the [Form](#) tag, this

is handled for you automatically. If you are *not* using these tags, continuations will not work.

Interceptor Concerns

Because continuations radically change the way your actions are invoked, it is important to understand how this affects interceptors. The most important thing to know is that continuations kick in only when the `execute()` method is called. This means that on every request (regardless of whether it is a new request or a continuation), the interceptors will be called. This is what makes it possible to apply new request parameters to your action even though the rest of the call stack appears to look the same.

This is generally exactly what you would want, except some interceptors, namely the [Execute and Wait Interceptor](#) and possibly the [Token Session Interceptor](#), have very different expectations about the workflow/lifecycle of the action invocation. In these cases, continuations should not be used.

Example

Getting started with continuations is extremely simple. The biggest thing to get used to is the very different conversational style with application workflow. Typically, you might have used session variables or hidden form fields to pass the state around. Using continuations, you use the Java language to handle that state. See the following body of a `Guess` class extending `ActionSupport`:

Content pulled from external source. Click [here](#) to refresh.

```
public class Guess extends ActionSupport {
    int guess;

    public String execute() throws Exception {
        int answer = new Random().nextInt(100) + 1;
        int tries = 5;

        while (answer != guess && tries > 0) {
            pause(SUCCESS);
        }
    }
}
```



```

        if (guess > answer) {
            addFieldError("guess", "Too high!");
        } elseif (guess < answer) {
            addFieldError("guess", "Too low!");
        }

        tries--;
    }

    if (answer == guess) {
        addActionMessage("You got it!");
    } else {
        addActionMessage("You ran out of tries, the answer was " + answer);
    }

    return SUCCESS;
}

public void setGuess(int guess) {
    this.guess = guess;
}
}

```

Note how the class keeps the state (tries, in this example) as a local variable in the `execute()` method. WebWork's continuations will automatically pick up the invocation after the `pause()` method call and will restore all local variables, as if the logical loop is continuing "magically" (read on for more info on how it works).

The view is nothing special, except for that fact that it adheres to the URL concerns and uses the [Form](#) tag to render the URL. This makes sure that the **continue** parameter is included in all requests.

Content pulled from external source. Click [here](#) to refresh.

```

<html><head><title></title></head><body><#list actionMessages as msg>
    ${msg}
</#list><@ww.form action="guess" method="post"><@ww.textfield label="Guess"
name="guess" /><@ww.submit value="Guess" /></@ww.form></body></html>

```

Advanced: How it Works

Continuations are not magic, though sometimes they might seem like they are. In fact, they work by using some very intelligent byte-code manipulation. This means that **in order to use continuations, your deployment environment allow for custom class loaders to handle loading your actions**. Typically this is not a problem, but it should be called out.

Once the class is requested to be loaded, WebWork will hand off the request to the RIFE/Continuations module, which will then check a few conditions:

1. Does the class extend `ActionSupport`?
2. Does the class have an `execute()` method?
3. In the `execute()` method, are there any calls to `pause()`?

If the answer is *yes* to all three conditions, the class is then instrumented and the `execute()` method is rewritten with try/catch code, goto statements, and intelligent "state restoration" code. All this happens transparently and does not affect the ability to debug the class or otherwise code it.

See the `pause()` method JavaDocs in the `ActionSupport` class for more info:

Content pulled from external source. Click [here](#) to refresh.

Stops the action invocation immediately (by throwing a `PauseException`) and causes the action invocation to return the specified result, such as `#SUCCESS`, `#INPUT`, etc.

The next time this action is invoked (and using the same continuation ID), the method will resume immediately after where this method was called, with the entire call stack in the `execute` method restored.

Note: this method can **only** be called within the `#execute()` method.

TODO: General document describing how FM works with WebWork. It should cover:

- What result type to use (freemarker)
- How to configure freemarker
- What tags are available (refer to the general tag documentation, don't re-document them here)
- Items available in the freemarker context
- General tips and tricks w/ FM (such as utilizing FreeMarkerManager)
- Common pitfalls

Interceptors

This page last changed on Oct 18, 2005 by [digi9ten](#).

See [Interceptor Configuration](#) for basic information about how interceptors are configured.

Overview

Interceptors are objects that dynamically intercept Action invocations. They provide the developer with the opportunity to define code that can be executed before and/or after the execution of an action. They also have the ability to prevent an action from executing. Interceptors provide developers a way to encapsulate common functionality in a re-usable form that can be applied to one or more Actions. See [XW:Interceptors](#) for further details. Below describes built in Webwork interceptors.

Webwork & XWork Interceptors

Interceptor classes are also defined using a key-value pair specified in the xwork configuration file. The names specified below come specified in [webwork-default.xml](#). If you extend the webwork-default package, then you can use the names below. Otherwise they must be defined in your package with a name-class pair specified in the <interceptors> tag.

Interceptor	Name	Description
Alias Interceptor	alias	Converts similar parameters that may be named differently between requests.
Chaining Interceptor	chain	Makes the previous action's properties available to the current action. Commonly used together with <result type="chain"> (in the previous action).

Component Interceptor	component	Enables and makes the components available to the Actions. Refer to components.xml
Conversion Error Interceptor	conversionError	adds conversion errors from the ActionContext to the Action's field errors
Execute and Wait Interceptor	execAndWait	an interceptor that executes the action in the background and then sends the user off to an intermediate waiting page.
Exception Interceptor	exception	Maps exceptions to a result.
File Upload Interceptor	fileUpload	an interceptor that adds easy access to file upload support. See the javadoc for more info
I18n Interceptor	i18n	remembers the locale selected for a user's session
Logger Interceptor	logger	Outputs the name of the action
Model Driven Interceptor	model-driven	If the action implements ModelDriven, pushes the getModel() result onto the valuestack.
Parameters Interceptor	params	Sets the request parameters onto the action.
Prepare Interceptor	prepare	If the action implements Preparable, calls its prepare() method.
Scope Interceptor	scope	simple mechanism for

		storing action state in the session or application scope
Servlet Config Interceptor	servlet-config	Give access to <code>HttpServletRequest</code> and <code>HttpServletResponse</code> (think twice before using this since this ties you to the Servlet api)
Static Parameters Interceptor	static-params	Sets the <code>xwork.xml</code> defined parameters onto the action. These are the <code><param></code> tags that are direct children of the <code><action></code> tag.
Timer Interceptor	timer	Outputs how long the action (including nested interceptors and view) takes to execute
Token Interceptor	token	Checks for valid token presence in action, prevents duplicate form submission
Token Session Interceptor	token-session	Same as above, but storing the submitted data in session when handed an invalid token
Validation Interceptor	validation	Performs validation using the validators defined in <code>{Action}-validation.xml</code>
Workflow Interceptor	workflow	Calls the <code>validate</code> method in your action class. If action errors created then it returns the INPUT view.

Order of Interceptor Execution

Interceptors provide an excellent means to wrap before/after processing. The concept reduces code duplication (think AOP).

```
<interceptor-stack name="xaStack">
  <interceptor-ref name="thisWillRunFirstInterceptor"/>
  <interceptor-ref name="thisWillRunNextInterceptor"/>
  <interceptor-ref name="followedByThisInterceptor"/>
  <interceptor-ref name="thisWillRunLastInterceptor"/>
</interceptor-stack>
```

Note that some interceptors will interrupt the stack/chain/flow... so the order is very important.

Interceptors implementing `com.opensymphony.xwork.interceptor.PreResultListener` will run after the Action executes its action method but before the Result executes

```
thisWillRunFirstInterceptor
  thisWillRunNextInterceptor
    followedByThisInterceptor
      thisWillRunLastInterceptor
        MyAction1
        MyAction2 (chain)
        MyPreResultListener
        MyResult (result)
      thisWillRunLastInterceptor
    followedByThisInterceptor
  thisWillRunNextInterceptor
thisWillRunFirstInterceptor
```

Alias Interceptor

This page last changed on Oct 18, 2005 by [digi9ten](#).

Content pulled from external source. Click [here](#) to refresh.

The aim of this Interceptor is to alias a named parameter to a different named parameter. By acting as the glue between actions sharing similiar parameters (but with different names), it can help greatly with action chaining.

Action's alias expressions should be in the form of `#{"name1" : "alias1", "name2" : "alias2" }`. This means that assuming an action (or something else in the stack) has a value for the expression named *name1* and the action this interceptor is applied to has a setter named *alias1*, *alias1* will be set with the value from *name1*.

Parameters

Content pulled from external source. Click [here](#) to refresh.

- `aliasesKey` (optional) - the name of the action parameter to look for the alias map (by default this is *aliases*).

Extending the Interceptor

Content pulled from external source. Click [here](#) to refresh.

This interceptor does not have any known extension points.

Examples

Content pulled from external source. Click [here](#) to refresh.

```
<action name="someAction" class="com.examples.SomeAction"><!-- The value for the foo
parameter will be applied as if it were named bar --><param name="aliases">#{ 'foo'
: 'bar' }</param><!-- note: the alias interceptor is included with the defaultStack
in webwork-default.xml --><interceptor-ref name="alias"/><interceptor-ref
name="basicStack"/><result name="success">good_result.ftl</result></action>
```


Chaining Interceptor

This page last changed on Oct 24, 2005 by [plightbo](#).

Content pulled from external source. Click [here](#) to refresh.

An interceptor that copies all the properties of every object in the value stack to the currently executing object, except for any object that implements Unchainable. A collection of optional *includes* and *excludes* may be provided to control how and which parameters are copied. Only includes or excludes may be specified. Specifying both results in undefined behavior. See the javadocs for {@link OgnlUtil#copy(Object, Object, java.util.Map, java.util.Collection, java.util.Collection)} for more information.

It is important to remember that this interceptor does nothing if there are no objects already on the stack. This means two things: One, you can safely apply it to all your actions without any worry of adverse affects. Two, it is up to you to ensure an object exists in the stack prior to invoking this action. The most typical way this is done is through the use of the **chain** result type, which combines with this interceptor to make up the action chaining feature.

Parameters

Content pulled from external source. Click [here](#) to refresh.

- excludes (optional) - the list of parameter names to exclude from copying (all others will be included).
- includes (optional) - the list of parameter names to include when copying (all others will be excluded).

Extending the Interceptor

Content pulled from external source. Click [here](#) to refresh.

There are no known extension points to this interceptor.

Examples

Content pulled from external source. Click [here](#) to refresh.

```
<action name="someAction" class="com.examples.SomeAction"><interceptor-ref
name="basicStack"/><result name="success"
type="chain">otherAction</result></action><action name="otherAction"
class="com.examples.OtherAction"><interceptor-ref name="chain"/><interceptor-ref
name="basicStack"/><result name="success">good_result.ftl</result></action>
```

Component Interceptor

This page last changed on Oct 18, 2005 by [digi9ten](#).

Content pulled from external source. Click [here](#) to refresh.

A simple interceptor that applies the WebWork IOC container ComponentManager against the executing action. Note, WebWork IOC is deprecated and it is highly recommended that you look at alternative solutions, such as Spring.

Parameters

Content pulled from external source. Click [here](#) to refresh.

- None

Extending the Interceptor

Content pulled from external source. Click [here](#) to refresh.

There are no known extension points to this interceptor.

Examples

Content pulled from external source. Click [here](#) to refresh.

```
<action name="someAction" class="com.examples.SomeAction"><interceptor-ref  
name="componentStack"/><interceptor-ref name="basicStack"/><result  
name="success">good_result.ftl</result></action>
```

Conversion Error Interceptor

This page last changed on Oct 25, 2005 by [plightbo](#).

To fully document this interceptor, it is best to look at the JavaDocs for the subclass of the interceptor, `ConversionErrorInterceptor`:

Content pulled from external source. Click [here](#) to refresh.

This interceptor adds any error found in the `ActionContext`'s `conversionErrors` map as a field error (provided that the action implements `ValidationAware`). In addition, any field that contains a validation error has its original value saved such that any subsequent requests for that value return the original value rather than the value in the action. This is important because if the value "abc" is submitted and can't be converted to an int, we want to display the original string ("abc") again rather than the int value (likely 0, which would make very little sense to the user).

... as well as the JavaDocs for the interceptor itself, `WebWorkConversionErrorInterceptor`:

Content pulled from external source. Click [here](#) to refresh.

This interceptor extends `ConversionErrorInterceptor` but only adds conversion errors from the `ActionContext` to the field errors of the action if the field value is not null, "", or {""} (a size 1 `String` array with only an empty `String`). See `ConversionErrorInterceptor` for more information, as well as the `Type Conversion` documentation.

Parameters

Content pulled from external source. Click [here](#) to refresh.

- None

Extending the Interceptor

Content pulled from external source. Click [here](#) to refresh.

There are no known extension points for this interceptor.

Examples

Content pulled from external source. Click [here](#) to refresh.

```
<action name="someAction" class="com.examples.SomeAction"><interceptor-ref  
name="params"/><interceptor-ref name="conversionError"/><result  
name="success">good_result.ftl</result></action>
```

Exception Interceptor

This page last changed on Oct 18, 2005 by [digi9ten](#).

Content pulled from external source. Click [here](#) to refresh.

This interceptor forms the core functionality of the exception handling feature. Exception handling allows you to map an exception to a result code, just as if the action returned a result code instead of throwing an unexpected exception. When an exception is encountered, it is wrapped with an `ExceptionHandler` and pushed on the stack, providing easy access to the exception from within your result.

Note: While you can configure exception mapping in your configuration file at any point, the configuration will not have any effect if this interceptor is not in the interceptor stack for your actions. It is recommended that you make this interceptor the first interceptor on the stack, ensuring that it has full access to catch any exception, even those caused by other interceptors.

Parameters

Content pulled from external source. Click [here](#) to refresh.

- None

Extending the Interceptor

Content pulled from external source. Click [here](#) to refresh.

There are no known extension points.

Examples

Content pulled from external source. Click [here](#) to refresh.

```
<xwork><include file="webwork-default.xml"/><package name="default"
extends="webwork-default"><global-results><result name="success"
```

```
error.ftl</result></global-results><global-exception-mappings><exception-mapping
exception="java.lang.Exception" result="error"/></global-exception-mappings><action
name="test"><interceptor-ref name="exception"/><interceptor-ref
name="basicStack"/><exception-mapping exception="com.acme.CustomException"
result="custom_error"/><result name="custom_error">custom_error.ftl</result><result
name="success" type="freemarker">test.ftl</result></action></package></xwork>
```

Execute and Wait Interceptor

This page last changed on Oct 18, 2005 by [plightbo](#).

Content pulled from external source. Click [here](#) to refresh.

The ExecuteAndWaitInterceptor is great for running long-lived actions in the background while showing the user a nice progress meter. This also prevents the HTTP request from timing out when the action takes more than 5 or 10 minutes.

Using this interceptor is pretty straight forward. Assuming that you are including webwork-default.xml, this interceptor is already configured but is not part of any of the default stacks. Because of the nature of this interceptor, it must be the **last** interceptor in the stack.

This interceptor works on a per-session basis. That means that the same action name (myLongRunningAction, in the above example) cannot be run more than once at a time in a given session. On the initial request or any subsequent requests (before the action has completed), the **wait** result will be returned. **The wait result is responsible for issuing a subsequent request back to the action, giving the effect of a self-updating progress meter.**

If no "wait" result is found, WebWork will automatically generate a wait result on the fly. This result is written in FreeMarker and cannot run unless FreeMarker is installed. If you don't wish to deploy with FreeMarker, you must provide your own wait result. This is generally a good thing to do anyway, as the default wait page is very plain.

Whenever the wait result is returned, the **action that is currently running in the background will be placed on top of the stack.** This allows you to display progress data, such as a count, in the wait page. By making the wait page automatically reload the request to the action (which will be short-circuited by the interceptor), you can give the appearance of an automatic progress meter.

Important: Because the action will be running in a separate thread, you can't use ActionContext because it is a ThreadLocal. This means if you need to access, for example, session data, you need to implement SessionAware rather than calling ActionContext.getSession().

The thread kicked off by this interceptor will be named in the form **actionNameBackgroundProcess**. For example, the *search* action would run as a thread named *searchBackgroundProcess*.

Parameters

Content pulled from external source. Click [here](#) to refresh.

- threadPriority (optional) - the priority to assign the thread

Extending the Interceptor

Content pulled from external source. Click [here](#) to refresh.

If you wish to make special preparations before and/or after the invocation of the background thread, you can extend the BackgroundProcess class and implement the beforeInvocation() and afterInvocation() methods. This may be useful for obtaining and releasing resources that the background process will need to execute successfully. To use your background process extension, extend ExecuteAndWaitInterceptor and implement the getNewBackgroundProcess() method.

Examples

Content pulled from external source. Click [here](#) to refresh.

```
<action name="someAction" class="com.examples.SomeAction"><interceptor-ref
name="completeStack"/><interceptor-ref name="execAndWait"/><result
name="success">longRunningAction-wait.jsp</result><result
name="success">longRunningAction-success.jsp</result></action><%@ taglib prefix="ww"
uri="/webwork" %><html><head><title>Please wait</title><meta http-equiv="refresh"
content="5;url=<ww:url includeParams="all" />" /></head><body>
    Please wait while we process your request.
    Click <a href="<ww:url includeParams="all" />"></a> if this page does not reload
automatically.
</body></html>
```

HibernateAndSpringEnabledExecuteAndWaitInterceptor

This page last changed on Oct 17, 2005 by [plightbo](#).

Find example code below for an extension of the *ExecuteAndWaitInterceptor*.

The goal of this code is to allow a background process to execute while having access to the same open Hibernate session object.

The `SessionFactory` dependency is injected into the *OpenSessionExecuteAndWaitInterceptor* by Spring. You may use other methods of dependency injection if you are more comfortable with them. By overriding the *getNewBackgroundProcess()* method, this interceptor uses our custom *OpenSessionBackgroundProcess* instead of the WebWork default.

Overriding the *beforeInvocation()* and *afterInvocation()* methods in the *OpenSessionBackgroundProcess* ensure that the session will stay open throughout the life of the background process, and any Spring transaction management will also be used.

As this code is heavily dependent on Spring and Hibernate, you shouldn't expect to see it packaged with a WebWork distribution. It does, however, serve as a useful example of extending the [Execute and Wait Interceptor](#)

OpenSessionExecuteAndWaitInterceptor.java

```
import net.sf.hibernate.SessionFactory;

import com.opensymphony.webwork.interceptor.BackgroundProcess;
import com.opensymphony.webwork.interceptor.ExecuteAndWaitInterceptor;
import com.opensymphony.xwork.ActionInvocation;

/**
 * The OpenSessionExecuteAndWaitInterceptor will obtain a Hibernate
 * Session Factory from a Spring.
 *
 * The session factory will then be passed to the BackgroundProcess,
 * to open a session, enable Spring's transaction management
 * capabilities, and bind the Session to the background thread.
 */
public class OpenSessionExecuteAndWaitInterceptor extends ExecuteAndWaitInterceptor
{
    SessionFactory sessionFactory;
```

```

    public SessionFactory getSessionFactory() {
        return sessionFactory;
    }

    public void setSessionFactory(SessionFactory sessionFactory) {
        this.sessionFactory = sessionFactory;
    }

    protected BackgroundProcess getNewBackgroundProcess(String arg0,
        ActionInvocation arg1, int arg2) {
        returnnew OpenSessionBackgroundProcess(arg0, arg1, arg2, sessionFactory);
    }
}

```

OpenSessionBackgroundProcess.java

```

import net.sf.hibernate.FlushMode;
import net.sf.hibernate.Session;
import net.sf.hibernate.SessionFactory;

import org.springframework.orm.hibernate.SessionFactoryUtils;
import org.springframework.orm.hibernate.SessionHolder;
import org.springframework.transaction.support.TransactionSynchronizationManager;

import com.opensymphony.webwork.interceptor.BackgroundProcess;
import com.opensymphony.xwork.ActionInvocation;

/**
 * The OpenSessionBackgroundProcess, when instantiated with a
 * HibernateSessionFactory, will open a session, enable Spring's transaction
 * management capabilities, and bind the Session to the background thread.
 */
public class OpenSessionBackgroundProcess extends BackgroundProcess {

    SessionFactory sessionFactory;

    Session openSession;

    public OpenSessionBackgroundProcess(String name,
        ActionInvocation invocation, int threadPriority,
        SessionFactory factory) {
        super(name, invocation, threadPriority);
        this.sessionFactory = factory;
    }

    protected void beforeInvocation() throws Exception {
        openSession = SessionFactoryUtils.getSession(sessionFactory, true);
        openSession.setFlushMode(FlushMode.NEVER);
        TransactionSynchronizationManager.bindResource(sessionFactory,
            new SessionHolder(openSession));
        super.beforeInvocation();
    }

    protected void afterInvocation() throws Exception {
        super.afterInvocation();
        TransactionSynchronizationManager.unbindResource(sessionFactory);
    }
}

```

```
        sessionFactoryUtils
            .closeSessionIfNecessary(openSession, sessionFactory);
    }

}
```

I18n Interceptor

This page last changed on Oct 18, 2005 by [digi9ten](#).

Content pulled from external source. Click [here](#) to refresh.

An interceptor that handles setting the locale specified in a session as the locale for the current action request. In addition, this interceptor will look for a specific HTTP request parameter and set the locale to whatever value is provided. This means that this interceptor can be used to allow for your application to dynamically change the locale for the user's session. This is very useful for applications that require multi-lingual support and want the user to be able to set his or her language preference at any point. The locale parameter is removed during the execution of this interceptor, ensuring that properties aren't set on an action (such as `request_locale`) that have no typical corresponding setter in your action.

For example, using the default parameter name, a request to **`foo.action?request_locale=en_US`**, then the locale for US English is saved in the user's session and will be used for all future requests.

Parameters

Content pulled from external source. Click [here](#) to refresh.

- `parameterName` (optional) - the name of the HTTP request parameter that dictates the locale to switch to and save in the session. By default this is **`request_locale`**
- `attributeName` (optional) - the name of the session key to store the selected locale. By default this is **`WW_TRANS_I18N_LOCALE`**

Extending the Interceptor

Content pulled from external source. Click [here](#) to refresh.

There are no known extensions points for this interceptor.

Examples

Content pulled from external source. Click [here](#) to refresh.

```
<action name="someAction" class="com.examples.SomeAction"><interceptor-ref  
name="il8n"/><interceptor-ref name="basicStack"/><result  
name="success">good_result.ftl</result></action>
```

Logger Interceptor

This page last changed on Oct 18, 2005 by [digi9ten](#).

Content pulled from external source. Click [here](#) to refresh.

This interceptor logs the the start and end of the execution an action (in English-only, not internationalized).

Parameters

Content pulled from external source. Click [here](#) to refresh.

There are no parameters for this interceptor.

Extending the Interceptor

Content pulled from external source. Click [here](#) to refresh.

There are no obvious extensions to the existing interceptor.

Examples

Content pulled from external source. Click [here](#) to refresh.

```
<!-- prints out a message before and after the immediate action execution --><action
name="someAction" class="com.examples.SomeAction"><interceptor-ref
name="completeStack"/><interceptor-ref name="logger"/><result
name="success">good_result.ftl</result></action><!-- prints out a message before any
more interceptors continue and after they have finished --><action name="someAction"
class="com.examples.SomeAction"><interceptor-ref name="logger"/><interceptor-ref
name="completeStack"/><result name="success">good_result.ftl</result></action>
```

Model Driven Interceptor

This page last changed on Oct 18, 2005 by [digi9ten](#).

Content pulled from external source. Click [here](#) to refresh.

Watches for ModelDriven actions and adds the action's model on to the value stack.

Note: The ModelDrivenInterceptor must come before the both StaticParametersInterceptor and ParametersInterceptor if you want the parameters to be applied to the model.

Parameters

Content pulled from external source. Click [here](#) to refresh.

- None

Extending the Interceptor

Content pulled from external source. Click [here](#) to refresh.

There are no known extension points to this interceptor.

Examples

Content pulled from external source. Click [here](#) to refresh.

```
<action name="someAction" class="com.examples.SomeAction"><interceptor-ref  
name="model-driven"/><interceptor-ref name="basicStack"/><result  
name="success">good_result.ftl</result></action>
```


Parameters Interceptor

This page last changed on Oct 20, 2005 by [plightbo](#).

Content pulled from external source. Click [here](#) to refresh.

This interceptor gets all parameters from `ActionContext#getParameters()` and sets them on the value stack by calling `{@link OgnlValueStack#setValue(String, Object)}`, typically resulting in the values submitted in a form request being applied to an action in the value stack. Note that the parameter map must contain a `String` key and often contains a `String[]` for the value.

Because parameter names are effectively OGNL statements, it is important that security be taken in to account. This interceptor will not apply any values in the parameters map if the expression contains an assignment (`=`), multiple expressions (`,`), or references any objects in the context (`#`). This is all done in the `{@link #acceptableName(String)}` method. In addition to this method, if the action being invoked implements the `{@link ParameterNameAware}` interface, the action will be consulted to determine if the parameter should be set.

In addition to these restrictions, a flag (`XWorkMethodAccessor#DENY_METHOD_EXECUTION`) is set such that no methods are allowed to be invoked. That means that any expression such as *person.doSomething()* or *person.getName()* will be explicitly forbidden. This is needed to make sure that your application is not exposed to attacks by malicious users.

While this interceptor is being invoked, a flag (`InstantiatingNullHandler#CREATE_NULL_OBJECTS`) is turned on to ensure that any null reference is automatically created - if possible. See the type conversion documentation and the `InstantiatingNullHandler` javadocs for more information.

Finally, a third flag (`XWorkConverter#REPORT_CONVERSION_ERRORS`) is set that indicates any errors when converting the the values to their final data type (`String[]` -> `int`) an unrecoverable error occurred. With this flag set, the type conversion errors will be reported in the action context. See the type conversion documentation and the `XWorkConverter` javadocs for more information.

If you are looking for detailed logging information about your parameters, turn on DEBUG level logging for this interceptor. A detailed log of all the parameter keys and values will be reported.

For more information on ways to restrict the parameter names allowed, see the `ParameterNameAware` javadocs:

Content pulled from external source. Click [here](#) to refresh.

This interface is implemented by actions that want to declare acceptable parameters. Works in conjunction with `{@link ParametersInterceptor}`. For example, actions may want to create a whitelist of parameters they will accept or a blacklist of parameters they will reject to prevent clients from setting other unexpected (and possibly dangerous) parameters.

Parameters

Content pulled from external source. Click [here](#) to refresh.

- None

Extending the Interceptor

Content pulled from external source. Click [here](#) to refresh.

The best way to add behavior to this interceptor is to utilize the `ParameterNameAware` interface in your actions. However, if you wish to apply a global rule that isn't implemented in your action, then you could extend this interceptor and override the `#acceptableName(String)` method.

Examples

Content pulled from external source. Click [here](#) to refresh.

```
<action name="someAction" class="com.examples.SomeAction"><interceptor-ref  
name="params"/><result name="success">good_result.ftl</result></action>
```

Prepare Interceptor

This page last changed on Oct 18, 2005 by [digi9ten](#).

Content pulled from external source. Click [here](#) to refresh.

This interceptor calls `prepare()` on actions which implement `Preparable`. This interceptor is very useful for any situation where you need to ensure some logic runs before the actual `execute` method runs.

A typical use of this is to run some logic to load an object from the database so that when parameters are set they can be set on this object. For example, suppose you have a `User` object with two properties: `id` and `name`. Provided that the `params` interceptor is called twice (once before and once after this interceptor), you can load the `User` object using the `id` property, and then when the second `params` interceptor is called the parameter `user.name` will be set, as desired, on the actual object loaded from the database. See the example for more info.

Parameters

Content pulled from external source. Click [here](#) to refresh.

- None

Extending the Interceptor

Content pulled from external source. Click [here](#) to refresh.

There are no known extension points to this interceptor.

Examples

Content pulled from external source. Click [here](#) to refresh.

```
<!-- Calls the params interceptor twice, allowing you to
pre-load data for the second time parameters are set -->
```

```
<action name="someAction" class="com.examples.SomeAction"><interceptor-ref  
name="params"/><interceptor-ref name="prepare"/><interceptor-ref  
name="basicStack"/><result name="success">good_result.ftl</result></action>
```

Content pulled from external source. Click [here](#) to refresh.

This is designed to solve a few simple issues related to wizard-like functionality in WebWork. One of those issues is that some applications have a application-wide parameters commonly used, such *pageLen* (used for records per page). Rather than requiring that each action check if such parameters are supplied, this interceptor can look for specified parameters and pull them out of the session.

This works by setting listed properties at action start with values from session/application attributes keyed after the action's class, the action's name, or any supplied key. After action is executed all the listed properties are taken back and put in session or application context.

To make sure that each execution of the action is consistent it makes use of session-level locking. This way it guarantees that each action execution is atomic at the session level. It doesn't guarantee application level consistency however there has yet to be enough reasons to do so. Application level consistency would also be a big performance overkill.

Note that this interceptor takes a snapshot of action properties just before result is presented (using a `{@link PreResultListener}`), rather than after action is invoked. There is a reason for that: At this moment we know that action's state is "complete" as it's values may depend on the rest of the stack and specifically - on the values of nested interceptors.

Parameters

Content pulled from external source. Click [here](#) to refresh.

- session - a list of action properties to be bound to session scope
- application - a list of action properties to be bound to application scope
- key - a session/application attribute key prefix, can contain following values:
 - ° CLASS - that creates a unique key prefix based on action namespace and action class, it's a default value

- ACTION - creates a unique key prefix based on action namespace and action name
- any other value is taken literally as key prefix
- type - with one of the following
 - start - means it's a start action of the wizard-like action sequence and all session scoped properties are reset to their defaults
 - end - means that session scoped properties are removed from session after action is run
 - any other value or no value means that it's in-the-middle action that is set with session properties before it's executed, and it's properties are put back to session after execution
- sessionReset - boolean value causing all session values to be reset to action's default values or application scope values, note that it is similliar to type="start" and in fact it does the same, but in our team it is sometimes semantically preferred. We use session scope in two patterns - sometimes there are wizzard-like action sequences that have start and end, and sometimes we just want simply reset current session values.

Extending the Interceptor

Content pulled from external source. Click [here](#) to refresh.

There are no know extension points for this interceptor.

Examples

Content pulled from external source. Click [here](#) to refresh.

```
<!-- As the filter and orderBy parameters are common for all my browse-type actions,
you can move control to the scope interceptor. In the session parameter you can
list
action properties that are going to be automatically managed over session. You
can
do the same for application-scoped variables-->
<action name="someAction" class="com.examples.SomeAction"><interceptor-ref
name="basicStack"/><interceptor-ref name="hibernate"/><interceptor-ref
name="scope"><param name="session">filter,orderBy</param></interceptor-ref><result
name="success">good_result.ftl</result></action>
```

Servlet Config Interceptor

This page last changed on Oct 18, 2005 by [digi9ten](#).

Content pulled from external source. Click [here](#) to refresh.

An interceptor which sets action properties based on the interfaces an action implements. For example, if the action implements `ParameterAware` then the action context's parameter map will be set on it.

This interceptor is designed to set all properties an action needs if it's aware of servlet parameters, the servlet context, the session, etc. Interfaces that it supports are:

- `ServletContextAware`
- `ServletRequestAware`
- `ServletResponseAware`
- `ParameterAware`
- `SessionAware`
- `ApplicationAware`
- `PrincipalAware`

Parameters

Content pulled from external source. Click [here](#) to refresh.

- None

Extending the Interceptor

Content pulled from external source. Click [here](#) to refresh.

There are no known extension points for this interceptor.

Examples

Content pulled from external source. Click [here](#) to refresh.

```
<action name="someAction" class="com.examples.SomeAction"><interceptor-ref  
name="servlet-config"/><interceptor-ref name="basicStack"/><result  
name="success">good_result.ftl</result></action>
```

Static Parameters Interceptor

This page last changed on Oct 18, 2005 by [digi9ten](#).

Content pulled from external source. Click [here](#) to refresh.

This interceptor populates the action with the static parameters defined in the action configuration. If the action implements Parameterizable, a map of the static parameters will be also be passed directly to the action.

Parameters are typically defined with `<param>` elements within `xwork.xml`.

Parameters

Content pulled from external source. Click [here](#) to refresh.

- None

Extending the Interceptor

Content pulled from external source. Click [here](#) to refresh.

There are no extension points to this interceptor.

Examples

Content pulled from external source. Click [here](#) to refresh.

```
<action name="someAction" class="com.examples.SomeAction"><interceptor-ref  
name="static-params"/><result name="success">good_result.ftl</result></action>
```

Timer Interceptor

This page last changed on Oct 18, 2005 by [digi9ten](#).

Content pulled from external source. Click [here](#) to refresh.

This interceptor logs the amount of time in milliseconds. In order for this interceptor to work properly, the logging framework must be set to at least the [INFO](#) level. This interceptor relies on the [Commons Logging API](#) to report its execution-time value.

Parameters

Content pulled from external source. Click [here](#) to refresh.

TODO: Describe the paramters for this Interceptor.

Extending the Interceptor

Content pulled from external source. Click [here](#) to refresh.

TODO: Discuss some possible extension of the Interceptor.

Examples

Content pulled from external source. Click [here](#) to refresh.

```
<!-- records only the action's execution time --><action name="someAction"
class="com.examples.SomeAction"><interceptor-ref
name="completeStack"/><interceptor-ref name="timer"/><result
name="success">good_result.ftl</result></action><!-- records action's execution time
as well as other interceptors--><action name="someAction"
class="com.examples.SomeAction"><interceptor-ref name="timer"/><interceptor-ref
name="completeStack"/><result name="success">good_result.ftl</result></action>
```

Token Interceptor

This page last changed on Oct 18, 2005 by [digi9ten](#).

Content pulled from external source. Click [here](#) to refresh.

Ensures that only one request per token is processed. This interceptor can make sure that back buttons and double clicks don't cause un-intended side affects. For example, you can use this to prevent careless users who might double click on a "checkout" button at an online store. This interceptor uses a fairly primitive technique for when an invalid token is found: it returns the result **invalid.token**, which can be mapped in your action configuration. A more complex implementation, `TokenSessionStoreInterceptor`, can provide much better logic for when invalid tokens are found.

Note: To set a token in your form, you should use the **token tag**. This tag is required and must be used in the forms that submit to actions protected by this interceptor. Any request that does not provide a token (using the token tag) will be processed as a request with an invalid token.

Parameters

Content pulled from external source. Click [here](#) to refresh.

- None

Extending the Interceptor

Content pulled from external source. Click [here](#) to refresh.

While not very common for users to extend, this interceptor is extended by the `TokenSessionStoreInterceptor`. The `#handleInvalidToken` and `#handleValidToken` methods are protected and available for more interesting logic, such as done with the token session interceptor.

Examples

Content pulled from external source. Click [here](#) to refresh.

```
<action name="someAction" class="com.examples.SomeAction"><interceptor-ref  
name="token"/><interceptor-ref name="basicStack"/><result  
name="success">good_result.ftl</result></action>
```

Token Session Interceptor

This page last changed on Oct 18, 2005 by [digi9ten](#).

Content pulled from external source. Click [here](#) to refresh.

This interceptor builds off of the `TokenInterceptor`, providing advanced logic for handling invalid tokens. Unlike the normal token interceptor, this interceptor will attempt to provide intelligent fail-over in the event of multiple requests using the same session. That is, it will block subsequent requests until the first request is complete, and then instead of returning the `invalid.token` code, it will attempt to display the same response that the original, valid action invocation would have displayed if no multiple requests were submitted in the first place.

Parameters

Content pulled from external source. Click [here](#) to refresh.

- None

Extending the Interceptor

Content pulled from external source. Click [here](#) to refresh.

There are no known extension points for this interceptor.

Examples

Content pulled from external source. Click [here](#) to refresh.

```
<action name="someAction" class="com.examples.SomeAction"><interceptor-ref  
name="token-session"/><interceptor-ref name="basicStack"/><result  
name="success">good_result.ftl</result></action>
```

Validation Interceptor

This page last changed on Oct 18, 2005 by [digi9ten](#).

Content pulled from external source. Click [here](#) to refresh.

This interceptor runs the action through the standard validation framework, which in turn checks the action against any validation rules (found in files such as *ActionClass-validation.xml*) and adds field-level and action-level error messages (provided that the action implements `com.opensymphony.xwork.ValidationAware`). This interceptor is often one of the last (or second to last) interceptors applied in a stack, as it assumes that all values have already been set on the action.

Note that this has nothing to do with the `com.opensymphony.xwork.Validateable` interface and simply adds error messages to the action. The workflow of the action request does not change due to this interceptor. Rather, this interceptor is often used in conjunction with the **workflow** interceptor.

Parameters

Content pulled from external source. Click [here](#) to refresh.

- None

Extending the Interceptor

Content pulled from external source. Click [here](#) to refresh.

There are no known extension points for this interceptor.

Examples

Content pulled from external source. Click [here](#) to refresh.

```
<action name="someAction" class="com.examples.SomeAction"><interceptor-ref  
name="params"/><interceptor-ref name="validation"/><interceptor-ref
```

```
/><result name="success">good_result.ftl</result></action>
```


Workflow Interceptor

This page last changed on Oct 18, 2005 by [digi9ten](#).

Content pulled from external source. Click [here](#) to refresh.

An interceptor that does some basic validation workflow before allowing the interceptor chain to continue. The order of execution in the workflow is:

1. If the action being executed implements `Validateable`, the action's `Validateable#validate()` validate method is called.
2. Next, if the action implements `ValidationAware`, the action's `ValidationAware#hasErrors()` `hasErrors` method is called. If this method returns `true`, this interceptor stops the chain from continuing and immediately returns `Action#INPUT`

Note: if the action doesn't implement either interface, this interceptor effectively does nothing. This interceptor is often used with the **validation** interceptor. However, it does not have to be, especially if you wish to write all your validation rules by hand in the `validate()` method rather than in XML files.

Parameters

Content pulled from external source. Click [here](#) to refresh.

- None

Extending the Interceptor

Content pulled from external source. Click [here](#) to refresh.

There are no known extension points for this interceptor.

Examples

Content pulled from external source. Click [here](#) to refresh.

```
<action name="someAction" class="com.examples.SomeAction"><interceptor-ref  
name="params"/><interceptor-ref name="validation"/><interceptor-ref  
name="workflow"/><result name="success">good_result.ftl</result></action>
```

Internationalization

This page last changed on Oct 24, 2005 by [digi9ten](#).

WebWork supports internationalization (in short, i18n) in two different places: the UI tags and the action/field error messages.

- [UI Tags](#)
- [Validation Examples](#)

Resource bundles are searched in the following order:

- ActionClass.properties
- BaseClass.properties (all the way to Object.properties)
- Interface.properties (every interface and sub-interface)
- package.properties (every of every base class, all the way to java/lang/package.properties)

To display i18n text, you can use a call to `getText()` in the property tag, or any other tag such as the UI tags (this is especially useful for labels of UI tags):

```
<ww:property value="getText('some.key')"/>
```

You may also use the text tag:


```
<ww:text name="'some.key'"/>
```

Also, note that there is an i18n tag that will push a resource bundle on to the stack, allowing you to display text that would otherwise not be part of the resource bundle search hierarchy mentioned previously.

```
<ww:i18n name="'some/package/bundle'"><ww:text name="'some.key'"/></ww:i18n>
```



Internationalization in SiteMesh decorators is possible, but there are a few quirks about it. Check out the [SiteMesh](#) page to learn how to be integrate WebWork and SiteMesh,

	including integration tips.
	Internationalization Interceptor Coming up in the next release of webwork (2.2) is the I18NInterceptor to switch the Locale of a request on the fly. You can check it out in the Cookbook under Transparent web-app I18N .

Using a master application catalog

Struts users should be familiar with the application.properties resource bundle, where you can put all the messages in the application that are going to be translated. WebWork2, though, splits the resource bundles per action or model class, and you may end up with duplicated messages in those resource bundles. A quick fix for that is to create a file called ActionSupport.properties in com/opensymphony/xwork and put it on your classpath. This will only work well if all your actions subclass ActionSupport.

Global resource bundles can also be specified via setting the **webwork.locale** in [webwork.properties](#).

Inversion of Control

This page last changed on Oct 17, 2005 by [plightbo](#).



These documents are out of date. As of WebWork 2.2, the WebWork IoC container has been deprecated (though not removed) and the WebWork team recommends you use [Spring](#) for all your IoC needs

Inversion of control is a way to handle dependencies of objects. In WebWork, objects that have their dependencies managed are called "components". For an overview of Inversion of Control (also referred to now as Dependency Injection), please read Martin Fowler's article on IoC at <http://www.martinfowler.com/articles/injection.html>. Besides WebWork's IoC container, there are numerous other containers available for you to use, including [Spring](#) and [Pico](#).

- [IoC Overview](#)
- [Xwork's Component Architecture](#)
- [How Webwork Uses Components](#)
- [Configuration of Components in Webwork and XWork](#)

Components

This page last changed on Oct 25, 2005 by [plightbo](#).



These documents are out of date. As of WebWork 2.2, the WebWork IoC container has been deprecated (though not removed) and the WebWork team recommends you use [Spring](#) for all your IoC needs

Overview

WebWork builds on XWork's component implementation by providing lifecycle management of component objects and then making these components available to your action classes (or any other user code for that matter) as required.

Two types of classes in WebWork can use an enabler interface for inversion of control: Actions and Components. In order for an Action class to have its components set, the ComponentInterceptor must be made available for the Action to set those resources. In turn, if those components require other components to be initialized and set for their own use, those initializations take place at the time the ComponentInterceptor intercepts the action as well.

Scopes and Lifecycle

Components can be configured to exist across three different scopes in WebWork:

1. for the duration of a single request,
2. across a user session, or
3. for the entire lifetime of the web application.

WW:WebWork lazy loads components, meaning that components, no matter what scope, are initialized at the time they are used and disposed of at the end of the given lifecycle of that scope. Thus, an application scoped component, for example, will be initialized the first time a user makes a request to an action that implements the

enabler interface of that component and will be disposed of at the time the application closes.

While components are allowed to have dependencies on other components they must not depend on another component that is of a narrower scope. So, for example, a session component cannot depend on a component that is only of request scope.

All components must be registered in the components.xml file, which is discussed in the Configuration section.

Obtaining a ComponentManager

During any request there are three component managers in existence, one for each scope. They are stored as an attribute called "DefaultComponentManager" in their respective scope objects. So if for example you need to retrieve the ComponentManager object for the request scope, the following code will do the trick:

```
ComponentManager cm = (ComponentManager)
request.getAttribute("DefaultComponentManager");
```

IoC Configuration

This page last changed on Oct 25, 2005 by [plightbo](#).



These documents are out of date. As of WebWork 2.2, the WebWork IoC container has been deprecated (though not removed) and the WebWork team recommends you use [Spring](#) for all your IoC needs

Configuration - web.xml

To configure WebWork's component management, the following lines must be added in the appropriate places to web.xml:

```
<filter><filter-name>container</filter-name><filter-class>com.opensymphony.webwork.lifecycle.Rec
modify appropriately --></filter-mapping><!-- Optionally you may instead apply the
filter to EVERY URI instead of just *.action. --><!-- You might want to do this for
example: --><!-- * your page flow goes to a jsp directly (as opposed to only
*.action URIs) --><!-- * the jsp has an action in it to be run with the ww:action
tag --><!-- * the action in the jsp implements any enabler interfaces to get
components served to it --><!-- The reason: (Per Patrick Lightbody) --><!-- "The
components work by looking for a ComponentManager in the request --><!-- attributes.
It gets placed there by a filter. If it is on *.action and a --><!-- request comes
in through a JSP, the filter won't be applied and it will --><!-- never work."
--><!-- The overhead in doing this is small, so don't worry overly much about the
--><!-- performance of this. --><!-- <filter-mapping> --><!--
<filter-name>container</filter-name> --><!-- <url-pattern>*/</url-pattern>
--><!-- </filter-mapping>
--><listener><listener-class>com.opensymphony.webwork.lifecycle.SessionLifecycleListener</listen
```

These settings allow WebWork to manage components across the application, session and request scopes. Note that even if one or more of the scopes are not required by your application, all three scopes need to be specified in web.xml for WebWork's component management to function correctly.

Configuration - xwork.xml

The ComponentInterceptor class is used to apply the IoC pattern to XWork actions (ie, to supply components to actions). The ComponentInterceptor should be declared in

the <interceptors> block of xwork.xml as follows:

```
<interceptor name="component"  
    class="com.opensymphony.xwork.interceptor.component.ComponentInterceptor"/>
```

You should ensure that any actions that are to be supplied with components have this interceptor applied. (See OS:XWork Interceptors for information on how to apply interceptors to actions.)

If you want to apply IoC to objects other than actions or other components, you will need to use the ComponentManager object directly.

Note too, that the ComponentInterceptor is applied as part of the webwork defaultStack. Thus, if you are applying the defaultStack to the action, you would include the ComponentInterceptor.

Configuration - components.xml

The components.xml file is used to specify the components that are to be available. The components specified here are loaded into XWork's ComponentManager and are then made available to any actions that are an instance of the specified enabler. The components.xml file must be placed in the root of the classpath (ie, in the WEB-INF/classes directory).

Here is an example components.xml file that configures a Counter component. The Counter object will live in session scope, and will be passed to any objects that are enabled due to their implementing the CounterAware interface:

```
<components><component><scope>session</scope><class>com.opensymphony.webwork.example.counter.Cou
```

Each component must have the following three attributes:

- *scope*: Valid values are *application*, *session* and *request*. This determines the component's lifetime. Application scope components will be created when the webapp starts up, and they will survive for the whole lifetime of the webapp. Session scoped components exist for the duration of a user session, while components in request scope only last for the duration of a single client request.
- *class*: This specifies the component's class. An instance of this object will live for

the duration of the specified scope, and will be made available to any actions (or other code) as required. Note that components are lazy-loaded, so if nothing makes use of the component during its lifetime, the component will never actually be instantiated. At the moment components must have a zero argument constructor.

- *enabler*: Any actions that are an instance of the enabler class or interface will be passed an instance of the component.

Note that while components are allowed to have dependencies on other components they must not depend on another component that is of a narrower scope. So for example, a session component cannot depend on a component that is only of request scope.



These documents are out of date. As of WebWork 2.2, the WebWork IoC container has been deprecated (though not removed) and the WebWork team recommends you use [Spring](#) for all your IoC needs

Overview

In many applications you have component objects that are required by a given class to use. In a nutshell, the IoC pattern allows a parent object (in the case of Webwork, XWork's ComponentManager instance) to give a resource Object to the action Object that needs it (usually an action, but it could be any object that implements the appropriate *enabler*) rather than said Object's needing to obtain the resource itself.

There are two ways of implementing IoC: Instantiation and using an enabler interface. With instantiation, a given action Object is instantiated with the resource Object as a constructor parameter. With enablers interfaces, the action will have an interface with a method, say "setComponent(ComponentObject r);" that will allow the resource to be passed to said action Object after it is instantiated. The ComponentObject is passed, because the Object implements the given interface. XWork uses *enablers* to pass components.

Why IoC?

So why is IoC useful? It means that you can develop components (generally services of some sort) in a top-down fashion, without the need to build a registry class that the client must then call to obtain the component instance.

Traditionally when implementing services you are probably used to following steps similar to these:

1. Write the component (eg an `ExchangeRateService`)
2. Write the client class (eg an `XWork` action)
3. Write a registry class that holds the component object (eg `Registry`)
4. Write code that gives the component object to the registry (eg `Registry.registerService(new MyExchangeRateService())`)
5. Use the registry to obtain the service from your client class (eg `ExchangeRateService ers = Registry.getExchangeRateService()`)
6. Make calls to the component from the client class (eg `String baseCurrencyCode = ers.getBaseCurrency()`)

Using IoC, the process is reduced to the following:

1. Write the component class (eg an `ExchangeRateService`)
2. Register the component class with `XWork` (eg `componentManager.addEnabler(MyExchangeRateService, ExchangeRateAware)`)
3. Write the client class, making sure it implements the enabling interface (eg an `XWork` action that implements `ExchangeRateAware`)
4. Access the component instance directly from your client action (eg `String baseCurrencyCode = ers.getBaseCurrency()`)

More advantages of Inversion of Control are the following:

1. Testability - You can more easily test your objects by passing mock objects using the enabler method rather than needing to create full containers that allow your objects to get the components they need.
2. A component describes itself. When you instantiate a component, you can easily determine what dependencies it requires without looking at the source or using trial and error.
3. Dependencies can be discovered easily using reflection. This has many benefits ranging from diagram generation to runtime optimization (by determining in advance which components will be needed to fulfill a request and preparing them asynchronously, for example).
4. Avoids the super-uber-mega-factory pattern where all the components of the app are held together by a single class that is directly tied back to other domain specific classes, making it hard to 'just use that one class'.
5. Adheres to Law of Demeter. Some people think this is silly, but in practise I've found it works much better. Each class is coupled to only what it actually uses (and it should never use too much) and no more. This encourages smaller responsibility specific classes which leads to cleaner design.
6. Allows context to be isolated and explicitly passed around. `ThreadLocals` may be ok in a web-app, but they aren't well suited for high concurrency async applications (such as message driven applications).



These documents are out of date. As of WebWork 2.2, the WebWork IoC container has been deprecated (though not removed) and the WebWork team recommends you use [Spring](#) for all your IoC needs

Writing Component Classes

In [XW:XWork](#) the actual component class can be virtually anything you like. The only constraints on it are that it must be a concrete class with a default constructor so that XWork can create instances of it as required. Optionally, a component may implement the Initializable and/or Disposable interfaces so it will receive lifecycle events just after it is created or before it is destroyed. Simply:

```
public class MyComponent implements Initializable, Disposable {
    public void init () {
        //do initialization here
    }

    public void dispose() {
        //do any clean up necessary before garbage collection of this component
    }
}
```

Component Dependencies

One feature that is not immediately obvious is that it is possible for components to depend on other components. For example if the ExchangeRateService described above depended on a Configuration component, XWork will pass the Configuration component through to the ExchangeRateService instance after ExchangeRateService is instantiated. Note that XWork automatically takes care of initializing the components in the correct order, so if A is an action or component that depends on B and C, and B depends on C and if A, B, and C have not been previously instantiated, the ComponentManager will in the following order:

1. Instantiate C and call its `init()` method if it implements `Initializable`.
2. Instantiate B, then using the enabler method, set C to be used by B
3. Call B's `init()` method, if it implements `Initializable`.
4. Set B using B's enabler method to be used by A.

And so on and so forth. Of course, if there are instances of B or C that would be reused in this case, those instances would be passed using the enabler method rather than a new instance.

Writing Enablers

An enabler should consist of just a single method that accepts a single parameter. The parameter class should either be the component class that is to be enabled, or one of the component's superclasses. XWork does not care what the name of the enabler's method is.

Here is an example of what the `ExchangeRateAware` enabler might look like:

```
public interface ExchangeRateAware {  
    public void setExchangeRateService(ExchangeRateService exchangeRateService);  
}
```

Note that typically an enabler would be an interface, however there is nothing to prevent you from using a class instead if you so choose.

Writing "Enabler-aware" Actions

All an action needs to do is implement the relevant enabler interface. XWork will then call the action's enabler method just prior to the action's execution. As a simple example:

```
public class MyAction extends ActionSupport implements ExchangeRateAware {  
    ExchangeRateService ers;  
  
    public void setExchangeRateService(ExchangeRateService exchangeRateService) {  
        ers = exchangeRateService;  
    }  
}
```

```

    public String execute() throws Exception {
        System.out.println("The base currency is " + ers.getBaseCurrency());
    }
}

```

If you have an object that is not an action or another component, you must explicitly tell XWork to supply any enabled components to your object by calling `componentManager.initializeObject(enabledObject);`

Using an external reference resolver

You can also use an external reference resolver in XWork, i.e., references that will be resolved not by XWork itself. One such example is using an external resolver to integrate XWork with the [Spring Framework](#)

You just need to write an external reference resolver and then tell XWork to use it in the package declaration:

```

<package
  name="default"
  externalReferenceResolver="com.atlassian.xwork.ext.SpringServletContextReferenceResolver">

```

Now, to use external references you do something like this:

```

<external-ref name="foo">Foo</external-ref>

```

Where the name attribute is the setter method name and Foo is the reference to lookup.

For more details and sample code about this integration, take a look at the javadocs to the `com.opensymphony.xwork.config.ExternalReferenceResolver` class (unfortunately unavailable online) and at [XW-122](#)

-Chris

J2SE 5 Support

This page last changed on Oct 17, 2005 by [plightbo](#).

Currently only in beta-status, an **xwork-tiger** project exists that is starting to add some basic J2SE 5 ("Tiger") support to WebWork. Currently, the only Java 5 implementation in xwork-tiger.jar is a Map and Collection support using generics.

In short, instead of specifying the types found in collections and maps as documented in [Type Conversion](#), **the collection's generic type is used**. This means you most likely don't need any **ClassName-conversion.properties** files.

JasperReports

This page last changed on Oct 30, 2005 by [plightbo](#).

TODO: document JasperReports integration.

TODO: General document describing how JSP works with WebWork. It should cover:

- What result type to use (dispatcher)
- What tags are available (refer to the general tag documentation, don't re-document them here)
- General tips and tricks w/ JSP
- Common pitfalls

OGNL is the Object Graph Navigation Language - see <http://www ognl.org> for the full documentation of OGNL. In this document we will only show a few examples of OGNL features that co-exist with Webwork.

- ## Webwork with OGNL

```
|  
  
|  
|--request  
|--application  
context map--- --OgnlValueStack(root)  
|--session  
|--attr  
|--parameters
```

Page 299

```
<ww: property value="myBean.myProperty"/>
```

For sessions,request, and the rest that lie in the context map:

```
ActionContext.getContext().getSession().put("mySessionPropKey", mySessionObject);
```

```
<ww: property value="#session.mySessionPropKey"/> or  
<ww: property value="#session\['mySessionPropKey'\]"/><ww: property  
value="#attr.mySessionPropKey"/>
```

Collections (Maps, Lists, Sets)

Dealing with collections(maps, lists, and sets) in webwork comes often, so here are a few examples using the select tag:

Syntax for list: {e1,e2}. This creates a List containing the String "name1" and "name2".

```
<webwork:select label="'lebal'" name="'nmae'" list="{ 'name1','name2' }" />
```

Syntax for map: #{key1:value1,key2:value2}. This creates a map that maps the string "foo" to the string "foovalue" and "bar" to the string "barvalue":

```
<webwork:select label="'lebal'" name="'nmae'" list="#{'foo':'foovalue',  
'bar':'barvalue'}" />
```

You may need to determine if an element exists in a collection. You can accomplish this with the operations `in` and `not in`

```
<ui:if test="'foo' in {'foo','bar'}">  
    muhahaha  
</ui:if><ui:else>  
    boo  
</ui:else><ui:if test="'foo' not in {'foo','bar'}">  
    muhahaha  
</ui:if><ui:else>  
    boo  
</ui:else>
```

To select a subset of a collection (called projection), you can use a wildcard within the collection.

- ? - All elements matching the selection logic
- ^ - Only the first element matching the selection logic
- \$ - Only the last element matching the selection logic

To obtain a subset of just male relatives from the object person:

```
person.relatives.{? #this.gender == 'male'}
```

Lambda Expressions

OGNL supports basic lambda expression syntax enabling you to write simple functions.

For example:

For all you math majors who didn't think you would ever see this one again.

Fibonacci: if $n=0$ return 0; elseif $n=1$ return 1; else return $\text{fib}(n-2)+\text{fib}(n-1)$;

$\text{fib}(0) = 0$

$\text{fib}(1) = 1$

$\text{fib}(11) = 89$

The lambda expression is everything inside the brackets. The `#this` variable holds the argument to the expression, which is initially starts at 11.

```
<ww:property value="#fib =:[#this==0 ? 0 : #this==1 ? 1 :  
#fib(#this-2)+#fib(#this-1)], #fib(11)" />
```

XWork-specific language features

The biggest addition that XWork provides on top of OGNL is the support for the ValueStack. While OGNL operates under the assumption there is only one "root", XWork's ValueStack concept requires there be many "roots".

For example, suppose we are using standard OGNL (not using XWork) and there are two objects in the OgnlContext map: "foo" -> foo and "bar" -> bar and that the foo object is also configured to be the single **root** object. The following code illustrates how OGNL deals with these three situations:

```
#foo.blah // returns foo.getBlah()
#bar.blah // returns bar.getBlah()
blah      // returns foo.getBlah() because foo is the root
```

What this means is that OGNL allows many objects in the context, but unless the object you are trying to access is the root, it must be prepended with a namespaces such as @bar. Now let's talk about how XWork is a little different...

In XWork, the entire ValueStack is the root object in the context. But rather than having your expressions get the object you want from the stack and then get properties from that (ie: peek().blah), XWork has a special OGNL PropertyAccessor that will automatically look at the all entries in the stack (from the top down) until it finds an object with the property you are looking for.

For example, suppose the stack contains two objects: Animal and Person. Both objects have a "name" property, Animal has a "species" property, and Person has a "salary" property. Animal is on the top of the stack, and Person is below it. The follow code fragments help you get an idea of what is going on here:

```
species    // call to animal.getSpecies()
salary     // call to person.getSalary()
name       // call to animal.getName() because animal is on the top
```

In the last example, there was a tie and so the animal's name was returned. Usually this is the desired effect, but sometimes you want the property of a lower-level object. To do this, XWork has added support for indexes on the ValueStack. All you have to do is:

```
\[0\].name    // call to animal.getName()  
\[1\].name    // call to person.getName()
```

Accessing static properties

OGNL supports accessing static properties as well as static methods. As the OGNL docs point out, you can explicitly call statics by doing the following:

```
@some.package.ClassName@FOO_PROPERTY  
@some.package.ClassName@someMethod()
```

However, XWork allows you to avoid having to specify the full package name and call static properties and methods of your action classes using the "vs" prefix:

```
@vs@FOO_PROPERTY  
@vs@someMethod()  
  
@vs1@FOO_PROPERTY  
@vs1@someMethod()  
  
@vs2@BAR_PROPERTY  
@vs2@someOtherMethod()
```

"vs" stands for "value stack". The important thing to note here is that if the class name you specify is just "vs", the class for the object on the top of the stack is used. If you specify a number after the "vs" string, an object's class deeper in the stack is used instead.

Differences from the WebWork 1.x EL

Besides the examples and descriptions given above, there are a few major changes in the EL since WebWork 1.x. The biggest one is that properties are no longer accessed

with a forward slash (/) but with a dot (.). Also, rather than using ".." to traverse down the stack, we now use "[n]" where n is some positive number. Lastly, in WebWork 1.x one could access special named objects (the request scope attributes to be exact) by using "@foo", but now special variables are accessed using "#foo". However, it is important to note that "#foo" does NOT access the request attributes. Because XWork is not built only for the web, there is no concept of "request attributes", and thus "#foo" is merely a request to another object in the OgnlContext other than the root.

Old Expression	New Expression
foo/blah	foo.blah
foo/someMethod()	foo.someMethod()
../bar/blah	[1].bar.blah
@baz	not directly supported, but #baz is similar
.	top or [0]

WebWork-specific named objects

name	value
#parameters['foo'] or #parameters.foo	request parameter ['foo'] (request.getParameter())
#request['foo'] or #request.foo	request attribute ['foo'] (request.getAttribute())
#session['foo'] or #session.foo	session attribute 'foo'
#application['foo'] or #application.foo	ServletContext attributes 'foo'
#attr['foo'] or #attr.foo	Access to PageContext if available, otherwise searches request/session/application respectively

Related Tools

This page last changed on Oct 17, 2005 by [plightbo](#).

Content pulled from external source. Click [here](#) to refresh.

WebWork comes with various related tools included in the webwork jar file. You can access these tools by simply unpacking the WebWork distribution and running **java -jar webwork.jar**. WebWork will automatically include all jars in the same directory as the webwork.jar file as well as all jars in the *lib* directory. This means you can invoke these tools either from within the standard directory structure found in the WebWork distribution, or from within your WEB-INF/lib directory.

You can access the help information for these tools by simply running the jar without any arguments.

1. [SiteGraph](#)
2. [QuickStart](#)

WebWork provides a quick way to get started called QuickStart. QuickStart is essentially a combination of a few technologies and some general conventions for developing web applications. What it lets you do is write applications without the need to even compile your sources, let alone have to deploy and redeploy them after every change. Instead, you can now develop your web applications just like if you were writing perl or PHP – on the fly and as quickly as you can think.

How to Use It

QuickStart is included in the WebWork distribution and can be launched by simply running **java -jar webwork.jar quickstart:mywebapp**. At this point you can access <http://localhost:8080/mywebapp> and begin developing your application. **At this time, QuickStart requires Java 1.5.**

OK, it's a little more work than that, but not much more. QuickStart assumes the following directory structure:

- webwork
 - lib - all your required libs, usually the ones you would put in WEB-INF/lib
 - webapps
 - mywebapp
 - src
 - java - your java sources that would normally be compiled to WEB-INF/classes
 - webapp
 - WEB-INF
 - classes - any additional configuration if you'd like
 - web.xml
 - webwork.jar
 - launcher.jar

You can quickly get started by copying one of the existing webapps in the WebWork distribution.

Once you have it up and running, you are free to change your classes, JSPs, template files, and other files on the fly – all without compiling or redeploying. Some files, such as web.xml, will require that you restart QuickStart for the changes to take affect. Similarly, some frameworks, such as Hibernate, do not offer the full class-reloading support that WebWork does. Your mileage may vary, but we think no matter what you'll love developing in QuickStart.

How It Works

QuickStart works by using the combination of WebWork's "share nothing" (or rather, "share very little") architecture, an embedded Jetty server, some advanced class loading, and the Eclipse Java compiler (don't worry, the Eclipse IDE is not required!)

Running webwork.jar bootstraps the classpath and includes every jar found in the **lib** directory. It also includes webwork.jar, of course. It then invokes the QuickStart application. This, in turn, starts a Jetty server that is configured to the webapp specified in the **quickstart:xxx** argument.

The Jetty server's context ClassLoader is specified as a custom ClassLoader that looks at the source files in **webapps/xxx/src/java** and compiles them on the fly. These classes are also reloaded whenever a change is detected.


Because WebWork creates a new action on every request, reloading the classes works great. You are free to change the entire class schema (methods, fields, inheritance, etc). Because none of the objects are cached or stored in long-term storage, you usually won't run in to any problems here.

Common Pitfalls

While WebWork is pretty good about making class reloading in QuickStart easy, other libraries and code are not. As a general rule of thumb, if any objects have long term state (singleton, session scope, etc), they will *not* be reloaded. The reloaded classes

will *only* take affect after a new instance has been created with the *new* keyword or through reflection.

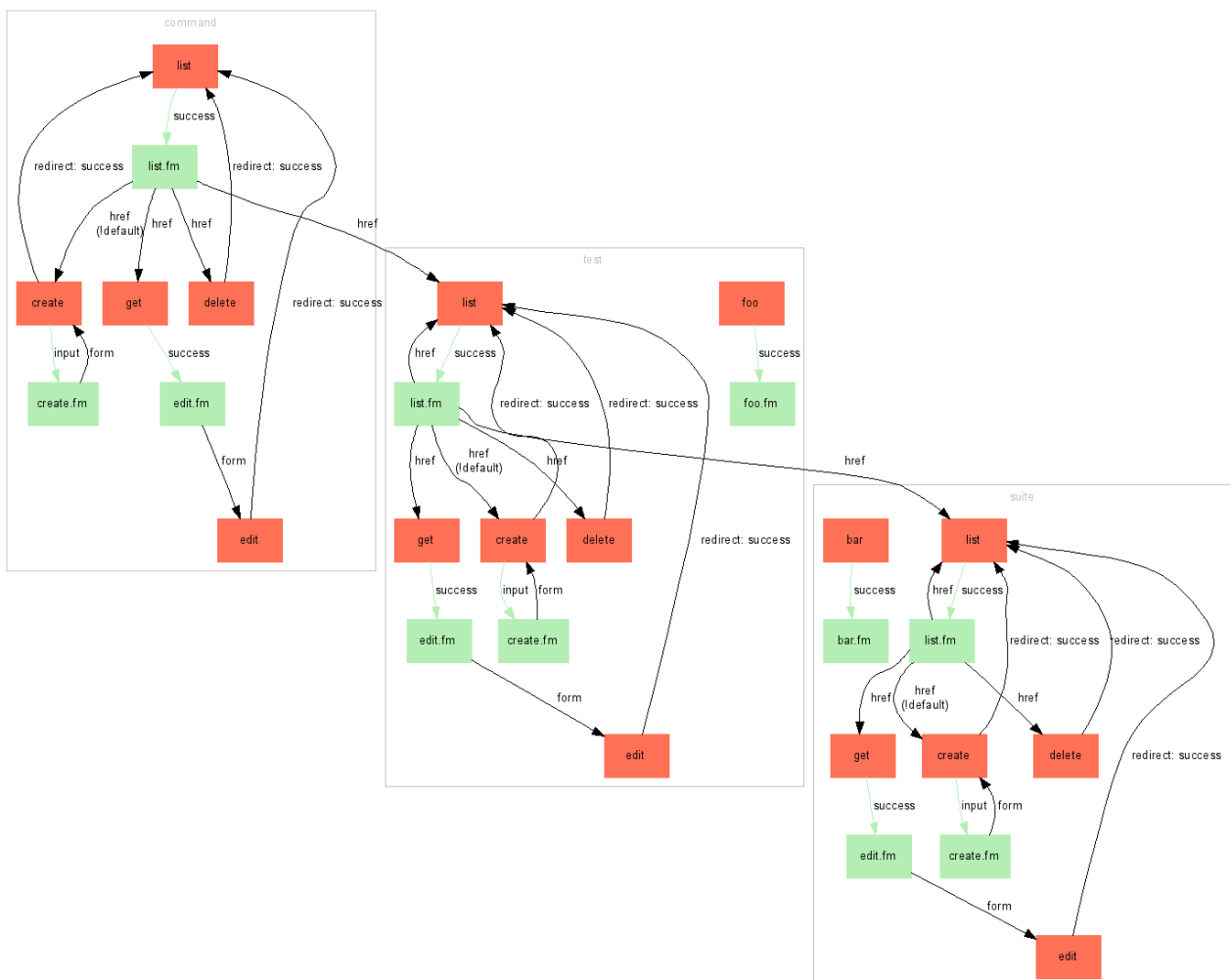
For example, Hibernate has been found to store references to the objects it persists for long periods of time because of it's caching mechanism. It also happens to hold a reference to the Class instance itself. This makes it very difficult, if not impossible, to allow you to change your models on the fly.

	<p>Most problems will manifest themselves through a <code>ClassCastException</code>, or some other weird class-related error. You may even find yourself banging your head against the wall because some <code>Foo</code> instance can't be cast to the <code>Foo</code> class. This is the biggest challenge with using QuickStart and can best be mitigated by using libraries and code that share very little state.</p>
---	---

A final word of warning: QuickStart is not meant for production use, or even to be used as the sole environment for application development. Rather, it is meant to help you quickly develop proof-of-concepts see results quickly. We recommend you always at least test in other applications servers, such as Tomcat, Resin, or even standalone Jetty.

Introduction

WebWork comes with a utility called SiteGraph. SiteGraph is used to generate graphical diagrams representing the flow of your web application. It does this by parsing your configuration files, action classes, and view (JSP, Velocity, and FreeMarker) files. An example of a typical output of SiteGraph is provided (for the full size, click [here](#)):



Additional information can be found in the JavaDocs:

Content pulled from external source. Click [here](#) to refresh.

SiteGraph is a tool that renders out GraphViz-generated images depicting your WebWork-powered web application's flow. SiteGraph requires GraphViz be installed and that the "dot" executable be in your command path. You can find GraphViz at <http://www.graphviz.org>.

Understanding the Output

There are several key things to notice when looking at the output from SiteGraph:

- Boxes: those shaded red indicate an action; those shaded green indicate a view file (JSP, etc).
- Links: arrows colored green imply that no new HTTP request is being made; black arrows indicate a new HTTP request.
- Link labels: labels may sometimes contain additional useful information. For example, a label of **href** means that the link behavior is that of a hyper-text reference. The complete label behaviors are provided:
 - **href** - a view file references an action by name (typically ending with the extension ".action")
 - **action** - a view file makes a call to the Action Tag
 - **form** - a view file is linked to an action using the [Form Tag](#)
 - **redirect** - an action is redirecting to another view or action
 - **! notation** - a link to an action overrides the method to invoke

Requirements

SiteGraph requires that your view files be structured in a very specific way. Because it has to read these files, only certain styles are supported. The requirements are:

- The JSP tags must use the "ww" namespace.
 - In JSP: `<ww:xxx/>`
 - In FreeMarker: `<@ww.xxx/>`
 - In Velocity: N/A
- Use of the [Form Tag](#) and Action Tag must be linking directly to the action name (and optional namespace). This means that `<ww:form action="foo"/>` is OK, but `<ww:form action="foo.action"/>` is not.
- All code is expected to be using the Alt Syntax.

Setting up

SiteGraph is built in to WebWork, so if you're up and running with WebWork, you don't need to do anything additional java packages. However, SiteGraph does require the "dot" package by [GraphViz](#).

You'll need to download the latest version of GraphViz and make sure that the dot executable (dot.exe in Windows) is in your command path. In Windows the GraphViz installer typically automatically adds dot.exe to your path. However, you may need to do this by hand depending on your system configuration.

Usage

You can use SiteGraph with the following command:

```
java -cp ... -jar webwork.jar
sitegraph
-config CONFIG_DIR
-views VIEWS_DIRS
-output OUTPUT
[-ns NAMESPACE]
```

Where:

Content pulled from external source. Click [here](#) to refresh.

```
Usage: -config CONFIG_DIR -views VIEWS_DIRS -output OUTPUT [-ns NAMESPACE]
CONFIG_DIR => a directory containing xwork.xml
VIEWS_DIRS => comma seperated list of dirs containing JSPs, VMs, etc
OUPUT      => the directory where the output should go
NAMESPACE  => the namespace path restriction (/ , /foo, etc)
```



You must supply the correct classpath when invoking the SiteGraph tool. Specifically, the XWork, WebWork, and their dependencies must be included in the classpath. Futhermore, **you must**

also include your action class files referenced in xwork.xml. Without the proper class path entries, SiteGraph will not function properly.

Once you have run SiteGraph, check the directory specified in the "output" argument (OUTPUT). In there you will find two files: **out.dot** and **out.gif**. You may immediately open up **out.gif** and view the web application flow. However, you may also wish to either run the **out.dot** file through a different GraphVis layout engine (neato, twopi, etc), so the original dot file is provided as well. You may also wish to edit the dot file before rendering the final flow diagram.

Automatic Execution

Some advanced users may wish to execute SiteGraph from within their application – this could be required if you are developing an application that supports WebWork plugin capabilities. This can easily be done. See the JavaDocs for more info:

Content pulled from external source. Click [here](#) to refresh.

If you wish to use SiteGraph through its API rather than through the command line, you can do that as well. All you need to do is create a new SiteGraph instance, optionally specify a Writer to output the dot content to, and then call `#prepare()`.

The command line version of SiteGraph does exactly this (except for overriding the Writer):

Content pulled from external source. Click [here](#) to refresh.

```
SiteGraph siteGraph = new SiteGraph(configDir, views, output, namespace);
siteGraph.prepare();
siteGraph.render();
```


Result Types

This page last changed on Aug 30, 2005 by [plightbo](#).

See [Result Configuration](#) for basic information about how results are configuration.

Overview

Result Types are classes that determine what happens after an Action executes and a Result is returned. Developers are free to create their own Result Types according to the needs of their application or environment. In WebWork 2 for example, Servlet and Velocity Result Types have been created to handle rendering views in web applications.

Note: All built in webwork result types implement the `com.opensymphony.xwork.Result` interface, which represents a generic interface for all action execution results, whether that be displaying a webpage, generating an email, sending a JMS message, etc.

Result types define classes and map them to names to be referred in the action configuration results. This serves as a shorthand name-value pair for these classes.

snippet of webwork-default.xml

```
...
<result-types><result-type name="dispatcher"
class="com.opensymphony.webwork.dispatcher.ServletDispatcherResult"
default="true"/><result-type name="redirect"
class="com.opensymphony.webwork.dispatcher.ServletRedirectResult"/><result-type
name="velocity"
class="com.opensymphony.webwork.dispatcher.VelocityResult"/><result-type
name="chain" class="com.opensymphony.xwork.ActionChainResult"/><result-type
name="xslt" class="com.opensymphony.webwork.views.xslt.XSLTResult"/><result-type
name="jasper"
class="com.opensymphony.webwork.views.jasperreports.JasperReportsResult"/><result-type
name="freemarker"
class="com.opensymphony.webwork.views.freemarker.FreemarkerResult"/><result-type
name="httpheader"
class="com.opensymphony.webwork.dispatcher.HttpHeaderResult"/><result-type
name="stream"
class="com.opensymphony.webwork.dispatcher.StreamResult"/></result-types>
...
```

snippet of your xwork.xml

```
<include file="webwork-default.xml"/><package name="myPackage"
extends="default"><action name="bar" class="myPackage.barAction"><!-- default result
type is "dispatcher" --><!-- default result name is "success"
--><result>foo.jsp</result><result
name="error">error.jsp</result></result></action></package>
```

Result Types

Webwork provides several implementations of the `com.opensymphony.xwork.Result` interface to make web-based interactions with your actions simple. These result types include:

Result Type	name	class
Dispatcher Result	dispatcher	com.opensymphony.webwork.dispatcher
Redirect Result	redirect	com.opensymphony.webwork.dispatcher
Action Chaining Result	chain	com.opensymphony.xwork.ActionChain
Velocity Result	velocity	com.opensymphony.webwork.dispatcher
FreeMarker Result	freemarker	com.opensymphony.webwork.views.fr
JasperReports Result	jasper	com.opensymphony.webwork.views.ja
XSL Result	xslt	com.opensymphony.webwork.views.xs
HTTPHeader Result	header	com.opensymphony.webwork.dispatcher
Stream Result	stream	com.opensymphony.webwork.dispatcher

Results are specified in a xwork xml config file(xwork.xml) nested inside `<action>`. If the `location` param is the only param being specified in the result tag, you can simplify it as follows:

```
<action name="bar" class="myPackage.barAction"><result name="success"
type="dispatcher"><param name="location">foo.jsp</param></result></action>
```

or simplified

```
<action name="bar" class="myPackage.barAction"><result name="success"
type="dispatcher">foo.jsp</result></action>
```

if you are extending webwork-default.xml, then the default result type is "dispatcher". Also, if you don't specify the name of a result, it is assumed to be "success". This means you can simplify the result down to

```
<action name="bar" class="myPackage.barAction"><result>foo.jsp</result></action>
```

NOTE: The Parse attribute enables the location element to be parsed for expressions. An example of how this could be useful:

```
<result name="success" type="redirect">/displayCart.action?userId=${userId}</result>
```

NOTE: You can also specify global-results to use with multiple actions. This can save time from having to add the same result to many different actions. For more information on result tags and global-results, see [Result Configuration](#) section.

Action Chaining Result

This page last changed on Dec 13, 2004 by [casey](#).

Action Chaining

A special kind of view that invokes `GenericDispatch` (using the previously existing `ActionContext`) and executes another action. This is useful if you need to execute one action immediately after another.

Parameters	Required	Description
actionName	yes	the name of the action that will be chained to
namespace	no	sets the namespace of the Action that we're chaining to. If namespace is null, this defaults to the current namespace.

```
<result name="success" type="chain"><param name="actionName">bar</param><param name="namespace">/foo</param></result>
```

invokes this

```
<action name="bar" class="myPackage.barAction">
  ...
</action>
```

Dispatcher Result

This page last changed on Dec 13, 2004 by [casey](#).

Dispatcher

Includes or forwards to a view (usually a jsp). Behind the scenes WebWork will use a `RequestDispatcher`, where the target servlet/JSP receives the same request/response objects as the original servlet/JSP. Therefore, you can pass data between them using `request.setAttribute()` – the WebWork action is available.

Parameters	Required	Description
location	yes	the location to go to after execution (ex. jsp)
parse	no	true by default. If set to false, the location param will not be parsed for Ognl expressions

```
<result name="success" type="dispatcher"><param
name="location">foo.jsp</param></result>
```

FreeMarker Result

This page last changed on Aug 29, 2005 by [plightbo](#).

FreeMarker

Also see [WebWork Freemarker Support](#).

Parameters	Required	Description
location	yes	the location of the template to process
parse	no	true by default. If set to false, the location param will not be parsed for Ognl expressions
contentType	no	defaults to "text/html" unless specified

```
<result name="success" type="freemarker">foo.ftl</result>
```

Location of the Template

The FreemarkerManager class configures the template loaders so that the template location can be either

- relative to the web root folder. eg /WEB-INF/views/home.ftl
- a classpath resource. eg com/company/web/views/home.ftl

Freemarker Support in WebWork

Freemarker views can be rendered either via the webwork result type `freemarker`, or by using the `dispatcher` result type in conjunction Webwork's `FreemarkerServlet`.

This document will focus on using the `freemarker` result since it is the recommended approach. An section follows to show how to use the `FreemarkerServlet`.

Configure your action to use the `freemarker` result type

The `freemarker` result type is defined in `webwork-default.xml`, so normally you just include it, and define your results to use `type="freemarker"`.

```
<include file="webwork-default.xml"/>
...
<action name="test" class="package.Test"><result name="success"
type="freemarker">/WEB-INF/views/testView.ftl</result></action>
...
```

Property Resolution

Your action properties are automatically resolved - just like in a velocity view.

for example `${name}` will result in `stack.findValue("name")`, which *generally* results in `action.getName()` being executed.

A search process is used to resolve the variable, searching the following scopes in order, until a value is found :

- freemarker variables
- value stack
- request attributes

- session attributes
- servlet context attributes

Objects in the Context

The following variables exist in the freemarker views

- `req` - the current `HttpServletRequest`
- `res` - the current `HttpServletResponse`
- `stack` - the current `OgnlValueStack`
- `ognl` - the `OgnlTool` instance
 - This class contains useful methods to execute OGNL expressions against arbitrary objects, and a method to generate a select list using the `<ww:select>` pattern. (i.e. taking the name of the list property, a listKey and listValue)
- `webwork` - an instance of `FreemarkerWebWorkUtil`
- `action` - the current WebWork action
- `exception` - *optional* the Exception instance, if the view is a JSP exception or Servlet exception view

FreeMarker configuration with recent (post 2.1) releases

To configure the freemarker engine that webwork uses, just add a file `freemarker.properties` to the classpath. The supported properties are those that the freemarker Configuration object expects - see the freemarker documentation for these. These properties are used for both the `freemarker` result type, and the webwork provided `FreemarkerServlet`.

```
default_encoding=ISO-8859-1
template_update_delay=5
locale=no_NO
```

Using webwork UI tags - or any JSP Tag Library

Freemarker has builtin support for using any JSP taglib. You can use JSP taglibs in FreeMarker even if

a) your servlet container has no support for JSP, or

b) you didn't specify the taglib in your web.xml - note how in the example below we refer to the taglib by its webapp-absolute URL, so no configuration in web.xml is needed.

```
<#assign ww=JspTaglibs["/WEB-INF/webwork.tld"] />

<@ww.form method="'post'" name="'inputform'" action="'save.action'" >
  <@ww.hidden name="'id'" />
  <@ww.textarea label="'Details'" name="'details'" rows=5 cols=40 />
  <@ww.submit value="'Save'" align="center" />
</@ww.form>
```

NOTE : numeric properties for tags MUST be numbers, not strings. as in the rows and cols properties above. if you use cols="40" you will receive an exception. Other than that, the freemarker tag container behaves as you would expect.

Using the FreemarkerServlet

The FreemarkerServlet provided in the freemarker.jar will work out of the box **however** it won't provide any webwork specific functionality such as the context variables, property resolution etc. Therefore webwork provides its own servlet to provide this integration.

Register the FreemarkerServlet in web.xml

To use freemarker as a view engine, the webwork2 FreemarkerServlet needs to be configured, and mapped to the file extension that you use for your templates.

```
<servlet><servlet-name>freemarker</servlet-name><servlet-class>com.opensymphony.webwork.views.freemarker.FreemarkerServlet</servlet-class></servlet>
```

Configure Actions to use this servlet (xwork.xml configuration)

To use the freemarker view, just use the `dispatcher` result type, and specify the location to the template file.

```
<action name="test" class="package.Test"><result name="success"
type="dispatcher">/WEB-INF/views/testView.ftl</result></action>
```

Extending the servlet

NOTE: these docs need to be revised, since the FreemarkerServlet has changed since they were written.

Please refer to the freemarker site for details about the base freemarker servlet.

Be careful when subclassing `com.opensymphony.webwork.views.freemarker.FreemarkerServlet` when overriding

```
protected TemplateModel createModel(
    ObjectWrapper wrapper,
    ServletContext servletContext,
    HttpServletRequest request,
    HttpServletResponse response)
```

Please call `super.createModel(...)` and wrap it with a new model to avoid problems with action property resolution.

HTTPHeader Result

This page last changed on Jan 18, 2005 by [wosc](#).

HTTPHeader

A custom Result type for evaluating HTTP headers against the ValueStack.

Parameters	Required	Description
status	no	the http servlet response status code that should be set on a response
parse	no	true by default. If set to false, the headers param will not be parsed for Ognl expressions
headers	no	header values

Example:

```
<result name="success" type="httpheader"><param name="status">204</param><param name="headers.a">a custom header value</param><param name="headers.b">another custom header value</param></result>
```

JasperReports Result

This page last changed on Aug 31, 2005 by [zepernick](#).

JasperReports

Generates a JasperReports report using the specified format or PDF if no format is specified.

Parameters	Required	Description
location	yes	the location to go to after execution
parse	no	true by default. If set to false, the location param will not be parsed for Ognl expressions
dataSource	yes	the Ognl expression used to retrieve the datasource from the value stack (usually a List)
format	no	the format in which the report should be generated, defaults to pdf. Valid Formats: (HTML,PDF,XLS,CSV)
contentDisposition	no	defaults to "inline" when using documentName unless specified
documentName	no	generates the http header "Content-disposition = <contentDisposition>; filename=<documentName>.<format>

```
<result name="success" type="jasper"><param name="location">foo.jasper</param><param name="dataSource">mySource</param><param name="format">CSV</param></result>
```

or for pdf

```
<result name="success" type="jasper"><param name="location">foo.jasper</param><param name="dataSource">mySource</param></result>
```

Redirect Result

This page last changed on Dec 13, 2004 by [casey](#).

Redirect

The response is told to redirect the browser to the specified location (a new request from the client). The consequence of doing this means that the action (action instance, action errors, field errors, etc) that was just executed is lost and no longer available. This is because actions are built on a single-thread model. The only way to pass data is through the session or with web parameters (url?name=value) which can be OGNL expressions.

Parameters	Required	Description
location	yes	the location to go to after execution
parse	no	true by default. If set to false, the location param will not be parsed for Ognl expressions

```
<result name="success" type="redirect"><param name="location">foo.jsp</param><param name="parse">false</param></result>
```

Stream Result

This page last changed on Jul 25, 2005 by [bdittmer](#).

Stream

A custom Result type for send raw data (via an InputStream) directly to the HttpServletResponse. Very useful for allowing users to download content.

Parameters	Required	Default	Description
inputName	no	inputStream	The name of the attribute in the action that is the InputStream (such as <code>getInputStream()</code>)
contentType	no	text/plain	The content type of the stream being returned
contentDisposition	no	inline	The Content-disposition attribute in the response header. A typical value is <i>filename="doc.pdf"</i>
bufferSize	no	1024	the size, in bytes, of the buffer

Example:

```
<result name="success" type="stream"><param
name="inputName">inputStream</param><param
name="contentType">${contentType}</param><param
name="contentDisposition">attachment; filename="${filename}"</param><param
name="bufferSize">2024</param></result>
```

Velocity Result

This page last changed on Aug 29, 2005 by [plightbo](#).

Velocity

This result mocks a JSP execution environment and then displays a Velocity template that will be streamed directly to the servlet output. Also see [Resources Available to Velocity Views](#)

Parameters	Required	Description
location	yes	the location to go to after execution
parse	no	true by default. If set to false, the location param will not be parsed for Ognl expressions

```
<result name="success" type="velocity"><param  
name="location">foo.vm</param></result>
```


Summary

Here's the quick summary of the references available to Velocity templates that are kicked-off from WebWork:

- `$actionInstanceVariable`
- `$req` - the current `HttpServletRequest`
- `$res` - the current `HttpServletResponse`
- `$stack` - the current `OgnlValueStack`
- `$ognl` - an `OgnlTool`
- `$webwork` - an instance of `WebWorkUtil`
- `$action` - the current WebWork action
- `$taglib` (or something like that) - access to the JSP tag library via Velocity macros (!!)

Detail

`$actionInstanceVariable`

Each of your action class instance variables (for which you've written a getter method) are available in your Velocity template as `$actionInstanceVariableName`.

In other words, if you have an instance variable in your Action class:

```
public class ProcessEditTableRowAction extends ActionSupport {  
    privateString fooString;
```

And you have a getter method for that string (in that same ActionClass):

```
publicString getFooString() { return fooString; }
```

Then in your Velocity template you can retrieve the value of that String by simply referring to:

```
$fooString
```

Note: For the most part you don't have to worry about lettercase. If your Velocity reference is `$fooString` (note the lowercase "f" in foo) and your getter method is called `getFooString` (note uppercase "F" in foo) then WebWork does the "right thing" and uses `getFooString()` to retrieve the value of `$fooString`.

But things can get weird in some circumstances. *TODO:* At least I think they can get weird. At one point, I remember having a problem with the lettercase of Velocity variable names and the corresponding getter methods that was being used to lookup the value. But I can't remember exactly how the problem manifested so can't provide an example. Need to do that.

TODO: I'm curious what takes care of translating the `$variableName` references to method calls on the Action object's getters. Where does that happen?

\$req

The current `HttpServletRequest` created and managed by your Servlet environment (Tomcat, Resin, etc.).

\$res

The current `HttpServletResponse` created and managed by your Servlet environment (Tomcat, Resin, etc.).

\$stack

The OGNL value stack. ([API Docs](#))

TODO: Talk about what's actually **on** the stack. Doesn't do a whole lot of good to know it's there without knowing what's on it.

\$ognl

A reference to an OGNL tool. ([API Docs](#))

([\[See OGNL Basics\]](#)

<http://wiki.opensymphony.com/display/XW/Ognl>])

At the time of this writing, there are no docs for that object, so if you really want to

know how it works, be a [Jedi](#) and "use the source". (The CVS repository is browsable here: <https://webwork.dev.java.net/source/browse/webwork/src/java>)

Tip:

Jason/Patrick note that one nifty thing you can do with this tool is to call static class methods. That's rather handy since Velocity doesn't offer access to class variables or methods, but only to instantiate objects that have been placed in the Velocity Context. So you can't normally do things like `Math.random()`.

To call a class method, from within Velocity template, using the OGNL tool, you do this:

```
$ognl.findValue("@com.acme.FooClass@FOO")
```

TODO: the original email detailing the above example used `$value` to refer to the OGNL tool, not `$ognl`. But the source indicates it's "`$ognl`" that's shared with the context. Which is right?

\$webwork

A reference to the `VelocityWebWorkUtil` class. ([API Docs](#))

At the time of this writing, the API docs for this object are effectively blank, so if you really want to know how it works, be a [Jedi](#) and "use the source". (The CVS repository is browsable here: <https://webwork.dev.java.net/source/browse/webwork/src/java>)

Tip:

Mathew notes that the `VelocityWebWorkUtil` class can instantiate other objects for you. This, too, is very handy since Velocity is normally constrained to only the objects that have been explicitly shared with it's context (and not something you have control over when using Velocity templates kicked off by WebWork).

To instantiate an object from a Velocity page (that's been kicked off by WebWork, of course) do this:

```
#set($object = $webwork.bean("com.foo.ClassName"))
```

This technique can be particularly handy for grabbing a reference to one of the [VelocityTools](#) objects. (The VelocityTools are not included with WebWork, you'll need to go grab the lib and put it in your classpath if you want to use `$webwork.bean()` to grab references to the tool objects.)

(Note: If you're rumaging through the source, the source-code for the `$webwork.bean()` method above is really in VelocityWebWorkUtil's parent class WebWorkUtil.)

\$action

A reference to the action context that called this template. *TODO*: Not really sure what's in there or what it's good for. Should probably note that.

#tag and **#bodytag**

Provides a way of using a JSP taglib from velocity.
Many, but not all taglibs can be used.

ex:

```
#tag( Text "name='title.edit'" )
```

provides a way to uses the `<ww:text name="title.edit" />` tag from velocity.

Note the `<ww:text />` is probably not that useful from velocity since

```
$action.getText('title.edit')
```

would perform the same thing.

see [UI tags#WW/UI+Tags](#) for more information about webworks tags. Many provide equivalent velocity examples. Usage with other tags could be inferred based on examples given.

Introduction

XSLTResult uses XSLT to transform action object to XML. Recent version has been specifically modified to deal with Xalan flaws. When using Xalan you may notice that even though you have very minimal stylesheet like this one

```
<xsl:template match="/result"><result /></xsl:template>
```

then Xalan would still iterate through every property of your action and it's all descendants.

If you had double-linked objects then Xalan would work forever analysing infinite object tree. Even if your stylesheet was not constructed to process them all. It's because current Xalan eagerly and extensively converts everything to it's internal DTM model before further processing.

That's why there's a loop eliminator added that works by indexing every object-property combination during processing. If it notices that some object's property were already walked through, it doesn't get any deeper. Say, you have two objects x and y with the following properties set (pseudocode):

```
x.y = y;  
and  
y.x = x;  
action.x=x;
```

Due to that modification the resulting XML document based on x would be:

```
<result><x><y/></x></result>
```

Without it there would be an endless x/y/x/y/x/y/... elements.

The XSLTResult code tries also to deal with the fact that DTM model is built in a manner that childs are processed before siblings. The result is that if there is object **x**

that is both set in action's **x** property, and very deeply under action's **a** property then it would only appear under a, not under x. That's not what we expect, and that's why XSLTResult allows objects to repeat in various places to some extent.

Sometimes the object mesh is still very dense and you may notice that even though you have relatively simple stylesheet execution takes a tremendous amount of time. To help you to deal with that obstacle of Xalan you may attach regexp filters to elements paths (xpath).

For example:

```
<result name="success" type="xslt"><param name="location">foo.xslt</param><param name="matchingPattern">^/result/[^/*]${</param><param name="excludingPattern">.*(hugeCollection).*</param></result>
```

In the above code the XSLT result would only walk through action's properties without their childs. It would also skip every property that has "hugeCollection" in their name. Element's path is first compared to excludingPattern - if it matches it's no longer processed. Then it is compared to matchingPattern and processed only if there's a match.

Usage

Parameters	Required	Description
location	yes	the location to go to after execution
parse	no	Defaults to false. If set to true, the location will be parsed for Ognl expressions
matchingPattern	no	Pattern that matches only desired elements, by default it matches everything
excludingPattern	no	Pattern that eliminates unwanted elements, by

		default it matches none
--	--	-------------------------

webwork.properties related configuration

Property	Description
webwork.xslt.nocache	Defaults to false. If set to true, disables stylesheet caching. Good for development, bad for production

```
<result name="success" type="xslt">foo.xslt</result>
```

Webwork provides a tag library decoupled from the view technology. In this section, we describe each tag in general terms, such as the attributes it supports, what the behaviors are, etc. Most tags are supported in all template languages (see [JSP Tags](#), [Velocity Tags - Old](#), and [FreeMarker Tags](#)), but some are currently only specific to one language. Whenever a tag doesn't have complete support for every language, it will be noted in the reference documents.

The types of tags can be broken in to two types: general and HTML. Besides function and responsibility, the biggest difference between the general tags and the HTML tags is the fact that the HTML tags support *templates* and *themes*. In addition to the general tag reference, we also provide examples for using these generic tags in each of the support languages.



Please make sure you have read the [Tag Syntax](#) document and understand how tag attribute syntax works.

General Tags

General tags are used for controlling the execution flow when your pages render. They also allow for data extraction from places other than your action or the value stack, such as [Internationalization](#), JavaBeans, and including additional URLs or action executions.

1. [Control Tags](#) provide control flow, such as *if*, *else*, and *iterator*.
2. [Data Tags](#) allow for data manipulation or creation, such as *bean*, *push*, and *i18n*.
3. (**Note:** old content may be salvageable from [Common Tags](#))

HTML Tags

Unlike the general tags, the HTML tags do not provide much control structure or logic. Rather, they are focussed on using data, either from your action/value stack or from the [Data Tags](#), and displaying it in rich and reusable HTML. All HTML tags have a unique behavior that they are driven by *templates* and *themes*. While the general tags simply output some content directly from the tag (if there is any content to output), the HTML tags defer to a template, often grouped together as a theme, to do the actual rendering.

This unique template support allows for you to use the HTML tags to build a rich set of reusable UI components that fit your exact requirements. Please read the [Themes and Templates](#) guide for more information on this powerful feature.

1. [Themes and Templates](#): a must-read explanation of how themes and templates are uses when rendering HTML tags.
2. [Form Tags](#) provide all form-related HTML output, such as *form*, *textfield*, and *select*. (**Note**: old content may be salvageable from [UI Tags](#))
3. [Non Form Tags](#) provide all non-form-related HTML output, such as *a*, *div*, and *tabbedPanel*. (**Note**: old content may be salvageable from [Non-UI Tags](#))

Language Specific Tag Support

WebWork strives to support whatever environment you are most comfortable working in. That is why WebWork does not require a single template language, but instead allows for almost any common language to be used and even provides hooks for new languages. By default, almost every single tag is supported in JSP, Velocity, and FreeMarker. In each of these sections, you'll find examples and techniques for applying the generic tag reference toward your specific language or template choice.

1. [JSP Tags](#)
2. [Velocity Tags](#) / [Velocity Tags - Old](#)
3. [FreeMarker Tags](#)



As of WebWork 2.2, FreeMarker has become the "standard" template language recommended by the WebWork team. There are many reasons for this

decision, which can be found in various forums archives, but it pretty much boils down to this: FreeMarker provides a richer set of features than Velocity and is also more developer-friendly when errors occur (ie: the error reports are more accurate). JSP, while still used, is much more difficult for applications that demand a more modular approach, such as changing the templates at runtime or uploading packaged "modules" of WebWork actions and template files.

Common Tags

This page last changed on Nov 29, 2004 by [plightbo](#).

Param

Sets a parameter for the parent tag. Examples include `ww:url` and `ww:action`.

```
<ww:action name="VelocityCounter" id="vc">
  <ww:param name="foo" value="'BAR'"/>
</ww:action>
javascript:popUp(' <ww:url value="wiki.opensymphony.com/exec/edit"><ww:param
name="name" value="Common Tags"/></ww:url>')
```

from `webwork.tld`:

```
<attribute>
  <name>name</name>
  <required>true</required>
  <rtexprvalue>true</rtexprvalue>
</attribute>
<attribute>
  <name>value</name>
  <required>false</required>
  <rtexprvalue>true</rtexprvalue>
</attribute>
```

Property

Used to get the value of a result attribute. If the value isn't given, the top of the stack will be returned.

```
<ww:property value="id" default="#session[OS:'customer'].id"/>
```

From `webwork.tld`:

```
<attribute>
  <name>value</name>
  <required>false</required>
  <rtexprvalue>true</rtexprvalue>
</attribute>
<attribute>
  <name>default</name>
  <required>false</required>
```

```
<rtexprvalue>true</rtexprvalue>
</attribute>
```

Push

Using `ww:push`, you can add an object of your choice to the top of the value stack. This is similar to what you can do with `ww:set` (see below), so read both before deciding which to use.

```
<ww:push value="counter">
  <ww:property value="count"/>
</ww:push>

To make an action available on the stack:

<ww:action name="SomeAction" id="sa"/>
<ww:push value="#sa">
  foo = <ww:property value="foo"/>
</ww:push>
```

from `webwork.tld`:

```
<attribute>
  <name>value</name>
  <required>true</required>
  <rtexprvalue>true</rtexprvalue>
</attribute>
```

Set

You can create your own named variables from within a JSP using the `ww:set` tag. Reference your variable later using the `#variableName` notation.

```
<ww:set name="huba" value="foo.bar" scope="webwork" />

<ww:property value="#huba.otherExpression().baz"/>
```

from `webwork.tld`:

```
<info>
  Sets the value of an object in the VS to a scope
  (page, stack, application, session). If the value
```

```

    is not given, the top of the stack is used. If the
    scope is not given, the default scope of "webwork"
    is used.
</info>
<attribute>
  <name>name</name>
  <required>true</required>
  <rtexprvalue>true</rtexprvalue>
</attribute>
<attribute>
  <name>value</name>
  <required>false</required>
  <rtexprvalue>true</rtexprvalue>
</attribute>
<attribute>
  <name>scope</name>
  <required>false</required>
  <rtexprvalue>true</rtexprvalue>
</attribute>

```

Url

This tag builds an encoded Url. The simplest version of the tag, `<ww:url/>`, outputs the relative url of the current page. Here is example output from such a tag:

`/jsp/cart.jsp?template=%5BLjava.lang.String%3B%40e29f36&id=%5BLjava.lang.String%3B%40`

Here is a more verbose example:

```

<ww:url value="www.google.com/search">
  <ww:param name="sourceid" value="'navclient'"/>
  <ww:param name="ie" value="'UTF-8'"/>
  <ww:param name="oe" value="'UTF-8'"/>
  <ww:param name="q" value="'webwork'"/>
</ww:url>

```

and the resulting output:

<http://www.google.com/search?sourceid=navclient&ie=UTF-8&oe=UTF-8&q=webwork>

from `webwork.tld`:

```

<attribute>
  <name>value</name>
  <required>false</required>
  <rtexprvalue>false</rtexprvalue>
</attribute>
<attribute>
  <name>id</name>
  <required>false</required>
  <rtexprvalue>false</rtexprvalue>

```

```
</attribute>
```

Control Tags

This page last changed on Oct 23, 2005 by [digi9ten](#).

Controls tags provide the ability to manipulate collections and conditionally produce content.

1. [else](#)
2. [elseif / elseif](#)
3. [if](#)

Iteration Tags

1. [append](#)
2. [generator](#)
3. [iterator](#)
4. [merge](#)
5. [sort](#)
6. [subset](#)

Iterator will iterate over a value. An iterable value can be either of: `java.util.Collection`, `java.util.Iterator`, `java.util.Enumeration`, `java.util.Map`, `array`.

Example:

```
<ww:iterator value="days">
  <p>day is: <ww:property/></p>
</ww:iterator>
```

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

The following example uses a `BeanTag` and places it into the `ActionContext`. The `iterator` tag will retrieve that object from the `ActionContext` and then calls its `getDays()` method as above. The `status` attribute is also used to create a `IteratorStatus` object, which in this example, its `odd()` method is used to alternate row colours:

```

<ww:bean name="com.opensymphony.webwork.example.IteratorExample" id="it">
  <ww:param name="day" value="'foo'"/>
  <ww:param name="day" value="'bar'"/>
</ww:bean>

<table border="0" cellspacing="0" cellpadding="1">
<tr>
  <th>Days of the week</th>
</tr>

<ww:iterator value="#it.days" status="rowstatus">
  <tr>
    <ww:if test="#rowstatus.odd == true">
      <td style="background: grey"><ww:property/></td>
    </ww:if>
    <ww:else>
      <td><ww:property/></td>
    </ww:else>
  </tr>
</ww:iterator>
</table>

```

The next example will further demonstrate the use of the status attribute, using a DAO obtained from the action class through OGNL, iterating over groups and their users (in a security context). The last() method indicates if the current object is the last available in the iteration, and if not, we need to separate the users using a comma:

```

<webwork:iterator value="groupDao.groups" status="groupStatus">
  <tr class="<webwork:if test="#groupStatus.odd ==
true">odd</webwork:if><webwork:else>even</webwork:else>">
    <td><webwork:property value="name" /></td>
    <td><webwork:property value="description" /></td>
    <td>
      <webwork:iterator value="users" status="userStatus">
        <webwork:property value="fullName" /><webwork:if
test="!#userStatus.last">,</webwork:if>
      </webwork:iterator>
    </td>
  </tr>
</webwork:iterator>

```

The next example iterates over a an action collection and passes every iterator value to another action.

```

<ww:action name="entries" id="entries"/>
<ww:iterator value="#entries.entries" >
  <ww:property value="name" />
  <ww:property />
  <ww:push value="..." />

  <ww:action name="edit" id="edit" >

```



```
<ww:param name="entry" value="[0]" />
</ww:action>
</ww:iterator>
```

The trick here lies in the use of the '[':0]' operator. It takes the current iterator value and passes it on to the edit action. Using the '[':0]' operator has the same effect as using `<ww:property />`. (The latter, however, does not work from inside the param tag).

See also OS:WebWork2 EL.

append

This page last changed on Oct 03, 2005 by [digi9ten](#).

append

Attribute	Type	Required	Default	Description
id	string			
name	string	TRUE		
namespace	string		last part (/foo/bar/baz.xyz -> /foo/bar)	
executeResult	boolean			
ignoreContextPath	boolean		FALSE	

else

This page last changed on Oct 03, 2005 by [digi9ten](#).

else

Attribute	Type	Required	Default	Description
id	string			

elseif

This page last changed on Oct 03, 2005 by [digi9ten](#).

elseif/elseif

Attribute	Type	Required	Default	Description
id	string			
test	string			

generator

This page last changed on Oct 05, 2005 by [digi9ten](#).

generator

Attribute	Type	Required	Default	Description
id	string	FALSE		
val	string	FALSE		
separator	string	FALSE		
count		FALSE		

if

This page last changed on Oct 05, 2005 by [digi9ten](#).

if

Attribute	Type	Required	Default	Description
id	string	FALSE		
test	string	TRUE		

iterator

This page last changed on Oct 05, 2005 by [digi9ten](#).

iterator

Attribute	Type	Required	Default	Description
id	string	FALSE		
status	string	FALSE		
value	string	FALSE		
name	string	TRUE		
namespace	string	FALSE	last part (/foo/bar/baz.xyz -> /foo/bar)	
executeResult	boolean	FALSE		
ignoreContextParameters	boolean	FALSE	FALSE	

merge

This page last changed on Oct 05, 2005 by [digi9ten](#).

merge

Attribute	Type	Required	Default	Description
id	string	FALSE		
name	string	TRUE		
namespace	string	FALSE	last part (/foo/bar/baz.xyz -> /foo/bar)	
executeResult	boolean	FALSE		
ignoreContextPath	boolean	FALSE	FALSE	

sort

This page last changed on Oct 05, 2005 by [digi9ten](#).

sort

Attribute	Type	Required	Default	Description
id	string	FALSE		
source	string	FALSE		
comparator	string	FALSE		
name	string	TRUE		
namespace	string	FALSE	last part (/foo/bar/baz.xyz -> /foo/bar)	
executeResult	boolean	FALSE		
ignoreContextParameters	boolean	FALSE	FALSE	

subset

This page last changed on Oct 05, 2005 by [digi9ten](#).

subset

Attribute	Type	Required	Default	Description
id	string	FALSE		
source	string	FALSE		
count	string	FALSE		
start	string	FALSE		
name	string	TRUE		
namespace	string	FALSE	last part (/foo/bar/baz.xyz -> /foo/bar)	
executeResult	boolean	FALSE		
ignoreContextParameters	boolean	FALSE	FALSE	

Data Tags

This page last changed on Oct 23, 2005 by [digi9ten](#).

Data tags provide various data-related functionality. This ranges from displaying the direct result of an action, to retrieving localized values.

1. [action](#)
2. [bean](#)
3. [debug](#)
4. [i18n](#)
5. [include](#)
6. [param](#)
7. [push](#)
8. [set](#)
9. [text](#)
10. [url](#)

action

This page last changed on Oct 11, 2005 by [digi9ten](#).

action

Attribute	Type	Required	Default	Description
id	string	FALSE		
name	string	TRUE		
namespace	string	FALSE	last part (/foo/bar/baz.xyz -> /foo/bar)	
executeResult	boolean	FALSE	FALSE	
ignoreContextParameters	boolean	FALSE	FALSE	

bean

This page last changed on Oct 11, 2005 by [digi9ten](#).

bean

Attribute	Type	Required	Default	Description
id	string	FALSE		
name	string	TRUE		

debug

This page last changed on Oct 11, 2005 by [digi9ten](#).

debug

Attribute	Type	Required	Default	Description
id	string	FALSE		

i18n

This page last changed on Oct 05, 2005 by [digi9ten](#).

i18n

Attribute	Type	Required	Default	Description
name	string	TRUE		

include

This page last changed on Oct 11, 2005 by [digi9ten](#).

include

Attribute	Type	Required	Default	Description
value	string	TRUE		URL to include

param

This page last changed on Oct 11, 2005 by [digi9ten](#).

param

Attribute	Type	Required	Default	Description
name	string	TRUE		
value	string	FALSE		

push

This page last changed on Oct 12, 2005 by [mariuszs](#).

push

Attribute	Type	Required	Default	Description
value	string	TRUE		

Examples

Example 1

```
<ww:push value="user"><ww:property value="firstName" /><ww:property value="lastName" /></ww:push>
```

instead of:

```
<ww:property value="user.firstName" /><ww:property value="user.lastName" />
```

set

This page last changed on Oct 11, 2005 by [digi9ten](#).

set

Attribute	Type	Required	Default	Description
name	string	TRUE		
value	string	FALSE		
scope	string	FALSE		

text

This page last changed on Oct 11, 2005 by [digi9ten](#).

text

Attribute	Type	Required	Default	Description
name	string	TRUE		Represents the i18n key.

url

This page last changed on Oct 11, 2005 by [digi9ten](#).

url

Attribute	Type	Required	Default	Description
id	string	FALSE		
value	string	FALSE	'webwork.request_uri' of current request	
scheme	string	FALSE		
encode	boolean	FALSE	TRUE	
includeParams	string	FALSE	get	"Possible values are 'none', 'get' or 'all'"
includeContext	boolean	FALSE	TRUE	

Form Tags

This page last changed on Oct 23, 2005 by [digi9ten](#).

Within the form tags, there are two classes of tags: the form tag itself, and all other tags, which make up the individual form elements. This is important as the behavior of the form tag itself is different than that of the elements enclosed within it. Before we go provide a reference for all the form tags, including the form tag itself, we must outline some general characteristics first.

Form Tag Themes

As previously noted in [Themes and Templates](#), the HTML Tags (which includes Form Tags) are all driven by templates. Templates are grouped together to form themes. By default, WebWork provides three themes:

- simple
- xhtml, which extends simple (default)
- ajax, which extends xhtml

Remember: the xhtml theme renders out a two-column table. If you need a different layout, we highly recommend that you do *not* write your own HTML, but rather create your own theme or utilize the simple theme.

The downside of using the simple theme is that it doesn't support as many of the attributes that the other themes do. For example, the label attribute does nothing in the simple theme. Similarly, the functionality offered by the simple theme is much less than that of the xhtml and ajax themes: the automatic display of error messages is not supported.

Common Attributes

All the form tags extend the UIBean class. This base class generally common attributes, grouped in to three classes: templated-related, javascript-related, and general attributes. We won't document what these attributes do here as that is taken

care of in each individual tag's reference. However, it is a good idea to familiarize yourself with the structure of the UI tags and what attributes are available for all tags.

In addition to these attributes, a special attribute exists for all form element tags: *form* (ie: `${parameters.form}`). This represents the parameters used to render the form tag and allows you to provide interaction between your form elements and the form itself. For example, in a template you could access the form's ID by calling `${parameters.form.id}`.

Template-Related Attributes

TODO: need to include code from UIBean.java

Javascript-Related Attributes

TODO: need to include code from UIBean.java

General Attributes

TODO: need to include code from UIBean.java

When Some Attributes Don't Apply

Note that some tags don't have any templates that utilize certain attributes, either because it doesn't make sense or it isn't required. For example, the form tag, while it supports the *tabindex* attribute, none of the themes render it out. Also, as mentioned, certain themes won't utilize some attributes.

Value/Name Relationship

In many of the tags, except for the form tag, there is a unique relationship between

the *name* and *value* attributes. The *name* attribute is what the form element gets named and eventually submitted as. This effectively is the expression to which you wish to bind the incoming value to. In most cases, it is a simple JavaBean property, such as "firstName". This would eventually call `setFirstName()`.

Similarly, you often wish to also display in your form elements existing data from the same JavaBean property. This time, the attribute *value* is used. A value of `"%{firstName}"` would call `getFirstName()` and display it in your form, allowing users to edit the value and re-submit it.

You could use the following code, and it would work just fine:

```
<@ww.form action="updatePerson"><@ww.textfield label="First name" name="firstName"
value="%{firstName}" />
...
</@ww.form>
```

However, because the relationship between *name* and *value* is so often predictable, we automatically do this for you, allowing you to do:

```
<@ww.form action="updatePerson"><@ww.textfield label="First name" name="firstName" />
...
</@ww.form>
```

While most attributes are exposed to the underlying templates as the same key as the attribute (ie: `${parameters.label}`), the *value* attribute is not. Instead, it can be accessed via the "nameValue" key (ie: `${parameters.nameValue}`) to indicate that it may have been generated from the *name* attribute rather than explicitly defined in the *value* attribute.

ID Name Assignment

All form tags automatically assign an ID for you. You are free to override this ID if you wish. The ID assignment works as follows:

1. For forms, the ID is assumed to the action name. In the previous example, the ID would be "updatePerson".

2. For form elements, the ID is assumed to be form's ID_element name

Required Attribute

The "required" attribute on many WebWork UI tags defaults to true only if you have client side validation enabled and there is a validator associated with that particular field.

Form Tag Reference

1. [checkbox](#) - renders a checkbox input field
2. [checkboxlist](#) - renders a list of checkboxes
3. [combobox](#) - renders a widget that fills a text box from a select
4. [datepicker](#) - renders
5. [doubleselect](#) - renders
6. [file](#) - renders
7. [form](#) - renders an input form
8. [hidden](#) - renders a hidden form field
9. [label](#) - renders renders a label
10. [password](#) - renders a password textfield
11. [radio](#) - renders a radio button
12. [select](#) - renders a select
13. [submit](#) - renders a submit button
14. [textarea](#) - renders a textarea
15. [textfield](#) - renders a textfield
16. [token](#) - renders a hidden field to stop double-submission of containing forms



It's very important to note that all tags that insert something into the valuestack (like `i18n` or bean tags) will remove those objects from the stack on its end tag. This means that if you instantiate a bean with the bean tag (`<ww:bean name=""br.univap.fcc.sgpw.util.FormatterHelper"">`) that bean will be available on the valuestack only until the `</ww:bean>` tag.

checkbox

This page last changed on Oct 02, 2005 by [digi9ten](#).

checkbox

Attribute	Type	Required	Default	Description
id	string	FALSE		
fieldValue	string	FALSE		
maxLength	integer	FALSE		
readonly	boolean	FALSE		
size	integer	FALSE		
name	string	TRUE		
value	string	FALSE		
required	boolean	FALSE		
disabled	boolean	FALSE		
theme	string	FALSE		
template	string	FALSE		
cssClass	string	FALSE		
cssStyle	string	FALSE		
label	string	FALSE		
labelposition	string	FALSE		
tabindex	string	FALSE		
onclick	string	FALSE		
ondblclick	string	FALSE		
onmousedown	string	FALSE		
onmouseup	string	FALSE		
onmouseover	string	FALSE		
onmousemove	string	FALSE		
onmouseout	string	FALSE		

onfocus	string	FALSE		
onblur	string	FALSE		
onkeypress	string	FALSE		
onkeydown	string	FALSE		
onselect	string	FALSE		
onchange	string	FALSE		

checkboxlist

This page last changed on Oct 02, 2005 by [digi9ten](#).

checkboxlist

Attribute	Type	Required	Default	Description
id	string	FALSE		
list	collection	TRUE		
listKey	string	FALSE		
listValue	string	FALSE		
name	string	TRUE		
value	string	FALSE		
required	boolean	FALSE		
disabled	boolean	FALSE		
theme	string	FALSE		
template	string	FALSE		
cssClass	string	FALSE		
cssStyle	string	FALSE		
label	string	FALSE		
labelposition	string	FALSE		
tabindex	string	FALSE		
onclick	string	FALSE		
ondblclick	string	FALSE		
onmousedown	string	FALSE		
onmouseup	string	FALSE		
onmouseover	string	FALSE		
onmousemove	string	FALSE		
onmouseout	string	FALSE		
onfocus	string	FALSE		

onblur	string	FALSE		
onkeypress	string	FALSE		
onkeydown	string	FALSE		
onselect	string	FALSE		
onchange	string	FALSE		

combobox

This page last changed on Oct 02, 2005 by [digi9ten](#).

combobox

Attribute	Type	Required	Default	Description
id	string	FALSE		
list	collection	TRUE		
maxLength	integer	FALSE		
readonly	boolean	FALSE		
size	integer	FALSE		
name	string	TRUE		
value	string	FALSE		
required	boolean	FALSE		
disabled	boolean	FALSE		
theme	string	FALSE		
template	string	FALSE		
cssClass	string	FALSE		
cssStyle	string	FALSE		
label	string	FALSE		
labelposition	string	FALSE		
tabindex	string	FALSE		
onclick	string	FALSE		
ondblclick	string	FALSE		
onmousedown	string	FALSE		
onmouseup	string	FALSE		
onmouseover	string	FALSE		
onmousemove	string	FALSE		
onmouseout	string	FALSE		

onfocus	string	FALSE		
onblur	string	FALSE		
onkeypress	string	FALSE		
onkeydown	string	FALSE		
onselect	string	FALSE		
onchange	string	FALSE		

datepicker

This page last changed on Oct 02, 2005 by [digi9ten](#).

datepicker

Attribute	Type	Required	Default	Description
id	string	FALSE		
maxLength	integer	FALSE		
readonly	boolean	FALSE		
size	integer	FALSE		
name	string	TRUE		
value	string	FALSE		
required	boolean	FALSE		
disabled	boolean	FALSE		
theme	string	FALSE		
template	string	FALSE		
cssClass	string	FALSE		
cssStyle	string	FALSE		
label	string	FALSE		
labelposition	string	FALSE		
tabindex	string	FALSE		
onclick	string	FALSE		
ondblclick	string	FALSE		
onmousedown	string	FALSE		
onmouseup	string	FALSE		
onmouseover	string	FALSE		
onmousemove	string	FALSE		
onmouseout	string	FALSE		
onfocus	string	FALSE		

onblur	string	FALSE		
onkeypress	string	FALSE		
onkeydown	string	FALSE		
onselect	string	FALSE		
onchange	string	FALSE		

doubleselect

This page last changed on Oct 02, 2005 by [digi9ten](#).

doubleselect

Attribute	Type	Required	Default	Description
id	string	FALSE		
list	collection	TRUE		
listKey	string	FALSE		
listValue	string	FALSE		
doubleList	collection	FALSE		
doubleListKey	string	FALSE		
doubleListValue	string	FALSE		
doubleName	string	FALSE		
doubleValue	string	FALSE		
formName	string	FALSE		
emptyOption	boolean	FALSE		
multiple	boolean	FALSE		
headerKey	string	FALSE		
headerValue	string	FALSE		
size	integer	FALSE		
name	string	TRUE		
value	string	FALSE		
required	boolean	FALSE		
disabled	boolean	FALSE		
theme	string	FALSE		
template	string	FALSE		
cssClass	string	FALSE		
cssStyle	string	FALSE		

label	string	FALSE		
labelposition	string	FALSE		
tabindex	string	FALSE		
onclick	string	FALSE		
ondblclick	string	FALSE		
onmousedown	string	FALSE		
onmouseup	string	FALSE		
onmouseover	string	FALSE		
onmousemove	string	FALSE		
onmouseout	string	FALSE		
onfocus	string	FALSE		
onblur	string	FALSE		
onkeypress	string	FALSE		
onkeydown	string	FALSE		
onselect	string	FALSE		
onchange	string	FALSE		

file

This page last changed on Oct 02, 2005 by [digi9ten](#).

file

Attribute	Type	Required	Default	Description
id	string	FALSE		
accept	string	FALSE		
size	integer	FALSE		
name	string	TRUE		
value	string	FALSE		
required	boolean	FALSE		
disabled	boolean	FALSE		
theme	string	FALSE		
template	string	FALSE		
cssClass	string	FALSE		
cssStyle	string	FALSE		
label	string	FALSE		
labelposition	string	FALSE		
tabindex	string	FALSE		
onclick	string	FALSE		
ondblclick	string	FALSE		
onmousedown	string	FALSE		
onmouseup	string	FALSE		
onmouseover	string	FALSE		
onmousemove	string	FALSE		
onmouseout	string	FALSE		
onfocus	string	FALSE		
onblur	string	FALSE		

onkeypress	string	FALSE		
onkeydown	string	FALSE		
onselect	string	FALSE		
onchange	string	FALSE		

form

This page last changed on Oct 16, 2005 by [roughley](#).

Talk generally about the form, behaviors, etc. Mention theme-specific notes as well.

Attribute	Type	Required	Default	Theme	Description
action	String	Yes	N/A	simple	...
namespace	String	No	The current namespace	simple	...
validate	boolean	No	false	xhtml	...

form

Attribute	Type	Required	Default	Description
id	string	FALSE	action attribute	
name	string	FALSE	action attribute	
action	string	FALSE		
target	string	FALSE		
enctype	string	FALSE		
method	string	FALSE		
namespace	string	FALSE	global namespace: ""	
onsubmit	string	FALSE		
validate	boolean	FALSE		
name	string	TRUE		
value	string	FALSE		
required	boolean	FALSE		
disabled	boolean	FALSE		
theme	string	FALSE		
template	string	FALSE		
cssClass	string	FALSE		

cssStyle	string	FALSE		
label	string	FALSE		
labelposition	string	FALSE		
tabindex	string	FALSE		
onclick	string	FALSE		
ondblclick	string	FALSE		
onmousedown	string	FALSE		
onmouseup	string	FALSE		
onmouseover	string	FALSE		
onmousemove	string	FALSE		
onmouseout	string	FALSE		
onfocus	string	FALSE		
onblur	string	FALSE		
onkeypress	string	FALSE		
onkeydown	string	FALSE		
onselect	string	FALSE		
onchange	string	FALSE		

The remote form allows the form to be submitted without the page being refreshed. The results from the form can be inserted into any HTML element on the page.

Validation

There are two flavours of validation

1. Normal request/response process - please see [Validation](#).
1. AJAX-based validation, which is performed when the user moves between fields - please see [Remote Form Validation](#)

Asynchronous Form Processing (AJAX)

To ajax enable the form, the form tag must be used specifying a theme="ajax". Additionally, the [submit](#) tag must be used to provide the button that will submit the form.

Remote Form Validation

This page last changed on Oct 16, 2005 by [roughley](#).

Asynchronous Client side form validation in WW2.2 is super easy, and super cool.

Here is what I learned while getting it running.

- 1) Have [WW Validation](#) working already. Go for it, use something cool like a nested validator with a email validation rule.
- 2) You need the latest [DWR](#) jar in WEB-INF/lib.
- 3) Add this code to your web.xml

web.xml

```
<servlet>
  <servlet-name>dwr</servlet-name>
  <servlet-class>uk.ltd.getahead.dwr.DWRServlet</servlet-class>
</servlet>

<servlet-mapping>
  <servlet-name>dwr</servlet-name>
  <url-pattern>/dwr/*</url-pattern>
</servlet-mapping>
```

- 4) Update an existing form. Depending on how extensively you used WW form controls this step might take a few moments. The controls all need to be generated via WW tags. No being lazy and just using <form name="xxx". Ill explain why this is in a moment. You will need to follow these rules to stay on the happy path:

- <ww:form must have an id element.
- Again use only ww: tags
- To turn client side validation on, set the validate="true" attribute
- Well I guess thats it.

Here is a sample form:

```
<ww:form id="input_form" action="'SaveUser.action'" method="post" validate="true">
  <ww:textfield label="'First Name'" name="'bo.firstName'" size="20"
required="true"/>
  <ww:textfield label="'Last Name'" name="'bo.lastName'" size="20"
required="true"/>
  <ww:textfield label="'Email'" name="'bo.emailAddress'" size="20"
required="true"/>
  <ww:submit name="'submit'" value="'Save'" cssClass="'primary'"/>
</ww:form>
```

See what WW generates for you. This is why its critical to use all the WW tags. There are two things happening here. In the ww:form tag ww includes all the JavaScript libs. And secondly, notice how the ids for all the child elements got filled in based on the name of the parent form? Thats the second half of the magic. WWs JavaScript uses these ids to getElementById.

Generated Code

```
<script src="/agility/webwork/validationClient.js"></script>
<script src="/agility/dwr/interface/validator.js"></script>
<script src="/agility/dwr/engine.js"></script>
<script src="/agility/webwork/template/xhtml/validation.js"></script>
<form namespace="" id="input_form" name="SubmitUser"
action="/agility/SubmitUser.action">

<p>
<label for="input_form_bo.firstName" class="label"><span
class="required">*</span>First Name:</label>
<input name="bo.firstName" size="20" value="admin" id="input_form_bo.firstName"
onblur="validate(this);" type="text">
</p>
<input type="submit" name="submit" value="Save" class="primary"/>
</form>
<p>
```

5) Thats about it. You should be off to the races using asynchronous client side validation!!!

hidden

This page last changed on Oct 02, 2005 by [digi9ten](#).

hidden

Attribute	Type	Required	Default	Description
id	string	FALSE		
name	string	TRUE		
value	string	FALSE		
required	boolean	FALSE		

label

This page last changed on Oct 02, 2005 by [digi9ten](#).

label

Attribute	Type	Required	Default	Description
id	string	FALSE		
for	string	FALSE		
name	string	TRUE		
value	string	FALSE		
required	boolean	FALSE		
disabled	boolean	FALSE		
theme	string	FALSE		
template	string	FALSE		
cssClass	string	FALSE		
cssStyle	string	FALSE		
label	string	FALSE		
labelposition	string	FALSE		
tabindex	string	FALSE		
onclick	string	FALSE		
ondblclick	string	FALSE		
onmousedown	string	FALSE		
onmouseup	string	FALSE		
onmouseover	string	FALSE		
onmousemove	string	FALSE		
onmouseout	string	FALSE		
onfocus	string	FALSE		
onblur	string	FALSE		
onkeypress	string	FALSE		

onkeydown	string	FALSE		
onselect	string	FALSE		
onchange	string	FALSE		

password

This page last changed on Oct 02, 2005 by [digi9ten](#).

password

Attribute	Type	Required	Default	Description
id	string	FALSE		
maxLength	integer	FALSE		
readonly	boolean	FALSE		
size	integer	FALSE		
showPassword	boolean	FALSE	FALSE	
name	string	TRUE		
value	string	FALSE		
required	boolean	FALSE		
disabled	boolean	FALSE		
theme	string	FALSE		
template	string	FALSE		
cssClass	string	FALSE		
cssStyle	string	FALSE		
label	string	FALSE		
labelposition	string	FALSE		
tabindex	string	FALSE		
onclick	string	FALSE		
ondblclick	string	FALSE		
onmousedown	string	FALSE		
onmouseup	string	FALSE		
onmouseover	string	FALSE		
onmousemove	string	FALSE		
onmouseout	string	FALSE		

onfocus	string	FALSE		
onblur	string	FALSE		
onkeypress	string	FALSE		
onkeydown	string	FALSE		
onselect	string	FALSE		
onchange	string	FALSE		

radio

This page last changed on Oct 02, 2005 by [digi9ten](#).

radio

Attribute	Type	Required	Default	Description
id	string	FALSE		
list	collection	TRUE		
listKey	string	FALSE		
listValue	string	FALSE		
name	string	TRUE		
value	string	FALSE		
required	boolean	FALSE		
disabled	boolean	FALSE		
theme	string	FALSE		
template	string	FALSE		
cssClass	string	FALSE		
cssStyle	string	FALSE		
label	string	FALSE		
labelposition	string	FALSE		
tabindex	string	FALSE		
onclick	string	FALSE		
ondblclick	string	FALSE		
onmousedown	string	FALSE		
onmouseup	string	FALSE		
onmouseover	string	FALSE		
onmousemove	string	FALSE		
onmouseout	string	FALSE		
onfocus	string	FALSE		

onblur	string	FALSE		
onkeypress	string	FALSE		
onkeydown	string	FALSE		
onselect	string	FALSE		
onchange	string	FALSE		

select

This page last changed on Oct 02, 2005 by [digi9ten](#).

select

Attribute	Type	Required	Default	Description
id	string	FALSE		
list	collection	TRUE		
listKey	string	FALSE		
listValue	string	FALSE		
emptyOption	boolean	FALSE		
multiple	boolean	FALSE		
headerKey	string	FALSE		
headerValue	string	FALSE		
size	integer	FALSE		
name	string	TRUE		
value	string	FALSE		
required	boolean	FALSE		
disabled	boolean	FALSE		
theme	string	FALSE		
template	string	FALSE		
cssClass	string	FALSE		
cssStyle	string	FALSE		
label	string	FALSE		
labelposition	string	FALSE		
tabindex	string	FALSE		
onclick	string	FALSE		
ondblclick	string	FALSE		
onmousedown	string	FALSE		

onmouseup	string	FALSE		
onmouseover	string	FALSE		
onmousemove	string	FALSE		
onmouseout	string	FALSE		
onfocus	string	FALSE		
onblur	string	FALSE		
onkeypress	string	FALSE		
onkeydown	string	FALSE		
onselect	string	FALSE		
onchange	string	FALSE		

submit

This page last changed on Oct 16, 2005 by [roughley](#).

The submit tag is used together with the [form](#) tag to provide asynchronous form submissions.



Be sure to setup the page containing this tag to be [Configured for AJAX](#)

Attribute	Type	Required	Default	Description
resultDivId	string	TRUE		The id of the HTML element to place the result (this can be the form's id or any id on the page)
notifyTopics	string	FALSE		Topic names to post an event to after the form has been submitted
onLoadJS	string	FALSE		Javascript code that will be executed after the form has been submitted. The format is onLoadJS='yourMethodName' NOTE: the words data and type must be left like that if you want the event type and the returned

				data.
--	--	--	--	-------

The remote form has three basic modes of use, using the resultDivId, the notifyTopics, or the onLoadJS. You can mix and match any combination of them to get your desired result. All of these examples are contained in the Ajax example webapp. Lets go through some scenarios to see how you might use it:

- Show the results in another div. If you want your results to be shown in a div, use the resultDivId where the id is the id of the div you want them shown in. This is an inner HTML approach. Your results get jammed into the div for you. Here is a sample of this approach:

```
Remote form replacing another div:
<div id='two' style="border: 1px solid yellow;">Initial content</div>
<ww:form
    id='theForm2'
    cssStyle="border: 1px solid green;"
    action='/AjaxRemoteForm.action'
    method='post'
    theme="ajax">

    <input type='text' name='data' value='WebWork User'>
    <ww:submit value="GO2" theme="ajax" resultDivId="two"/>

</ww:form>
```

- Notify other controls(divs) of a change. Using an pub-sub model you can notify others that your control changed and they can take the appropriate action. Most likely they will execute some action to refresh. The notifyTopics does this for you. You can have many topic names in a comma delimited list. IE: notifyTopics="newPerson, dataChanged" . Here is an example of this approach:

```
<ww:form id="frm1" action="newPersonWithXMLResult" theme="ajax" >
    <ww:textfield label="'Name'" name="'person.name'" value="person.name"
size="20" required="true" />
    <ww:submit id="submitBtn" value="Save" theme="ajax" cssClass="primary"
notifyTopics="personUpdated, systemWorking" />
</ww:form>

<ww:div href="/listPeople.action" theme="ajax" errorText="error opps"
loadingText="loading..." id="cart-body" >
    <ww:action namespace="" name="listPeople" executeResult="true" />
</ww:div>
```

- Massage the results with JavaScript. Say that your result returns some happy XML and you want to parse it and do lots of cool things with it. The way to do this is with a onLoadJS handler. Here you provide the name of a JavaScript function to be called back with the result and the event type. The only key is that you must use the variable names 'data' and 'type' when defining the callback. For example: onLoadJS="myFancyDancyFunction(data, type)". While I talked about XML in this

example, your not limited to XML, the data in the callback will be exactly whats returned as your result.

Here is an example of this approach:

```
<script language="JavaScript" type="text/javascript">
    function doGreatThings(data, type) {
        //Do whatever with your returned fragment...
    //Perhapps.... if xml...
    var xml = dojo.xml.domUtil.createDocumentFromText(data);
        var people = xml.getElementsByTagName("person");
        for(var i = 0;i < people.length; i++){
            var person = people[i];
            var name = person.getAttribute("name")
            var id = person.getAttribute("id")
            alert('Thanks dude. Person: ' + name + ' saved great!!!');
        }

    }
</script>

<ww:form id="frm1" action="newPersonWithXMLResult" theme="ajax" >
    <ww:textfield label="'Name'" name="'person.name'" value="person.name"
size="20" required="true" />
    <ww:submit id="submitBtn" value="Save" theme="ajax" cssClass="primary"
onLoadJS="doGreatThings(data, type)" />
</ww:form>
```

textarea

This page last changed on Oct 02, 2005 by [digi9ten](#).

textarea

Attribute	Type	Required	Default	Description
id	string	FALSE		
cols	integer	FALSE		
rows	integer	FALSE		
readonly	boolean	FALSE	FALSE	
wrap	boolean	FALSE	FALSE	
name	string	TRUE		
value	string	FALSE		
required	boolean	FALSE		
disabled	boolean	FALSE		
theme	string	FALSE		
template	string	FALSE		
cssClass	string	FALSE		
cssStyle	string	FALSE		
label	string	FALSE		
labelposition	string	FALSE		
tabindex	string	FALSE		
onclick	string	FALSE		
ondblclick	string	FALSE		
onmousedown	string	FALSE		
onmouseup	string	FALSE		
onmouseover	string	FALSE		
onmousemove	string	FALSE		
onmouseout	string	FALSE		

onfocus	string	FALSE		
onblur	string	FALSE		
onkeypress	string	FALSE		
onkeydown	string	FALSE		
onselect	string	FALSE		
onchange	string	FALSE		

textfield

This page last changed on Oct 02, 2005 by [digi9ten](#).

textfield

Attribute	Type	Required	Default	Description
id	string	FALSE		
maxLength	integer	FALSE		
readonly	boolean	FALSE	FALSE	
size	integer	FALSE		
name	string	TRUE		
value	string	FALSE		
required	boolean	FALSE		
disabled	boolean	FALSE		
theme	string	FALSE		
template	string	FALSE		
cssClass	string	FALSE		
cssStyle	string	FALSE		
label	string	FALSE		
labelposition	string	FALSE		
tabindex	string	FALSE		
onclick	string	FALSE		
ondblclick	string	FALSE		
onmousedown	string	FALSE		
onmouseup	string	FALSE		
onmouseover	string	FALSE		
onmousemove	string	FALSE		
onmouseout	string	FALSE		
onfocus	string	FALSE		

onblur	string	FALSE		
onkeypress	string	FALSE		
onkeydown	string	FALSE		
onselect	string	FALSE		
onchange	string	FALSE		

token

This page last changed on Oct 02, 2005 by [digi9ten](#).

token

Attribute	Type	Required	Default	Description
id	string	FALSE		
name	string	TRUE	webwork.token	
value	string	FALSE		

FreeMarker tags are extensions of the generic [Tags and UI Components](#) provided by WebWork. You can get started almost immediately by simply knowing the generic structure in which the tags can be accessed: `<@ww.xxx> ...</@ww.xxx>`, where xxx is any of the tags supported by WebWork.

Syntax

For example, in JSP you might create a form like so:

```
<ww:form action="updatePerson"><ww:textfield label="First name"
name="firstName"/><ww:submit value="Update"/></ww:form>
```

In FreeMarker the same form is built like so:

```
<@ww.form action="updatePerson"><@ww.textfield label="First name"
name="firstName"/><@ww.submit value="Update"/></@ww.form>
```

While this covers almost all you need to know for FreeMarker tags, there are a few other advanced features you should read about, specifically with how attributes and parameters work together, and how attribute types (String, List, etc) can affect the tag behavior.

Attributes and Parameters

Unlike older versions of JSP (in which the [JSP Tags](#) are based), FreeMarker allows for *dynamic attributes*, much like JSP 2.0. What this means is that you can supply attributes to the tags that the tag doesn't even support. Those attributes that cannot be applied directly to the tag object will instead be set on the tag's general **parameters** map.

For example, suppose you have the following code in JSP:

```
<ww:url value="somePage"><ww:param name="personId" value="%{personId}" /></ww:url>
```

In FreeMarker, you can simplify this as:

```
<@ww.url value="somePage" personId="%{personId}" />
```

In addition to being able to replace cases where you might use the param tag, you can also use this functionality when building additional templates or themes for your [Form Tags](#). For example, suppose you created a "three column" theme to replace the typical two column theme (xhtml). You might want an additional parameter to display in the third column called "description". Your form can be:

```
<@ww.form action="updatePerson"><@ww.textfield label="First name" name="firstName" description="..." /><@ww.submit value="Update" /></@ww.form>
```

And then in your new template you can refer to the description using **`${parameters.description}`**.



Sometimes you may still wish to use the param tag, such as when you are nesting complex HTML within tags. The param tag has support beyond what FreeMarker can provide as inline attributes: it can take the entire body of the param tag and apply that as the *value* attribute.

Attribute Types

Remember that all tag attributes must first be set as Strings – they are then later evaluated (using [OGNL](#)) to a different type, such as List, int, or boolean. This generally works just fine, but it can be limiting when using FreeMarker which provides more advanced ways to apply attributes. Suppose the following example:

```
<@ww.select label="Foo label - ${foo}" name="${name}" list="%{{1, 2, 3}}"/>
```

What will happen here is that each attribute will be evaluated to a string as best it can. This may involve calling the `toString()` method on the internal FreeMarker objects in the hash. In this case, all objects will end up being exactly what you would expect. Then, when the tag runs, the *list* attribute will be converted from a String to a List using [OGNL](#)'s advanced collection support.

But suppose you wish to use FreeMarker's list or hash support instead? You can do this:

```
<@ww.select label="Foo label - ${foo}" name="${name}" list={1, 2, 3}/>
```

Notice that the list attribute no longer has quotes around it. Now it will come in to the tag as an object that can't easily be converted to a String. Normally, the tag would just call `toString()`, which would return "[1, 2, 3]" and be unable to be converted back to a List by OGNL. Rather than go through all this back and forth anyway, the FreeMarker tag support within WebWork will recognize collections and not pass them through the normal tag attribute, but instead set them directly in the **parameters** map, ready to be consumed by the template.

In the end, everything tends to do what you would expect, but it can help to understand the difference of when OGNL is being used and when it isn't, and how attribute types get converted.

JSP Tag Support

While WebWork provides native FreeMarker Tags, you might wish to use other third-party tags that are only available for JSP. Fortunately, FreeMarker has the ability to run JSP tags. To do so, you must include the `JspSupportServlet` outlined in [web.xml 2.1.x compatibility](#), as this allows the FreeMarker integration to get access to the required objects needed to emulate a JSP taglib container.

Once you've done that, you can simply add something like this in your templates:

```
<#assign cewolf=JspTaglibs["/WEB-INF/cewolf.tld"] />
```

```
...  
<@cewold.xxx ... />
```

JSP Tags

This page last changed on Oct 06, 2005 by [plightbo](#).

TODO: fill this out...

Non Form Tags


This page last changed on Sep 26, 2005 by [digi9ten](#).

1. [a](#)
2. [component](#)
3. [div](#)
4. [panel](#)
5. [table](#)
6. [tabbedpane](#)
7. [tabbedPanel](#)

a

This page last changed on Oct 16, 2005 by [roughley](#).

The a tag is primarily an AJAX tag, providing a remote call from the current page to another URL.

		Be sure to setup the page containing this tag to be Configured for AJAX		
Attribute	Type	Required	Default	Description
id	string	TRUE		The id to assign the component
href	string	TRUE		The URL to call to obtain the content
preInvokeJS	string	FALSE		A javascript snippet that will be invoked prior to the execution of the target href. If provided must return true or false. True indicates to continue executing target... false says do not execute link target. Possible uses are for confirm dialogs.
afterLoading	string	FALSE		Javascript code that will be

				executed after the remote call has been made
notifyTopics	string	FALSE		Topic names to post an event to after the remote call has been made
errorText	string	FALSE		The text to display to the user if there is an error fetching the content
showErrorTransportText	string	FALSE		true/false - when to show the error message as content when the URL had problems
name	string	TRUE		
value	string	FALSE		
required	boolean	FALSE		
disabled	boolean	FALSE		
theme	string	FALSE		This tag will usually use the ajax theme
template	string	FALSE		
cssClass	string	FALSE		
cssStyle	string	FALSE		
label	string	FALSE		
labelposition	string	FALSE		
tabindex	string	FALSE		

onclick	string	FALSE		
ondblclick	string	FALSE		
onmousedown	string	FALSE		
onmouseup	string	FALSE		
onmouseover	string	FALSE		
onmousemove	string	FALSE		
onmouseout	string	FALSE		
onfocus	string	FALSE		
onblur	string	FALSE		
onkeypress	string	FALSE		
onkeydown	string	FALSE		
onselect	string	FALSE		
onchange	string	FALSE		

Nested Elements

The remote A tag supports nested param elements. `<ww:param name="" value=""/>`

Handling Results

- Currently the remote a will try to evaluate the results as JavaScript. So the action defined in href should return a javascript snippet that will be executed upon return. Th
- The afterLoading method is called after the the above javascript gets evaluated. The afterLoading javascript does not currently have access to the returned data.

Example

```
<ww:a id="'link1'" theme="'ajax'" href="/DoIt.action" errorText="'An error occurred'"
showErrorTransportText="true">
  <ww:param name="'id'" value="'1'"/>
</ww:a>
```

Results in:

```
<a dojoType="BindAnchor" evalResult="true" id="link1" href="/DoIt.action?id=1"
errorHtml="An error occurred" showTransportError="true"></a>
```

Here is an example that uses the postInvokeJS. This example is in altSyntax=true:

```
<ww:a id="test" theme="ajax" href="/simpleResult.action" preInvokeJS="confirm(\`You
sure\`)">A</ww:a>
```



Remember that if you want your results to be executed... and your using sitemesh... you must NOT decorate the request w/ sitemesh.

Common Configuration

The following configuration is required for the ajax theme. It is suggested that this goes into a common template so that it doesn't need to be manually included on every page.

Content pulled from external source. Click [here](#) to refresh.

```
<script language="JavaScript" type="text/javascript">
  // Dojo configuration
  djConfig = {
    baseRelativePath: "<ww:url includeParams="none" value="/webwork/dojo/">",
    isDebug: false,
    debugAtAllCosts: true // not needed, but allows the Venkman debugger to work
with the includes
  };
</script>

<script language="JavaScript" type="text/javascript"
  src="<ww:url includeParams="none" value="/webwork/dojo/dojo.js"
/>"></script>
<script language="JavaScript" type="text/javascript"
  src="<ww:url includeParams="none" value="/webwork/CommonFunctions.js"
/>"></script>

<script language="JavaScript" type="text/javascript">
  dojo.require("dojo.io.BrowserIO");
  dojo.require("dojo.event.topic");
  dojo.require("webwork.widgets.Bind");
  dojo.require("webwork.widgets.BindDiv");
  dojo.require("webwork.widgets.BindButton");
  dojo.require("webwork.widgets.BindAnchor");
  dojo.hostenv.writeIncludes(); // not needed, but allows the Venkman debugger to
work with the includes
</script>
```



Lessons Learned

Dojo is particular about the onload event.

[This thread details the whole thing](#)

But if you want the cliff notes... Dont do this:

```
window.onload= function()
```

```
Unknown macro: { . . . }
```

This was preventing dojo from parsing the widgets, since dojo hooks into window.onload in dojo.js, which you just overwrote

You should use this :

```
<script language="JavaScript"
type="text/javascript">
dojo.event.connect (window, "onload" ,
function() {
...
});
```

Loosly Coupled Components

A benefit of using dojo as the basis of many of these components is being able to loosely couple UI components. There are 2 attributes of importance - "listenTopics" and "notifyTopics".

- If a component has a "notifyTopics" attribute, then after the processing has been completed a message will be published to the topic names supplied as a value (comma delimited).
- If a component has a "listenTopics" attribute, then when a message is published to the topic names supplied as a value (comma delimited), the component will re-refresh its content.

As well as this, you can publish to topic names with javascript code.

```
dojo.event.topic.publish( "topic_name", "content" );
```

The "topic_name" attribute is required, the "content" is not and most elements are triggered without having this attribute.

An example of using this approach would be that multiple remote DIV's could be

updated after a remote link has been clicked - especially if the remote links were controls/actions for a row in a table that is itself contained in a remote DIV.

[Developing WW Ajax Widgets](#) - Developers of WW who are interested in maintaining and creating new DOJO widgets should read this.

component

This page last changed on Oct 02, 2005 by [digi9ten](#).

component

Attribute	Type	Required	Default	Description
id	string	FALSE		
name	string	TRUE		
value	string	FALSE		
required	boolean	FALSE		
disabled	boolean	FALSE		
theme	string	FALSE		
template	string	FALSE		
cssClass	string	FALSE		
cssStyle	string	FALSE		
label	string	FALSE		
labelposition	string	FALSE		
tabindex	string	FALSE		
onclick	string	FALSE		
ondblclick	string	FALSE		
onmousedown	string	FALSE		
onmouseup	string	FALSE		
onmouseover	string	FALSE		
onmousemove	string	FALSE		
onmouseout	string	FALSE		
onfocus	string	FALSE		
onblur	string	FALSE		
onkeypress	string	FALSE		
onkeydown	string	FALSE		

onselect	string	FALSE		
onchange	string	FALSE		

div

This page last changed on Oct 16, 2005 by [roughley](#).

The div tag is primarily an AJAX tag, providing a remote call from the current page to update a section of content without having to refresh the entire page.



Be sure to setup the page containing this tag to be [Configured for AJAX](#)

Attribute	Type	Required	Default	Description
id	string	TRUE		The id to assign the DIV
name	string	FALSE		The name to assign the DIV
href	string	TRUE		The URL to call to obtain the content
delay	boolean	FALSE		How long to wait before fetching the content (in milliseconds)
updateFreq	boolean	FALSE		How often to re-fetch the content (in milliseconds)
loadingText	boolean	FALSE		The text to display to the user while the new content is being fetched (especially good if the content will take awhile)

errorText	boolean	FALSE		The text to display to the user if there is an error fetching the content
showErrorTransport	boolean	FALSE		true/false - when to show the error message as content when the URL had problems
listenTopics	boolean	FALSE		Topic name to listen to (comma delimited), that will cause the DIV's content to be re-fetched
afterLoading	boolean	FALSE		Javascript code that will be executed after the content has been fetched
theme	string	FALSE		This tag will usually use the ajax theme
template	string	FALSE		
cssClass	string	FALSE		
cssStyle	string	FALSE		
label	string	FALSE		
labelposition	string	FALSE		
tabindex	string	FALSE		

onclick	string	FALSE		
ondblclick	string	FALSE		
onmousedown	string	FALSE		
onmouseup	string	FALSE		
onmouseover	string	FALSE		
onmousemove	string	FALSE		
onmouseout	string	FALSE		
onfocus	string	FALSE		
onblur	string	FALSE		
onkeypress	string	FALSE		
onkeydown	string	FALSE		
onselect	string	FALSE		
onchange	string	FALSE		

Basic Functions

The remote DIV is handy for a few basic use cases.

Get remote data and refresh

First in its simplest form as mentioned above, it can load its contents from a remote div. Additionally it can refresh them periodically. So... whats that do for you? Well say for example want to show the latest weather from weather.com on your page and update it every 1 minutes. You could do this like this:

```
<ww:div
  id="weather"
  cssStyle="border: 1px solid yellow;"
  href="http://www.weather.com/address_to_feed"
  delay="2000"
  updateFreq="60000"
  errorText="There was an error"
  loadingText="loading...">Initial Content
</ww:div>
```

Initialize div with remote data

OK that was fun, but lets say that you want to initially load the data from a remote page then refresh it. This example contrasts to the above example in that the initial data is NOT just 'Initial Content'.

Here is how you can do that:

```
<ww:div href="/listPeople.action" theme="ajax" errorText="error opps"
        loadingText="loading..." id="cart-body" >
  <ww:action namespace="" name="listPeople" executeResult="true" />
</ww:div>
```

Just stick an action in the body... that will be the initial data

Be a listener for data

Perhaps the coolest aspect of the remote div is its ability to be a listener in a pub-sub model. Wow you say... HTML pages dont have a pub-sub model. That used to be true but... WebWork has turned that notion upside down... read about our rich Decoupled Components via Pub and Sub. So.... back to the example... the div can listen to topics in a pub sub model... did we mention how cool that is yet? Just use the listenTopics attribute to specify one or more topics(comma delimited) that this div should listen for to trigger a refresh.

```
<ww:div
  id="three"
  cssStyle="border: 1px solid yellow;"
  href="/ShowThankYouEmail.action"
  theme="ajax"
  listenTopics="user_downloads_webwork, user_loves_webwork"
  delay="1000">Initial Content</ww:div>

<ww:a
  id="link1"
  theme="ajax"
  href="/DownloadWebWork.action"
  notifyTopics="user_downloads_webwork, the_world_is_a_better_place"
  errorText="An Error occurred">Download</ww:a>
```

See how the ww:a and the ww:div are loosely coupled via a pub-sub model??? When users click on the hyperlink called Download, the WebWork anchor publishes a message to two topics, one called user_downloads_webwork and the other called

the_world_is_a_better_place. Our div listens to the first topic as well as a topic called user_loves_webwork. So using this model... anytime a user downloads webwork... or a user loves webwork... our website will personally thank them!!! Did I mention how cool this is?

Additional Functions

There are also javascript functions to refresh the content, stop and start the refreshing of the component. For the remote div with the component id "remotediv1":

To start refreshing use the javascript:

```
remotediv1.start();
```

To stop refreshing use the javascript:

```
remotediv1.stop();
```

To refresh the content use the javascript:

```
remotediv1.bind();
```

panel

This page last changed on Oct 16, 2005 by [roughley](#).

The panel tag is used together with the [tabbedPanel](#) tag.

Attribute	Type	Required	Default	Description
id	string	TRUE		The id to assign the panel
tabName	string	TRUE		The text of the tab to display in the header tab list
href	string	FALSE		The URL to call to obtain the content (required if remote panel)
remote	string	FALSE		true/false - determines whether this is a remote panel (ajax) or a local panel (content loaded into visible/hidden containers)
errorText	string	FALSE		The text to display to the user if there is an error fetching the content (for remote panel)
showErrorTransportText	string	FALSE		true/false - when to show the error

				message as content when the URL had problems (for remote panel)
listenTopics	string	FALSE		Topic name to listen to (comma delimited), that will cause the panels content to be re-fetched

Example

The following is an example of a tabbedpanel and panel tag utilizing local and remote content.

Content pulled from external source. Click [here](#) to refresh.

```

<ww:tabbedPanel id="test2" theme="simple" >
  <ww:panel id="left" tabName="left" theme="ajax">
    This is the left pane<br/>
    <ww:form >
      <ww:textfield name="tt" label="Test Text" /> <br/>
      <ww:textfield name="tt2" label="Test Text2" />
    </ww:form>
  </ww:panel>
  <ww:panel remote="true" href="/AjaxTest.action" id="ryh1" theme="ajax"
tabName="remote one" />
  <ww:panel id="middle" tabName="middle" theme="ajax">
    middle tab<br/>
    <ww:form >
      <ww:textfield name="tt" label="Test Text44" /> <br/>
      <ww:textfield name="tt2" label="Test Text442" />
    </ww:form>
  </ww:panel>
  <ww:panel remote="true" href="/AjaxTest.action" id="ryh21" theme="ajax"
tabName="remote right" />
</ww:tabbedPanel>

```

tabbedpane

This page last changed on Oct 02, 2005 by [digi9ten](#).

tabbedpane


Attribute	Type	Required	Default	Description
id	string	FALSE		
contentName	string	TRUE		name of collection to use
selectedIndex	integer	FALSE		
name	string	TRUE		
value	string	FALSE		
required	boolean	FALSE		
disabled	boolean	FALSE		
theme	string	FALSE		
template	string	FALSE	tabbedpane	
cssClass	string	FALSE		
cssStyle	string	FALSE		
label	string	FALSE		
labelposition	string	FALSE		
tabindex	string	FALSE		
onclick	string	FALSE		
ondblclick	string	FALSE		
onmousedown	string	FALSE		
onmouseup	string	FALSE		
onmouseover	string	FALSE		
onmousemove	string	FALSE		
onmouseout	string	FALSE		
onfocus	string	FALSE		

onblur	string	FALSE		
onkeypress	string	FALSE		
onkeydown	string	FALSE		
onselect	string	FALSE		
onchange	string	FALSE		

tabbedPanel

This page last changed on Oct 16, 2005 by [roughley](#).

The tabbedpanel component is primarily an AJAX component, where each tab can either be local content or remote content (refreshed each time the user selects that tab).

		Be sure to setup the page containing this tag to be Configured for AJAX		
Attribute	Type	Required	Default	Description
id	string	TRUE		The id to assign the component

This tag is used in together with the [panel](#) tag.

Example

The following is an example of a tabbedpanel tag utilizing local and remote content.

Content pulled from external source. Click [here](#) to refresh.

```
<ww:tabbedPanel id="test2" theme="simple" >
  <ww:panel id="left" tabName="left" theme="ajax">
    This is the left pane<br/>
    <ww:form >
      <ww:textfield name="tt" label="Test Text" /> <br/>
      <ww:textfield name="tt2" label="Test Text2" />
    </ww:form>
  </ww:panel>
  <ww:panel remote="true" href="/AjaxTest.action" id="ryh1" theme="ajax"
tabName="remote one" />
  <ww:panel id="middle" tabName="middle" theme="ajax">
    middle tab<br/>
    <ww:form >
      <ww:textfield name="tt" label="Test Text44" /> <br/>
      <ww:textfield name="tt2" label="Test Text442" />
    </ww:form>
  </ww:panel>
  <ww:panel remote="true" href="/AjaxTest.action" id="ryh21" theme="ajax"
tabName="remote right" />
</ww:tabbedPanel>
```

Additional Configuration

If you are looking for the "nifty" rounded corner look, there is additional configuration. This assumes that the background color of the tabs is white. If you are using a different color, please modify the parameter in the Rounded() method.

```
<link rel="stylesheet" type="text/css" href="<ww:url value="/webwork/tabs.css"/>">
<link rel="stylesheet" type="text/css" href="<ww:url
value="/webwork/niftycorners/niftyCorners.css"/>">
<link rel="stylesheet" type="text/css" href="<ww:url
value="/webwork/niftycorners/niftyPrint.css"/>" media="print">
<script type="text/javascript" src="<ww:url
value="/webwork/niftycorners/nifty.js"/>"></script>
<script type="text/javascript">
    dojo.event.connect(window, "onload", function() {
        if (!NiftyCheck())
            return;
        Rounded("li.tab_selected", "top", "white", "transparent", "border
#ffffffS");
        Rounded("li.tab_unselected", "top", "white", "transparent", "border
#ffffffS");
        // "white" needs to be replaced with the background color
    });
</script>
```

table

This page last changed on Oct 02, 2005 by [digi9ten](#).

table

Attribute	Type	Required	Default	Description
id	string	FALSE		
sortColumn	integer	FALSE		
sortOrder	string	FALSE		
modelName	string	TRUE		
name	string	TRUE		
value	string	FALSE		
required	boolean	FALSE		
disabled	boolean	FALSE		
theme	string	FALSE		
template	string	FALSE	table	
cssClass	string	FALSE		
cssStyle	string	FALSE		
label	string	FALSE		
labelposition	string	FALSE		
tabindex	string	FALSE		
onclick	string	FALSE		
ondblclick	string	FALSE		
onmousedown	string	FALSE		
onmouseup	string	FALSE		
onmouseover	string	FALSE		
onmousemove	string	FALSE		
onmouseout	string	FALSE		
onfocus	string	FALSE		

onblur	string	FALSE		
onkeypress	string	FALSE		
onkeydown	string	FALSE		
onselect	string	FALSE		
onchange	string	FALSE		

Non-UI Tags

This page last changed on Jun 06, 2005 by [plightbo](#).

These are tags that interact with the value stack, and control the logic of the page.

Tag Name	Description
<u>Common Tags</u>	
	Add parameters to tags that support it
	Fetches a value and prints it
	Add an object of your choice to the top of the value stack
	Create your own named variables
URL tag (<ww:url />)	Builds an encoded URL
<u>Componentisation Tags</u>	
	Provides another method to call Actions
	Instantiate a bean that can be used to access functionality
	Used to include another page or action
<u>Flow-Control Tags</u>	
	Used to determine if a statement is true or false
	Used to determine if a statement is true or false after a previous test.
	Used to determine if the preceding statement was false
<u>Iteration Tags</u>	
	Iterate over a value
	Create Iterator
	Append a list of iterators
	Iterate over a portion of an iterable object

	Merge several iterators into one
	Sort an iterator

Common Tags

<ww:param />

Allows you to add parameters to tags that support adding parametric tags.

attribute	required	description
value	no	This attribute is used to pass data to the tag.
name	no	The name of the action to invoke.

You can place param tags within the body of parametric supporting tags and param will add its parameter to its parent. It evaluates the body as the value if no value is given.

In this example, each param will add its parameter to Counter. This means param will call Counter's appropriate setter method.

```
<ww:bean name="'webwork.util.Counter'" id="year">
  <ww:param name="'first'" value="text('firstBirthYear')"/>
  <ww:param name="'last'" value="2000"/>

  <ui:combobox label="'Birth year'" size="6" maxlength="4" name="'birthYear'"
list="#year"/>
</ww:bean>
```

[return to top](#)

<ww:property />

The property tag fetches a value and prints it

attribute	required	description
id	no	This attribute assigns a unique name to an element (note).
value	no	This attribute is used to pass data to the tag.
escape	no	Determines if the contents should be escaped appropriately for valid HTML characters

Some examples will illustrate these different uses:

```
Print getX().getY()
<ww:property value="x.y"/>
```

HTML characters will be escaped by default, whereas the contents of property tags with bodies will not be escaped. This behavior can be overridden by explicitly setting the escape attribute. Quoted text that is escaped will have its outer quotes stripped.

Note also that if the property tag has an empty body, it behaves the same as having no body and prints the value, though both spaces and carriage returns constitute nonempty content.

[return to top](#)

<ww:push />

Using ww:push, you can add an object of your choice to the top of the value stack.

attribute	required	description
value	yes	This attribute is used to pass data to the tag.

This is similar to what you can do with ww:set (see below), so read both before

deciding which to use.

```
<ww:push value="counter">
  <ww:property value="count"/>
</ww:push>
```

To make an action available on the stack:

```
<ww:action name="'SomeAction'" id="sa"/>
<ww:push value="#sa">
  foo = <ww:property value="foo"/>
</ww:push>
```

[return to top](#)

<ww:set />

You can create your own named variables from within a JSP using the `ww:set` tag. Reference your variable later using the `#` `variableName` notation.

attribute	required	description
name	yes	Unique name for the variable, accessed as "#name".
value	no	This attribute is used to pass data to the tag.
scope	no	Scope of the variable: page, request, session, application

Sets the value of an object in the VS to a scope. If the value is not given, the top of the stack is used. If the scope is not given, the default scope is the action context which is only available in the PageContext if no action has been executed on the same request.

Componentisation Tags

[return to top](#)

<ww:action />

Action tag provides another method to call Actions. This is an alternative way to invoke an action besides calling an url; i.e. - *.action that would be sent to the ServletDispatcher.

attribute	required	description
id	no	This attribute assigns a unique name to an element (note).
name	yes	The name of the action to invoke.
namespace	no	Namespace of this action
executeResult	no	Whether to execute result

If the **id** attribute is given, the executed action is assigned a name reference that can be later retrieved from the context "**#id**". This is the most common use, to execute an action and get it onto the stack.

If you specify "executeResult=true", the action will execute **and** continue to the result. This is used to act like include and return the rendered view.

In this example, the ClientInfo action will be executed and its methods will be used to retrieve information and perform a conditional test.

```
<ww:action name="ClientInfo" id="cinfo"><ww:param name="detailedMode"
value="false"/></ww:action>
Browser:<ww:property value="#cinfo.browser"/><br>
Version:<ww:property value="#cinfo.version"/><br>
Supports GIF:<ww:if test="#cinfo.supportsType('image/gif') ==
true">Yes</ww:if><ww:else>No</ww:else><br>
```

[return to top](#)

<ww:bean />

Create a JavaBean and instantiate its properties. It is then placed in the ActionContext for later use.

attribute	required	description
id	no	This attribute assigns a unique name to an element (note).
name	yes	The name of the action to invoke.

In this example, Counter is used as a bean. We can now call the methods we desire. In this case, we setFirst() to first birth year which is 1975 and we setLast() to 2000. We then display a combo box using Counter as an Iterator.

```
<ww:bean name="'webwork.util.Counter'" id="year">
  <ww:param name="'first'" value="text('firstBirthYear')"/>
  <ww:param name="'last'" value="2000"/>

  <ui:combobox label="'Birth year'" size="6" maxlength="4" name="'birthYear'"
list="#year"/>
</ww:bean>
```

[return to top](#)

<ww:include />

Used to include another page or action.

attribute	required	description
page	no	Name of page or action.
value	no	This attribute is used to pass data to the tag.

In this example, beaninfo.jsp will introspec on people0 which is a Person. Take a look at beaninfo.jsp example and notice how it retrieves the parent value off the

ValueStack with "..".

```
<ww:property value="people[0]">
  <ww:include value="'beaninfo.jsp'"/>
</ww:property>
```

In this example, an Action is invoked.

```
<h1>RSS viewer</h1>
<ww:include value="'rss.viewer.action'"/>
```

Flow Control Tags

[return to top](#)

<ww:if />

Used to determine if a statement is true or false.

attribute	required	description
id	no	This attribute assigns a unique name to an element (note).
test	yes	This attribute is the conditional expression evaluated by WW's parser. It returns boolean true or false.

In this example, if will evaluate its body since the test condition is true. `elseif` and `else` will not evaluate.

```
<ww:if test="true == true">
  <b>if: Success</b>
</ww:if>
```

```

<ww:elseif test="true == true">
  <b>elseif: Failure</b>
</ww:elseif>

<ww:else>
  <b>else: Failure</b>
</ww:else>

```

[return to top](#)

<ww:elseif />

Used to determine if a statement is true or false after a previous test.

attribute	required	description
id	no	This attribute assigns a unique name to an element (note).
test	yes	This attribute is the conditional expression evaluated by WW's parser. It returns boolean true or false.

In this example, elseif will evaluate its body since its test condition is true and if is false.

```

<ww:if test="true == false">
  <b>if: Failures</b>
</ww:if>

<ww:elseif test="true == true">
  <b>elseif: Success</b>
</ww:elseif>

<ww:else>
  <b>else: Failure</b>
</ww:else>

```

[return to top](#)

<ww:else />

Used to determine if the preceding statement was false.

attribute	required	description
id	no	This attribute assigns a unique name to an element (note).

In this example, else will evaluate its body since both if and elseif conditions are false.

```
<ww:if test="true == false">
  <b>if: Failures</b>
</ww:if>

<ww:elseif test="true == false">
  <b>elseif: Failure</b>
</ww:elseif>

<ww:else>
  <b>else: Success</b>
</ww:else>
```

Iteration Tags

[return to top](#)

<ww:iterator />

Iterator will iterate over a value. An iterable value can be either of: java.util.Collection, java.util.Iterator, java.util.Enumeration, java.util.Map, array, XML Node, or XML NodeList.

attribute	required	description
id	no	This attribute assigns a unique name to an element (note).
status	no	This attribute indicates the name of the IteratorStatus object to be exposed. An

		IteratorStatus allows one to get information about the status of the iteration: getCount(), getIndex(), isFirst(), isLast(), isEven(), isOdd().
value	no	This attribute is used to pass data to the tag.

In this example, iterator will iterate over Counter. property will output the current value which is 1 through 10.

```
<ww:bean name="'webwork.util.Counter'">
  <ww:param name="'last'" value="10"/>

  <ww:iterator>
    <ww:property/><br />
  </ww:iterator>
</ww:bean>
```

In this example, we use a couple of IteratorStatus to see where we are within iterations.

```
<h1>Testing iterator status</h1>

<ww:bean name="'webwork.util.Counter'" id="rowcounter">
  <ww:param name="'first'" value="0"/>
  <ww:param name="'last'" value="5"/>
</ww:bean>

<table border="1">
  <ww:iterator value="#rowcounter" status="rowstatus">
    <tr>
      <ww:bean name="'webwork.util.Counter'" id="colcounter">
        <ww:param name="'first'" value="0"/>
        <ww:param name="'last'" value="5"/>
      </ww:bean>

      <ww:iterator value="#colcounter" status="colstatus">
        <!--
          if it is (first row) or (first column) or (last row) then
          output the column number.
        -->
        <ww:if test="#rowstatus.first==true || #colstatus.first==true ||
#rowstatus.last==true">
          <th><ww:property value="#colstatus.count"/></th>
        </ww:if>

        <ww:else>
          <td><ww:property/></td>
```

```

        </ww:else>

    </ww:iterator>

</tr>
</ww:iterator>
</table>

```

Here we use the `IteratorStatus` determine every other row to insert an extra line break. This is very useful for shading alternate rows in an HTML table. Both `even` and `odd` attributes are available.

```

<ww:iterator status="status">
    <ww:if test="#status.odd == true"> <br /> </ww:if>
    <br />
</ww:iterator>

Here we use the IteratorStatus determine every fourth row to insert an extra line
break.
<ww:iterator status="status">
    <ww:if test="#status.modulus(4) == 0"> <br /> </ww:if>
    <br />
</ww:iterator>

```

Following are the list of operations available on the status object:

- `even` : boolean - returns true if the current iteration is even
- `odd` : boolean - returns true if the current iteration is odd
- `count` : int - returns the count (1 based) of the current iteration
- `index` : int - returns the index (0 based) of the current iteration
- `first` : boolean - returns true if the iterator is on the first iteration
- `last` : boolean - returns true if the iteration is on the last iteration
- `modulus(operand : int)` : int - returns the current count (1 based) modulo the given operand

[return to top](#)

<ww:generator />

Generate will create Iterators from `val`.

attribute	required	description
id	no	This attribute assigns a unique name to an element (note).
count	no	This attribute indicates how many items there are.
separator	no	This attribute is the character the StringTokenizer will use to create tokens.
val	yes	This attribute is the list of values the generator should use to create tokens.

In this example, two Iterators are created. One for `val="foo,bar,xyzzy"` and the other for `val=" "`.

```
<h1>Testing append, subset, and value generators</h1>

<table border="1">
  <ww:bean name="'webwork.util.Counter'">
    <ww:param name="'last'" value="5"/>
    <ww:iterator id="colcount">
      <tr>

        <!--
          Generator will create an Iterator that has 5 items.
          The first 3 are "foo,bar,xyzzy". Item 4 and 5 will be
          foo and bar respectively. If the count is more than
          the items, you start over.
        -->

        <ww:generator val="'foo,bar,xyzzy'" separator="','" count="#colcount"
id="values"/>


        <!--
          Generator will create an Iterator that has infinite
          . Count=-1 means indefinite.
        -->
        <ww:generator val="' '" count="-1" id="space"/>
        <ww:append>
          <ww:param name="'source'" value="#values"/>
          <ww:param name="'source'" value="#space"/>

          <ww:subset count="6">
            <ww:iterator>
```

```

        <td width="40"><ww:property/></td>
      </ww:iterator>
    </iterator:subset>
  </iterator:append>
</tr>
</ww:iterator>
</ww:bean>
</table>

```

 This tag is mostly superfluous, now that we can do this in OGNL:

```

<ww:iterator value="{1, 2, 3, 4}">
</ww:iterator>

```

[return to top](#)

<ww:append />

Append will append a list of iterators. The values of the iterators will be appended and treated as one iterator. The outputs from the iterator will be in the sequence the sources were added.

attribute	required	description
id	no	This attribute assigns a unique name to an element (note).

In this example, the two iterators #values and #spaces are appended. This means #spaces values are after #values.

```

<h1>Testing append, subset, and value generators</h1>

<table border="1">
  <ww:bean name="'webwork.util.Counter'">
    <ww:param name="'last'" value="5"/>
    <ww:iterator id="colcount">
      <tr>
        <ww:generator val="'foo,bar,xyzzzy'" separator="','" count="#colcount"
id="values"/>
        <ww:generator val="' ' " count="-1" id="space"/>
        <ww:append>
          <ww:param name="'source'" value="#values"/>
          <ww:param name="'source'" value="#space"/>

        <ww:subset count="6">

```

```

        <ww:iterator>
            <td width="40"><ww:property/></td>
        </ww:iterator>
    </iterator:subset>
</iterator:append>
</tr>
</ww:iterator>
</ww:bean>
</table>

```

[return to top](#)

<ww:subset />

Subset will iterate over a portion of its source. It will start at start and continue for count. You can set count to -1 if you want to iterate until the end of source. If you do not supply a source, the current object on the ValueStack- "." will be used.

attribute	required	description
id	no	This attribute assigns a unique name to an element (note).
count	no	This attribute indicates how many items there are.
source	no	This attribute is the source the tag will use to perform work on. It may be Enumeration, Iterator, or a Collection.
start	no	This attribute indicates the index to start reading.

In this example, subset will iterate over 6 items for the current object in the ValueStack.

```

<h1>Testing append, subset, and value generators</h1>

<table border="1">
    <ww:bean name="'webwork.util.Counter'">
        <ww:param name="'last'" value="5"/>

```

```

<ww:iterator id="colcount">
  <tr>
    <ww:generator val="'foo,bar,xyzy'" separator="','" count="#colcount"
id="values"/>
    <ww:generator val="' '" count="-1" id="space"/>
    <ww:append>
      <ww:param name="'source'" value="#values"/>
      <ww:param name="'source'" value="#space"/>

      <ww:subset count="6">
        <ww:iterator>
          <td width="40"><ww:property/></td>
        </ww:iterator>
      </iterator:subset>
    </iterator:append>
  </tr>
</ww:iterator>
</ww:bean>
</table>

```

[return to top](#)

<ww:merge />

Merge several iterators into one. It weaves them together. If one iterator runs out, it will drop off and the others will continue weaving until there are no more values.

attribute	required	description
id	no	This attribute assigns a unique name to an element (note).

In this example, #foo, #bar, and #xyzy iterators are merged together. So, the output will be foo, bar, xyzy until #foo and #xyzy iterators run out in which case #bar will finish.

```

Three value generators with merge and subset limits:<br>
<ww:generator val="'foo'" count="5" id="foo"/>
<ww:generator val="'bar'" count="10" id="bar"/>
<ww:generator val="'xyzy'" count="5" id="xyzy"/>
<ww:merge>
  <ww:param name="'source'" value="#foo"/>
  <ww:param name="'source'" value="#bar"/>
  <ww:param name="'source'" value="#xyzy"/>

  <ww:subset count="30">
    <ww:iterator status="status">
      <ww:property value="#status.count"/><ww:property/><br>

```

```
</ww:iterator>
</iterator:subset>
</iterator:merge>
```

[return to top](#)

<ww:sort />

Sort allows you to sort an iterator. It uses `Collections.sort()` given the comparator you supply.

attribute	required	description
id	no	This attribute assigns a unique name to an element (note).
comparator	yes	This attribute will be the Comparator used to sort the Collection.
source	no	This attribute is the source the tag will use to perform work on. It may be Enumeration, Iterator, or a Collection.

In this example, we sort ascending.

```
<ww:bean name="com.opensymphony.webwork.util.Counter" id="counter">
  <ww:param name="first" value="0"/>
  <ww:param name="last" value="5"/>
</ww:bean>

<ww:bean name="com.opensymphony.webwork.util.Sorter" id="sorter"/>

Ascending:<br />
<ww:sort source="#counter" comparator="#sorter.ascending">
  <ww:iterator>
    <ww:property/><br />
  </ww:iterator>
</iterator:sort>
```

In this example, we sort descending.

```

<ww:bean name="com.opensymphony.webwork.util.Sorter" id="sorter"/>

<ww:bean name="com.opensymphony.webwork.util.Counter" id="counter">
  <ww:param name="'first'" value="0"/>
  <ww:param name="'last'" value="5"/>
</ww:bean>

Descending:<br>
<ww:sort source="#counter" comparator="#sorter.descending">
  <ww:iterator>
    <ww:property/><br>
  </ww:iterator>
</ww:sort>

```

In this example, we sort ascending over strings.

```

<ww:bean name="com.opensymphony.webwork.util.Sorter" id="sorter"/>

Sorting strings:<br>
<ww:generator val="Rickard,Maurice,Hristo" separator="," id="names"/>
<ww:sort source="#names" comparator="#sorter.ascending">
  <ww:iterator>
    <ww:property/><br>
  </ww:iterator>
</iterator:sort>

```

Notes

i Id The "id" attribute assigns a name to an element. This name must be unique in a document. This attribute is the standard id supported by JSP TagSupport and is therefore always a string. You do not need to indicate a string literal as you would for the rest of WW attributes; i.e. - `id="age"`. Instead you should use `id="age"`.

i It's very important to note that all tags that insert something into the valuestack (like `i18n` or `bean` tags) will remove those objects from the stack on its end tag. So, if you instantiate a bean with the `bean` tag (`<ww:bean name="br.univap.fcc.sgpw.util.FormatterHelper">`) that bean will be available on the valuestack only until the `</ww:bean>` tag.

URL tag

This page last changed on Dec 14, 2004 by [plightbo](#).

The URL tag builds an encoded URL. If you do not include a value, then the tag will point to the current page.

attribute	required	description
value	no	This attribute is used to pass data to the tag.
id	no	When specified, causes the URL not to be printed to the output but rather stored in the ActionContext using the id as the key
scheme	no	can be "http" or "https"
includeContext	no	Determines whether the context path should be prepended to absolute urls or not. Default is true
encode	no	Determines if the contents should be escaped appropriately for valid HTML characters
includeParams	no	The includeParams attribute may have the value 'none' (no params), 'get'(only GET params) or 'all'(GET and POST params). It is used when the url tag is used without a value or page attribute. Its value is looked up on the ValueStack. If no includeParams is specified then 'get' is used.

In this example, the form action value will be an url hiturl.action that is encoded.

```
<form action="<ww:url value='hiturl.action'"/>" method="POST">
  ...
</form>
```

In this example, we are adding name/value pairs to the URL. The URL tag will build up the URL appropriately. You can also place them in the normal way with "?"; i.e., - 'hiturl.action?user=john'.

```
<form action="<ww:url value='hiturl.action'>
  <ww:param name='user' value='john'"/>
  </ww:url>" method="POST">
  ...
</form>
```

By default, port 80 is assumed to be the "http" port and port 443 is assumed to be the "https" port. However, some servers, such as Tomcat, use different default ports, such as 8080 and 8443. You can change these values by setting the configuration elements in webwork.properties:

- webwork.url.http.port
- webwork.url.https.port

Tag Syntax

This page last changed on Sep 12, 2005 by [plightbo](#).

The tag syntax in WebWork is extremely easy to understand. To quickly get started, all you need to know is that all attributes are applied as Strings initially. They are then parsed for the syntax `%{ ... }`, and anything in between the braces is evaluated against the value stack. Finally, the resulting String is either applied directly



Upgrade note!

The tag syntax was not always this easy – if you are upgrading from WebWork 2.1.7 or previous versions, you may wish to read about the [altSyntax](#).

Some Examples

The *altSyntax* is an option that can be defined in [webwork.properties](#). By default it is set to true and it is **strongly** recommend you do not change that unless you are upgrading from WebWork 2.1.7 or previous versions.

The altSyntax changes the behavior of how tags are interpreted. Instead of evaluating each tag parameter against the value stack and needing single quotes to mark string literals, only marked expressions are evaluated.

Example:

the following code uses the [Tag Syntax](#):

```
<ww:iterator value="cart.items">
  ...
  <ww:textfield label="'Cart item No.' + #rowstatus.index + ' note'"
name="'cart.items[' + #rowstatus.index + '].note'" value="note" />
</ww:iterator>
```

this is somewhat counter intuitive to normal HTML tag behaviour, and you get loads of single quotes. Now the same example in altSyntax:

```
<ww:iterator value="cart.items">
  ...
  <ww:textfield label="Cart item No. %{#rowstatus.index} note"
name="cart.items[%{#rowstatus.index}].note" value="%{note}" />
</ww:iterator>
```

Only expressions enclosed with `%{ }` are evaluated. The code is shorter and clearer, very similar to JSTL EL usage. Quoting problems, eg. with javascript function calls, are avoided.

In order to fully understand why this option exists and what the differences are, it is best to get a bit of history about WebWork.



If you are *not* upgrading from WebWork 2.1.7 or previous versions and you don't care about the history of WebWork's

evolution, you can skip this section. See the [Tag Syntax](#) section for more information on the standard tag syntax support

History

In WebWork 2.1.4, the altSyntax option was introduced. The book, WebWork in Action, while based around WebWork 2.1.7, was entirely written with the assumption that the altSyntax was enabled. As of WebWork 2.2, the altSyntax is turned on by default and eventually the old syntax will no longer be supported and will be removed from the code.

In order to understand why this was

Themes and Templates

This page last changed on Oct 06, 2005 by [plightbo](#).

An outline of themes and templates and how they related. Grab content from [Themes](#) and [Templates](#) (the content is probably very out of date, but could be useful). Some content may also be useful from [WebWork 2 UI Tag Guide](#)

Overview

In WebWork, the UI tags wrap generic HTML controls while providing tight integration with the core framework. The tags have been designed to minimize the amount of logic in compiled code and delegate the actual rendering of HTML to a template system. Templates can be grouped together and separated into different [Themes](#). The UI tags attempt to cover the most common scenarios, while providing a Component Tag for creating custom components. The UI tags also provide built-in support for displaying inline error messages.

Templates

WebWork uses the Velocity template system to render the actual HTML output for all UI tags (jsp and velocity). A default implementation of all templates has been included with the core distribution allowing users to use WebWork's UI tags "out of the box". Templates can be edited individually or replaced entirely allowing for complete customization of the resulting HTML output. In addition, the default template can be overridden on a per tag basis allowing for a very fine level of control.

The templates can be found in the distribution package in a directory called `template` under the `src/java` directory or in the `webwork-x.x.jar` file. For template customization, copy the `template` directory into the root directory of your application or place it in the classpath. Webwork will attempt to load the templates from those two places first. Otherwise, the templates will be loaded from the webwork jar file.

```
/myApp
/META-INF
/WEB-INF
/template
```

Inside the `template` directory, you will find two template sets called [Themes](#) (xhtml and simple). The default template set that is used with UI tags is 'xhtml' unless specified by the theme attribute in your UI tag or in the [webwork.properties](#) file with the `webwork.ui.theme` variable. You can modify the pre-existing templates or create

your own.

The AbstractUI class is the base class that other UI tags extend. It provides a set of attributes that are common across UI components. The AbstractUI class defines an abstract method:

```
protectedabstractString getTemplateName();
```

The AbstractUI class will load the template specified by the subclass or optionally, a template specified by the user using the template attribute. The following will load myOwnTextTemplate.vm for the textfield UI tag instead of the built in template text.vm

NOTE: You have to create a template file called myOwnTextTemplate.vm and store it in xhtml for this to work.

```
<!-- loads /template/xhtml/myOwnTextTemplate.vm --><ww:ui textfield  
label=" 'mylabel' " name=" 'myname' " template=" 'myOwnTextTemplate.vm' " />
```

otherwise

```
<!-- loads default /template/xhtml/text.vm --><ww:ui textfield label=" 'mylabel' "  
name=" 'myname' " />
```

Built in templates

The default templates that correspond to each UI tag are as follows:

UI tag	default template
checkboxList	checkboxlist.vm
checkbox	checkbox.vm
combobox	combobox.vm
component	empty.vm
doubleSelect	doubleselect.vm
file	file.vm

form	form.vm(to open) form-close.vm(to close)
hidden	hidden.vm
label	label.vm
password	password.vm
radio	radiomap.vm
select	select.vm
submit	submit.vm
tabbedpane	tabbedpane.vm
textarea	textarea.vm
textfield	text.vm
token	token.vm

Accessing variables

A VelocityContext object is created and used by all WW velocity views with the following context parameters:

- tag - a reference to the tag object
- stack - the current OgnlValueStack
- ognl - a reference to the utility class OgnlTool
- req - a reference to the HttpServletRequest object
- res - a reference to the HttpServletResponse
- webwork - instance of WebWorkUtil
- action - action associated with the current request/ActionInvocation
- parameters - map of the current parameters

These variables can be accessed in the template by using \$TAG_NAME where TAG_NAME is one of tag, stack, ognl, req, ...). The template file is then processed. A few examples:

NOTE: The bang (!) will print the value if its defined and "" if its not

```
$!req.requestURI
$!req.method
```

```
$!tag.templateDir  
$!tag.theme
```

```
$!parameters.name  
$!parameters
```

Understanding the Webwork Template System

Look at how the template is found and loaded. A peek into AbstractUITag shows us the string used to build the template:

```
protectedString buildTemplateName(String myTemplate, String myDefaultTemplate) {  
    ...  
    return "/" + getTemplateDir() + "/" + getTheme() + "/" + template;  
}
```

With the defaults, this will return the string for the textfield UI tag

```
/template/xhtml/text.vm
```

Webwork will attempt to find these values before it uses the default ones. You don't have to override any values and can modify the built in templates if you so desire(your choice). Webwork searches for these values in the order they are listed:

- getTemplateDir()
 - webwork.ui.templateDir value in webwork.properties
 - otherwise, "template" is returned
- getTheme()
 - in UI tag theme attribute
 - webwork.ui.theme value in webwork.properties
 - otherwise, "xhtml" is returned
- template
 - in UI tag template attribute
 - otherwise, defaults to specified template

Templates with CSS

The default templates define several properties for use with CSS when HTML is

generated from webwork tags. These properties can be found in a stylesheet located in the `/template/xhtml` directory called `styles.css`. You can use this stylesheet as a skeleton for your application and build on it or create your own, but remember you must include a link to the stylesheet within your jsp or velocity page.

`styles.css`:

```
.label {font-style:italic; }
.errorLabel {font-style:italic; color:red; }
.errorMessage {font-weight:bold; text-align: center; color:red; }
.checkboxLabel {}
.checkboxErrorLabel {color:red; }
.required {color:red;}
.requiredLabel {font-weight:bold; font-style:italic; }
```

referencing the stylesheet with a link inside your webpage (relative path or you can specify it from the root of your container):

```
<link rel ="stylesheet" type="text/css"
href="/webwork-example/template/xhtml/styles.css" title="Style">
```

Note: Webwork now has new attributes in the UI tags for more generic support of HTML styles and classes to make the look and feel even more flexible to implement. These are defined respectively as `cssStyle` and `cssClass`.

```
<ui:textfield label="'lebal'" name="'nmae'" cssStyle="'float:left; color:red'"
cssClass="'myclass'" />
```

Creating Custom Components

At first glance the component tag doesn't look that impressive. The ability to specify a single template and use a number of predetermined attributes looks rather lacking. But the supplied tag offers a number of benefits to developers.

Before diving right into the custom component, first I will identify some advantages to using the component tag to create your components. Then I will detail the two types of error messages in WebWork 2 and how our custom component (for displaying one of these types) fits into the equation. Finally, I will present a sample Action class, Jsp file and template file for our component. When we are finished, you will be able to incorporate the new component into your application.

Why use the component tag?

- removes the need to develop your own Jsp tag library
- provides integrated support for accessing the ValueStack
- leverages XWork's support for internationalization, localization and error handling
- faster prototyping using templates (editable text files) instead of compiled code
- re-use and combine existing templates

More on error message support:

In WebWork 2, there are two types of error messages: field error messages and action error messages. Field error messages are used to indicate a problem with a specific control and are displayed inline with the control. A number of tags provide built-in support for displaying these types of messages. Action error messages on the other hand, indicate a problem with executing an action. Many things can go wrong in a web application, especially an application that relies on external resources such as a database, remote web service, or other resource that might not be accessible during the execution of an action. Handling an error gracefully and presenting the user with a useful message can often be the difference in a positive user/customer experience and a bad one.

When these types of errors occur, it is more appropriate to display these messages separate from individual controls on the form. In the example below, we will create a custom component that can be used to display action error messages in a bulleted list. This component can then be used on all your forms to display these error messages.

The action class below was created to handle a promotion on the website: a free e-certificate. It will try to email the certificate, but an exception will be thrown.

Action class:

```
package example;

import com.opensymphony.xwork.ActionSupport;

public class AddUser extends ActionSupport {

    privateString fullname;
```

```

privateString email;

publicString execute() throws Exception {
    // we are ignoring field validation in this example
try {
    MailUtil.sendCertificate(email, fullname);
    } catch (Exception ex) {
        // there was a problem sending the email
// in a real application, we would also
// log the exception
        addActionError("We are experiencing a technical problem and have
contacted our support staff. " +
                        "Please try again later.");
    }

    if (hasErrors()) {
        return ERROR;
    } else {
        return SUCCESS;
    }
}

publicString getFullname() {
    return fullname;
}

public void setFullname(String fullname) {
    this.fullname = fullname;
}

publicString getEmail() {
    return email;
}

public void setEmail(String email) {
    this.email = email;
}
}

```

Jsp page:

```

<%@ taglib uri="webwork" prefix="ui" %><html><head><title>custom component
example</title></head><!-- don't forget this --><link rel ="stylesheet"
type="text/css" href="/webwork-example/template/xhtml/styles.css"
title="Style"><body><ui:form action="AddUser.action"
method="POST"><table><ui:component template="action-errors.vm" /><ui:textfield
label="Full Name" name="fullname" /><ui:textfield label="Email" name="email"
/><ui:submit name="submit" value="Send me a free E-Certificate!"
/></table></ui:form></body></html>

```

HTML output (before submitting):

```

<html><head><title>custom component example</title></head><link rel ="stylesheet"
type="text/css" href="/webwork-example/template/xhtml/styles.css"
title="Style"><body><form action="AddUser.action" method="POST" /><table><tr><td>

```

```
align="right" valign="top"><span class="label">Full Name:</span></td><td><input
type="text" name="fullname" value="" /></td></tr><tr><td align="right"
valign="top"><span class="label">Email:</span></td><td><input type="text"
name="email" value="" /></td></tr><tr><td colspan="2"><div align="right"><input
type="submit" name="submit" value="Send me a free
E-Certificate!" /></div></td></tr></table></form></body></html>
```

The template below will loop through any action errors and display them to the user in a bulleted list.

Template (action-errors.vm)

```
#set ($actionErrors = $stack.findValue("actionErrors"))

#if ($actionErrors)
<tr><td colspan="2"><span class="errorMessage">The following errors
occurred:</span><ul>
    #foreach ($actionError in $actionErrors)
    <li><span class="errorMessage">$actionError</span></li>
    #end
</ul></td></tr>
#end
```

HTML output (after submitting):

```
<html><head><title>custom component example</title></head><link rel ="stylesheet"
type="text/css" href="/webwork-example/template/xhtmll/styles.css"
title="Style"><body><form action="AddUser.action" method="POST" /><table><tr><td
colspan="2"><span class="errorMessage">The following errors occurred:</span><ul><li
class="errorMessage">
    We are experiencing a technical problem and have contacted our
    support staff. Please try again later.
</li></ul></td></tr><tr><td align="right" valign="top"><span
class="label">Full Name:</span></td><td><input type="text" name="fullname"
value="Sample User" /></td></tr><tr><td align="right" valign="top"><span
class="label">Email:</span></td><td><input type="text" name="email"
value="user@example.com" /></td></tr><tr><td colspan="2"><div align="right"><input
type="submit" name="submit" value="Send me a free
E-Certificate!" /></div></td></tr></table></form></body></html>
```

Themes

This page last changed on Oct 07, 2005 by [botah](#).

A Theme is a set of [Templates](#) used to customize web page development with UI tags. They provide a powerful mechanism to help web developers spice up the UI with a mixture of styles (colors, fonts, etc). For example, you may want half your form textfields to have a blue background and half a white background. A couple of notes:

- Webwork comes with 2 pre-defined themes; "simple" and "xhtml" (default). The default location webwork looks for themes in your web application is `/template`. The default theme is `xhtml`. Default, meaning that it will be used if you don't specify a theme attribute in your UI tag. Note: The default values can be overridden in your [webwork.properties](#) file with `webwork.ui.theme=` and `webwork.ui.templateDir=`.
- Custom themes can also be created to tailor your own needs. We recommend you consult the pre-defined templates as a starting point before you create your own.
- Every time a UI tag is used, the tag is rendered into html by referencing a template. So they play a key role in how fields look and are positioned in a page. Currently, only velocity templates are supported out of the box. However, expect to see more support for creating JSP templates in future releases.

Note: Before moving forward, it is recommended that you review how the webwork template system works. (see [Templates](#))

xhtml

`xhtml` comes configured as the default theme for Webwork. It extends the simple theme providing built in functionality for error reporting, table positioning, and labeling. Lets look at one of the most common UI tags used, `textfield`, and show the proper way to write your views with the `xhtml` theme.

As you may already know, the default UI template used for the `textfield` tag is `text.vm` located under the directory `/template/xhtml`.

```
#*
```

```

-- text.vm
*#
## notice the re-use of the simple theme template text.vm
#parse("/template/xhtml/controlheader.vm")
#parse("/template/simple/text.vm")
#parse("/template/xhtml/controlfooter.vm")

```

When this template is loaded, it will first parse and render the `/template/xhtml/controlheader.vm`. Within `controlheader.vm` you will notice functionality for error reporting, labeling and table positioning. If `ActionSupport` is returning with some errors, they are rendered into html using this this template. Also you will notice how it grabs the `parameter.label` value and positions it with the `textfield` using the table elements `tr` and `td`.

```

*#
-- controlheader.vm
*#
## Only show message if errors are available.
## This will be done if ActiveSupport is used.

#if( $fieldErrors.get($parameters.name) )
  #set ( $hasFieldErrors = $fieldErrors.get($parameters.name))
  #foreach ( $error in $fieldErrors.get($parameters.name))
    <tr>
      #if ( $parameters.labelposition == 'top')
        <td align="left" valign="top" colspan="2">
      #else
        <td align="center" valign="top" colspan="2">
      #end
      <span class="errorMessage">${webwork.htmlEncode( $error )}</span></td></tr>
    #end
  #end

## Provides alignment behavior with table tags
## if the label position is top,
## then give the label it's own row in the table
## otherwise, display the label to the left on same row

<tr>
  #if ( $parameters.labelposition == 'top')
    <td align="left" valign="top" colspan="2">
  #else
    <td align="right" valign="top">
  #end

    #if ( $hasFieldErrors)
      <span class="errorLabel">
    #else
      <span class="label">
    #end

## If you want to mark required form fields with an asterisk,
## you can set the required attribute
## Ex. <ui:textfield label="mylabel" name="myname" required="'true'" />
    #if ( $parameters.label)

```



```

        #if ($parameters.required) <span class="required">*</span> #end
        $!webwork.htmlEncode($parameters.label):
    #end
    </span></td>

## add the extra row
#if ($parameters.labelposition == 'top')
</tr><tr>
#end
    <td>

```

The next template being parsed in `/template/xhtml/text.vm` is `/template/simple/text.vm`. Here you see the actual html input text tag being rendered and how the parameters are passed in.

```

#*
-- text.vm
--
-- Required Parameters:
-- * label      - The description that will be used to identify the control.
-- * name       - The name of the attribute to put and pull the result from.
--               Equates to the NAME parameter of the HTML INPUT tag.
--
-- Optional Parameters:
-- * labelposition - determines where the label will be place in relation
--                   to the control. Default is to the left of the control.
-- * size         - SIZE parameter of the HTML INPUT tag.
-- * maxlength    - MAXLENGTH parameter of the HTML INPUT tag.
-- * disabled     - DISABLED parameter of the HTML INPUT tag.
-- * readonly     - READONLY parameter of the HTML INPUT tag.
-- * onkeyup      - onkeyup parameter of the HTML INPUT tag.
-- * tabindex     - tabindex parameter of the HTML INPUT tag.
-- * onchange     - onkeyup parameter of the HTML INPUT tag.
--
*#
<input type="text"
        name="$!webwork.htmlEncode($parameters.name)"
#if ($parameters.size)          size="$!webwork.htmlEncode($parameters.size)"
#end
#if ($parameters.maxlength)
maxlength="$!webwork.htmlEncode($parameters.maxlength)" #end
#if ($parameters.nameValue)
value="$!webwork.htmlEncode($parameters.nameValue)" #end
#if ($parameters.disabled == true) disabled="disabled"
#end
#if ($parameters.readonly)      readonly="readonly"
#end
#if ($parameters.onkeyup)
onkeyup="$!webwork.htmlEncode($parameters.onkeyup)" #end
#if ($parameters.tabindex)
tabindex="$!webwork.htmlEncode($parameters.tabindex)" #end
#if ($parameters.onchange)
onchange="$!webwork.htmlEncode($parameters.onchange)" #end
#if ($parameters.id)          id="$!webwork.htmlEncode($parameters.id)"
#end
#if ($parameters.cssClass)
class="$!webwork.htmlEncode($parameters.cssClass)" #end
#if ($parameters.cssStyle)

```

```
"$!webwork.htmlEncode($parameters.cssStyle)"      #end
/>
```

And finally, the controlfooter.vm is parsed to close up the td and tr tags that were previously opened in controlheader.vm

```
##
-- controlheader.vm
*#

</td></tr>
```

In our view, since the tr and td elements are already created for us, we can simply wrap them with table elements. For the sake of learning, we will just use normal html table objects, but feel free to look into how the table.vm tag gets rendered and possibly use that.

```
<%@ taglib uri="webwork" prefix="ui" %><link rel="stylesheet" type="text/css"
href="template/xhtml/styles.css" title="Style"><html><head><title>JSP
PAGE</title></head><body><form><table>
  <!-- we can set the required attribute to true if we want to
        display and asterisk next to required form fields
  -->

  <ui:textfield label="'Username'" required="'true'" name="'user'" /><ui:textfield
label="'Email'" name="'email'" /></table></form></body></html>

<link rel="stylesheet" type="text/css" href="template/xhtml/styles.css"
title="Style"><html><head><title>VM PAGE</title></head><body><form><table>
  #tag( TextField "label='Username'" "name='user'" )<br>
  #tag( TextField "label='Email'" "name='email'" )<br></table></form></body></html>
```

css_xhtml

The css_xhtml theme is a theme based entirely on css and thus avoids the dependency on tables.

Lets review the code to see how this theme works.

Generated HTML code

```
<div>
<p>
<label for="frm1_caseBean.title" class="label"><span
class="required">*</span>Title:</label>
<input name="caseBean.title" size="70" id="frm1__caseBean.title"
```

```
this);" type="text">
</p>
</div>
```

The first thing of note is the containing DIV tag. This is the container element that will be filled with Field Validation errors. The second element is the P, this is a block element and will thusly but both the lable and the control on its own line. notice how we use the 'for' attribute of the label tag? That helps to identify that this label is for a control. This feature is uber cool for checkboxes as it extends the clickable range of the checkbox to the label as well!!!

You might also have noticed the CSS class attributes... lets review these:

```
.label {
    font-style:italic;
    float:left;
    width:30%
}
.errorLabel {font-style:italic; color:red; }
.errorMessage {font-weight:bold; text-align: center; color:red; }
.checkboxLabel {}
.checkboxErrorLabel {color:red; }
.required {color:red;}
```

This style sheet is included in the WW2.2 packaging. All you need to do to take advantage of it is include this line in your header:

```
<link
href="<%=request.getContextPath()%>/webwork/template/css_xhtml/styles.css"
rel="stylesheet" type="text/css">
```

If you want to customize any attribute of these CSS rules.. just put your deltas into your own css file and include it AFTER the above css include.

Some interesting things about the theme are how it supports both standard validation and DWR validation. Lets review the structure of some generated code to understand how the css_xhtml theme does validation.

```
<div>

<p>

<div errorfor="frm1_bo.emailAddress" classname="errorMessage"
class="errorMessage">Incorrect Email</div><label classname="errorLabel"
```


```

for="frml_bo.emailAddress" class="errorLabel"><span
class="required">*</span>Email:</label>

<input name="bo.emailAddress" size="20" value="not_yet_d@efined.com"
id="frml_bo.emailAddress" onblur="validate(this);" type="text">
</p>
</div>

```

The validation gets inserted into the structure for you by either the validation.js script if your using the DWR Client Side validator or server side template if your using standard validation.

	<p>No Orientation Support</p> <p>Some themes support label orientation... ie: either top or bottom... this theme does not.</p>
---	---

simple

The `simple` theme provides no additional functionality from HTML tags (similar to struts). This theme is considered the low end of the structure and can be re-used (extended) like `xhtml` to add additional functionality or behavior. You can easily create your own theme and extend this one to create complex pages that fit your own needs. To use the pre-defined theme `simple`

```

<%@ taglib uri="webwork" prefix="ui" %><link rel ="stylesheet" type="text/css"
href="template/xhtml/styles.css" title="Style"><html><head><title>JSP
PAGE</title></head><body><form><ui:label name="'userlabel'" label="'user"
theme="'simple'"/><ui:textfield name="'user'" theme="'simple'"/><ui:label
name="'emaillabel'" label="'user" theme="'simple'"/><ui:textfield name="'email'"
theme="'simple'"/></form></body></html>

```

```

<link rel ="stylesheet" type="text/css" href="template/xhtml/styles.css"
title="Style"><html><head><title>VM PAGE</title></head><body><form>
#tag( Label   "name='userlabel'" "label='user'" "theme='simple'" )
#tag( TextField "name='user'" "theme='simple'" )<br>

#tag( Label   "name='emaillabel'" "label='email'" "theme='simple'" )
#tag( TextField "name='email'" "theme='simple'" )<br></form></body></html>

```

Creating your own theme

Creating themes is quite simple and can save valuable time enabling you to minimize UI code when it comes to creating complex UI pages. It is recommended you understand webwork templates before you continue (see [Templates](#)). The steps required to use a theme.

1. Define a name for your theme by creating a subdirectory under /template directory. The name of the subdirectory you create will be the same as the value you specify in your UI tag theme attribute.

```
/template/myTheme
```

```
<ui:textfield label="'foo'" name="'bar'" theme="'myTheme'" />
```

2. Create a velocity template for every UI tag that you want to use with your theme. For example, if you have forms with only textfields and nothing else, then all you need in your subdirectory is a text.vm template. Note: if you create a text.vm and reference another template with the parse tag, then you must make sure these templates are defined as well(ex. controlheader.vm).
3. If you want your new theme to be the default so you don't have to specify the theme attribute every time in the UI tag, then modify the webwork.ui.theme value in your webwork.properties file. Otherwise, you can just specify the theme attribute in all your UI tags.

As a good starting point, a good idea is to copy the contents of xhtml into a new subdirectory. Therefore, you can modify the templates and still refer back to the originals as a reference point.

Overview

In WebWork 2, the UI tags wrap generic HTML controls while providing tight integration with the core framework. The tags have been designed to minimize the amount of logic in compiled code and delegate the actual rendering of HTML to a template system. The UI tags attempt to cover the most common scenarios, while providing a Component Tag for creating custom components. The UI tags also provide built-in support for displaying inline error messages.

Template System

WebWork 2 uses the Velocity template system to render the actual HTML output for all UI tags. A default implementation of all templates has been included with the core distribution allowing users to use WebWork's UI tags "out of the box". Templates can be edited individually or replaced entirely allowing for complete customization of the resulting HTML output. In addition, the default template can be overridden on a per tag basis allowing for a very fine level of control. The default templates are located in the webwork-2.0.jar file under /decorators/xhtml.

Update (2003-11-13): It is now found in /template/xhtml

Note: This is going to change with the theme support that was added.

The AbstractUI class is the base class that other UI tags extend. It provides a set of attributes that are common across UI components. There are 3 attributes that determine which template is rendered:

- templateDir (defaults to [webwork.ui.templateDir](#) - /template/)
- theme (defaults to [webwork.ui.theme](#) - xhtml)
- templateName (default is tag dependent, subclasses must implement getDefaultTemplate() to provide this)

These 3 attributes combine to determine the location of the template: `templateDir` + `theme` + `templateName`. For example, the default template for the `select` tag is `"/template/xhtml/select.vm"`. Any one of these attributes can be overridden by the user at run time.

The `AbstractUI` class is responsible for loading the correct template. As part of the `doStartTag()` method, a `VelocityContext` object is created with the following items:

- `tag` - a reference to the tag object
- `stack` - the `ValueStack`
- `ognl` - a reference to the utility class `OgnlTool`
- `req` - a reference to the `HttpServletRequest` object

These variables can be accessed in the template by using `$TAG_NAME` where `TAG_NAME` is one of `tag`, `stack`, `ognl` or `req`. The template file is then processed.

Template support for CSS

The default templates provided with WebWork define two properties for use with CSS. The first property, `'label'`, is applied to text specified in a `label` attribute in a `JspTag`. The second property, `'errorMessage'`, is applied when displaying inline error messages. Many developers need to highlight error messages more prominently by using an alternative color. The code block below can be used in an external stylesheet or in a style element to display error messages in red.

```
.errorMessage {  
    color: red;  
}
```

Note: A default stylesheet is not provided with the core distribution. Users are encouraged to define their own stylesheet using the properties WebWork exposes or modify the supplied templates to work with an existing stylesheet.

Validation

Validation and error handling are an integrated part of the core framework. The UI tags extend this by providing built-in support for displaying inline error messages. To use this functionality, just have your actions implement the `ValidationAware` interface or extend the `ActionSupport` class. The `ActionSupport` class provides a default implementation of `ValidationAware` and also provides support for internationalization. UI tags that support displaying error messages include: Component Tag, Password Tag, Radio Tag, Select Tag, Textarea Tag and Textfield Tag.

A very simple example of validation is included below. The key point to understand is how to add an error message to a specific control. WebWork will display the specified message inline with the form element.

Users are encouraged to take a closer look at the validation framework in XWork. It provides a number of benefits including separating validation code from your action class, declaring rules based on field type, and providing different validation rules for each action alias. The XWork validation framework is based on an interceptor allowing validation to be added or removed from an action with a quick configuration file change. For more information see the [XW:Validation Framework](#).

Example: Displaying field errors

Action class:

```
import com.opensymphony.xwork.ActionSupport;

public class RegisterEmail extends ActionSupport {

    privateString email;

    public void setEmail(String email) {
        this.email = email;
    }

    publicString getEmail() {
        return email;
    }

    publicString execute() throws Exception {
        if (email == null || email.equals("")) {
            // the first parameter is the name attribute
            // specified in <ui:textfield>
            // the second parameter is the error message
            // to display
```



```

        addFieldError("email", "Email address is required.");
    }

    if (hasErrors()) {
        return ERROR;
    } else {
        return SUCCESS;
    }
}
}

```

Jsp page:

```

<%@ taglib uri="webwork" prefix="ui" %>

<html>
<head><title>validation example</title></head>
<style type="text/css">
.errorMessage {
    color: red;
}
</style>
<body>

<form name="frmRegister" action="RegisterEmail.action" method="POST">
    <table>
        <ui:textfield label="Email" name="email" value="" size="50" />
        <tr>
            <td colspan="2">
                <input type="submit" name="submit" value="Register" />
            </td>
        </tr>
    </table>
</form>

</body>
</html>

```

HTML output (before submitting):

```

<html>
<head><title>validation example</title></head>
<style type="text/css">
.errorMessage {
    color: red;
}
</style>
<body>

<form name="frmRegister" method="POST" action="RegisterEmail.action">
    <table>
        <tr>
            <td align="right" valign="top"><span class="label">Email:</span></td>
            <td>
                <input type="text" name="email" value="" size="50"/>
            </td>
        </tr>
    </table>

```

```

        </td>
    </tr>
    <tr>
        <td colspan="2">
            <input type="submit" name="submit" value="Register" />
        </td>
    </tr>
</table>
</form>

</body>
</html>

```

HTML output (after submitting):

```

<html>
<head><title>validation example</title></head>
<style type="text/css">
.errorMessage {
    color: red;
}
</style>
<body>

<form name="frmRegister" method="POST" action="RegisterEmail.action">
    <table>
        <tr>
            <td colspan="2"><span class="errorMessage">
                Email address is required.
            </span></td>
        </tr>
        <tr>
            <td align="right" valign="top"><span class="label">Email:</span></td>
            <td>
                <input type="text" name="email" value="" size="50"/>
            </td>
        </tr>
        <tr>
            <td colspan="2">
                <input type="submit" name="submit" value="Register" />
            </td>
        </tr>
    </table>
</form>

</body>
</html>

```

Example: Displaying action errors

Add the following to the JSP file above the form tag:

```
<ww:if test="hasErrors()"><p><span class="errorMessage"><ww:if  
test="hasActionErrors()"><b>Errors:</b><br><ul><ww:iterator  
value="actionErrors"><li><ww:property/></li></ww:iterator></ul></ww:if><ww:else><b>Please  
fix the errors marked in red to continue</b></ww:else></span></p></ww:if>
```

This will display any action errors that may have occurred.

Creating Custom Components

At first glance the component tag doesn't look that impressive. The ability to specify a single template and use a number of predetermined attributes looks rather lacking. But the supplied tag offers a number of benefits to developers.

Before diving right into the custom component, first I will identify some advantages to using the component tag to create your components. Then I will detail the two types of error messages in WebWork 2 and how our custom component (for displaying one of these types) fits into the equation. Finally, I will present a sample Action class, Jsp file and template file for our component. When we are finished, you will be able to incorporate the new component into your application.

Why use the component tag?

- removes the need to develop your own Jsp tag library
- provides integrated support for accessing the ValueStack
- leverages XWork's support for internationalization, localization and error handling
- faster prototyping using templates (editable text files) instead of compiled code
- re-use and combine existing templates

More on error message support:

In WebWork 2, there are two types of error messages: field error messages and action error messages. Field error messages are used to indicate a problem with a specific control and are displayed inline with the control. A number of tags provide built-in support for displaying these types of messages. Action error messages on the other hand, indicate a problem with executing an action. Many things can go wrong in a web application, especially an application that relies on external resources such as a database, remote web service, or other resource that might not be accessible during

the execution of an action. Handling an error gracefully and presenting the user with a useful message can often be the difference in a positive user/customer experience and a bad one.

When these types of errors occur, it is more appropriate to display these messages separate from individual controls on the form. In the example below, we will create a custom component that can be used to display action error messages in a bulleted list. This component can then be used on all your forms to display these error messages.

The action class below was created to handle a promotion on the website: a free e-certificate. It will try to email the certificate, but an exception will be thrown.

Action class:

```
package example;

import com.opensymphony.xwork.ActionSupport;

public class AddUser extends ActionSupport {

    privateString fullname;
    privateString email;

    publicString execute() throws Exception {
        // we are ignoring field validation in this example
    try {
        MailUtil.sendCertificate(email, fullname);
    } catch (Exception ex) {
        // there was a problem sending the email
        // in a real application, we would also
        // log the exception
        addActionError("We are experiencing a technical problem and have
        contacted our support staff. " +
            "Please try again later.");
    }

    if (hasErrors()) {
        return ERROR;
    } else {
        return SUCCESS;
    }
}

    publicString getFullname() {
        return fullname;
    }

    public void setFullname(String fullname) {
        this.fullname = fullname;
    }
}
```

```

    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }
}

```

Jsp page:

```

<%@ taglib uri="webwork" prefix="ui" %>

<html>
<head><title>custom component example</title></head>
<style type="text/css">
.errorMessage {
    color: red;
}
</style>
<body>

<ui:form action="AddUser.action" method="POST">
<table>
    <ui:component template="action-errors.vm" />
    <ui:textfield label="Full Name" name="fullname" />
    <ui:textfield label="Email" name="email" />
    <ui:submit name="submit" value="Send me a free E-Certificate!" />
</table>
</ui:form>

</body>
</html>

```

HTML output (before submitting):

```

<html>
<head><title>custom component example</title></head>
<style type="text/css">
.errorMessage {
    color: red;
}
</style>
<body>

<form action="AddUser.action" method="POST" />

<table>

    <tr>
        <td align="right" valign="top"><span class="label">Full Name:</span></td>
        <td>

```

```

<input type="text" name="fullname" value="" />
</td>
</tr>
<tr>
<td align="right" valign="top"><span class="label">Email:</span></td>
<td>
<input type="text" name="email" value="" />
</td>
</tr>

<tr>
<td colspan="2">
<div align="right">
<input type="submit" name="submit" value="Send me a free E-Certificate!"/>
</div>
</td>
</tr>

</table>
</form>

</body>
</html>

```

The template below will loop through any action errors and display them to the user in a bulleted list.

Template (action-errors.vm)

```

#set ($actionErrors = $stack.findValue("actionErrors"))

#if ($actionErrors)
<tr>
<td colspan="2">
<span class="errorMessage">The following errors occurred:</span>
<ul>
#foreach ($actionError in $actionErrors)
<li><span class="errorMessage">$actionError</span></li>
#end
</ul>
</td>
</tr>
#end

```

HTML output (after submitting):

```

<html>
<head><title>custom component example</title></head>
<style type="text/css">
.errorMessage {
    color: red;
}
</style>

```

```

<body>

<form action="AddUser.action" method="POST" />

<table>

<tr>
  <td colspan="2">
    <span class="errorMessage">The following errors occurred:</span>
    <ul>
      <li class="errorMessage">
        We are experiencing a technical problem and have contacted our
        support staff. Please try again later.
      </li>
    </ul>
  </td>
</tr>

  <tr>
    <td align="right" valign="top"><span class="label">Full Name:</span></td>
    <td>
      <input type="text" name="fullname" value="Sample User" />
    </td>
  </tr>

  <tr>
    <td align="right" valign="top"><span class="label">Email:</span></td>
    <td>
      <input type="text" name="email" value="user@example.com" />
    </td>
  </tr>

  <tr>
    <td colspan="2">
      <div align="right">
        <input type="submit" name="submit" value="Send me a free E-Certificate!" />
      </div>
    </td>
  </tr>

</table>
</form>

</body>
</html>

```

Changes from WebWork 1

TODO

UI Tags

This page last changed on Dec 14, 2004 by [plightbo](#).

Click on a tag to find more information on the tag. Note that all UI tags are now evaluated against the value stack so you need to (single) quote your literal string values.

The actual rendering of these tags can be customized. The location of the tag templates is defined by the `webwork.ui.theme` property in `webwork.properties`. See the [Themes](#) and [Templates](#) references for more details.

Name	Tag	Description
Checkbox tag	<code><ww:checkbox /></code>	render a checkbox input field
Checkboxlist tag	<code><ww:checkboxlist /></code>	render a list of checkboxes
Combobox tag	<code><ww:combobox /></code>	widget that fills a text box from a select
Component tag	<code><ww:component /></code>	render a custom ui widget
File tag	<code><ww:file /></code>	renders a file select input field
Form tag	<code><ww:form /></code>	defines an input form
Hidden tag	<code><ww:hidden /></code>	render a hidden field
Label tag	<code><ww:label /></code>	render a label that displays read-only information
Password tag	<code><ww:password /></code>	render a password input field
Radio tag	<code><ww:radio /></code>	renders a radio button input field
Select tag	<code><ww:select /></code>	renders a select element
Submit tag	<code><ww:submit /></code>	renders a submit button
Table tag	<code><ww:table /></code>	renders a table
Tabbedpane tag	<code><ww:tabbedpane /></code>	renders a tabbedpane

Textarea tag	<code><ww:textarea /></code>	renders a text area input field
Textfield tag	<code><ww:textfield /></code>	renders an input field of type text
Token tag	<code><ww:token /></code>	stop double-submission of forms
Internationalization Tags		
I18n tag	<code><ww:i18n /></code>	put resource bundle in stack
Text tag	<code><ww:text /></code>	renders string from bundle

Notes



The "required" attribute on many WebWork UI tags isn't defaults to true only if you have client side validation enabled and there is a validator associated with that particular field.



It's very important to note that all tags that insert something into the valuestack (like i18n or bean tags) will remove those objects from the stack on its end tag. This means that if you instantiate a bean with the bean tag (`<ww:bean name=""br.univap.fcc.sgpw.util.FormatterHelper"">` that bean will be available on the valuestack only until the `</ww:bean>` tag.

Checkbox tag

This page last changed on Jun 22, 2005 by [mnrsiat](#).

<ww:checkbox />

Renders an HTML input element of type checkbox, populated by the specified property from the OgnlValueStack.

Sample Usages

```
JSP
<ww:checkbox label="'checkbox test'" name="'checkboxField1'" value="aBoolean"
fieldValue="'true'"/>

Velocity
#tag( Checkbox "label='checkbox
test'" "name='checkboxField1'" "value=aBoolean" "fieldValue='true'" )

HTML (simple template, aBoolean == true)
<input type="checkbox" name="checkboxField1" value="true" checked="checked" />
```

Attributes

Name	Required	Description
id	no	HTML id attribute
name	yes	HTML name attribute
label	no	Text used as label in template
labelposition	no	Alignment of label (left,right,center)
required	no	Is field required for form submission
value	no	Boolean which if true adds 'checked="checked"' to tag
fieldValue	yes	the actual HTML value attribute of the checkbox
tabindex	no	HTML tabindex attribute

onchange	no	HTML onchange attribute
onclick	no	HTML onclick attribute
cssClass	no	HTML class attribute
cssStyle	no	HTML style attribute
theme	no	Theme to use
template	no	Name of template to use

Checkboxlist tag

This page last changed on Dec 14, 2004 by [casey](#).

<ww:checkboxlist />

Creates a series of checkboxes from a list. Setup is like <ww:select /> or <ww:radio />, but creates checkbox tags.

Attributes

Name	Required	Description
id	no	HTML id attribute
name	yes	HTML name attribute
value	no	Data to pass as field value
label	no	Text used as label in template
labelposition	no	Alignment of label (left,right,center)
required	no	Is field required for form submission
list	no	Iterable source to populate from
listKey	no	Property of list objects to get field value from
listValue	no	Property of list objects to get field content from
cssClass	no	HTML class attribute
cssStyle	no	HTML style attribute
theme	no	Theme to use
template	no	Name of template to use

Combobox tag

This page last changed on Jun 22, 2005 by [mnrsiat](#).

<ww:combobox />

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

In this example, the SELECT will be populated from id=year attribute. Counter is itself an Iterator. It will span from first to last. The population is done via javascript, and requires that this tag be surrounded by a <form>.

Note that unlike the <ww:select/> tag, there is no ability to define the individual <option> tags' id attribute or content separately. Each of these is simply populated from the toString() method of the list item. Presumably this is because the select box isn't intended to actually submit useful data, but to assist the user in filling out the text field.

```
JSP:
<ww:bean name="'webwork.util.Counter'" id="year"><ww:param name="'first'"
value="text('firstBirthYear')"/><ww:param name="'last'" value="2000"/><ww:combobox
label="'Birth year'" size="6" maxlength="4" name="'birthYear'"
list="#year"/></ww:bean>

Velocity:
#tag( ComboBox "label='Birth
year'" "size='6'" "maxlength='4'" "name='birthYear'" "list=#year" )
```

Attributes

Name	Required	Description
id	no	HTML id attribute
name	yes	HTML name attribute
value	no	Data to pass as field value
list	no	Iterable source to populate from. If this is missing, the select widget

		is simply not displayed.
size	no	HTML size attribute
maxlength	no	HTML maxlength attribute
disabled	no	HTML disabled attribute
tabindex	no	HTML tabindex attribute
onkeyup	no	HTML onkeyup attribute
onchange	no	HTML onchange attribute
onclick	no	HTML onclick attribute
cssClass	no	HTML class attribute
cssStyle	no	HTML style attribute
label	no	Text used as label in template
labelposition	no	Alignment of label (left,right,center)
theme	no	Theme to use
template	no	Name of template to use

Component tag

This page last changed on Apr 29, 2005 by [jcarreira](#).

<ww:component />

Renders an custom UI widget using the specified templates. Additional objects can be passed in to the template using the param tags. Objects provided can be retrieve from within the template via `$parameters._paramname_`.

In the bottom JSP and Velocity samples, two parameters are being passed in to the component. From within the component, they can be accessed as `$parameters.get('key1')` and `$parameters.get('key2')`. Velocity also allows you reference them as `$parameters.key1` and `$parameters.key2`.

Currently, your custom UI components can be written in Velocity, JSP, or Freemarker, and the correct rendering engine will be found based on file extension.

Remember: the value params will always be resolved against the OgnlValueStack so if you mean to pass a string literal to your component, make sure to wrap it in quotes i.e. `value="value1"` otherwise, the the value stack will search for an Object on the stack with a method of `getValue1()`. (now that i've written this, i'm not entirely sure this is the case. i should verify this manana)

Sample Usages

```
JSP
<ww:component template="/my/custom/component.vm"/>
or

<ww:component template="/my/custom/component.vm"><ww:param name="key1"
value="value1"/><ww:param name="key2" value="value2"/></ww:component>

Velocity
#tag( Component "template=/my/custom/component.vm" )

or

#bodytag( Component "template=/my/custom/component.vm" )
#param( "key1" "value1" )
#param( "key2" "value2" )
#end
```

Attributes

Name	Required	Description
id	no	HTML id attribute
name	yes	HTML name attribute
value	no	Data to pass as field value
label	no	Text used as label in template
labelposition	no	Alignment of label (left,right,center)
required	no	Is field required for form submission
cssClass	no	HTML class attribute
cssStyle	no	HTML style attribute
theme	no	Theme to use
template	yes	Name of template to use

File tag

This page last changed on Dec 14, 2004 by [casey](#).

<ww:file />

File upload form tag.

Sample Usages

```
<ww:file name="'uploadedFile'" />
```

Attributes

Name	Required	Description
id	no	HTML id attribute
name	yes	HTML name attribute
value	no	Data to pass as field value
label	no	Text used as label in template
labelposition	no	Alignment of label (left,right,center)
required	no	Is field required for form submission
accept	no	HTML accept attribute: list of file types to accept
onchange	no	HTML onchange attribute
onclick	no	HTML onclick attribute
cssClass	no	HTML class attribute
cssStyle	no	HTML style attribute
theme	no	Theme to use
template	no	Name of template to use

Form tag

This page last changed on Dec 14, 2004 by [casey](#).

<ww:form />

An HTML form.

Sample Usages

```
<ww:form name="myForm" action="submit.action">
...
</ww:form>
```

```
<ww:form name="myForm" action="submit" namespace="/foo">
...
</ww:form>
```

Attributes

Name	Required	Description	
name	yes	HTML name attribute	
value	no	Data to pass as field value	
openTemplate	no	Template to use for the open form tag	
action	yes	HTML action attribute	
namespace	no	Namespace of action	
target	no	HTML target attribute	
method	no	'GET' or 'POST'	
enctype	no	HTML enctype attribute	

validate	no	Whether to use client-side validation	
cssClass	no	HTML class attribute	
cssStyle	no	HTML style attribute	
theme	no	Theme to use	
template	no	Name of template to use for the closing form tag	

Note: The **action** attribute works in two different ways. The original (and deprecated) format is shown in the first sample usage. The newer format allows you to specify just the action name (as well as a namespace, if there is one) as defined in xwork.xml. This is **required** if you wish to do any form of [Client-Side Validation](#)

Hidden tag

This page last changed on Dec 14, 2004 by [casey](#).

<ww:hidden />

An HTML input tag of type "hidden".

Sample Usages

```
<ww:hidden name=" 'someName' " value="value"/>
```

Name	Required	Description
id	no	HTML id attribute
name	yes	HTML name attribute
value	no	Data to pass as text
cssClass	no	HTML class attribute
cssStyle	no	HTML style attribute
theme	no	Theme to use
template	no	Name of template to use

I18n tag

This page last changed on Dec 14, 2004 by [casey](#).

```
<ww:i18n />
```

Place a resource bundle on the value stack, for access by the text tag.

Attributes

Name	Required	Description
name	yes	Name of bundle

Sample Usages

-see [Text tag](#)

Label tag

This page last changed on Jan 13, 2005 by [mbogaert](#).

<ww:label />

An HTML LABEL that will allow you to output label:name combination that has the same format treatment as the rest of your UI controls.

Attributes

Name	Required	Description
id	no	HTML id attribute
name	no	HTML name attribute
for	no	HTML for attribute
value	no	Data to pass as text
label	no	Text used as label in template
labelposition	no	Alignment of label (left,right,center)
required	no	Is field required for form submission
cssClass	no	HTML class attribute
cssStyle	no	HTML style attribute
theme	no	Theme to use
template	no	Name of template to use

In this example, a label is rendered. The label is retrieved from a ResourceBundle by calling ActionSupport's `getText()` method giving you an output of User name: a label.

```
<ww:label label="text('user_name')" name="'a label'"/>
```

Password tag

This page last changed on Dec 14, 2004 by [casey](#).

<ww:password />

An HTML input tag of type password.

In this example, a password control is displayed. For the label, we are calling ActionSupport's `getText()` to retrieve password label from a resource bundle.

```
<ww:password label="text('password')" name="password" size="10" maxlength="15"/>
```

Name	Required	Description
id	no	HTML id attribute
name	yes	HTML name attribute
value	no	Data to pass as text
size	no	HTML size attribute
maxlength	no	HTML maxlength attribute
disabled	no	HTML disabled attribute
readonly	no	HTML readonly attribute
onkeyup	no	HTML onkeyup attribute
tabindex	no	HTML tabindex attribute
onchange	no	HTML onchange attribute
onclick	no	HTML onclick attribute
show	no	Redisplay value (security concerns)
label	no	Text used as label in template
labelposition	no	Alignment of label (left,right,center)
required	no	Is field required for form

		submission
cssClass	no	HTML class attribute
cssStyle	no	HTML style attribute
theme	no	Theme to use
template	no	Name of template to use

Radio tag

This page last changed on Dec 14, 2004 by [casey](#).

<ww:radio />

An HTML Radiobox UI widget

In this example, a radio control is displayed with a list of genders. The gender list is built from attribute id=genders. WW calls getGenders() which will return a Map. For examples using listKey and listValue attributes, see the section select tag.

```
<ww:action name=" 'GenderMap' " id="genders"/><ww:radio label=" 'Gender' " name=" 'male' "
list="#genders.genders" />
```

Attributes

Name	Required	Description
id	no	HTML id attribute
name	yes	HTML name attribute
value	no	Data to pass as field value
list	no	Iterable source to populate from
listKey	no	Property of list objects to get field value from
listValue	no	Property of list objects to get field content from
disabled	no	HTML disabled attribute
tabindex	no	HTML tabindex attribute
onchange	no	HTML onchange attribute
onclick	no	HTML onclick attribute
label	no	Text used as label in template
labelposition	no	Alignment of label

		(left,right,center)
theme	no	Theme to use
template	no	Name of template to use
required	no	Is field required for form submission

Select tag

This page last changed on Jul 09, 2005 by [mnrsiat](#).

<ww:select />

Generates a select list filled with a specified list. The "listKey" attribute is the property to pull from each item in the list to generate the value of the <option> tag for that item. The "listValue" attribute fills the label of the option (the display name). One great feature is that it will auto-select the appropriate option based on the "value" attribute. If the value matches the current listKey, that option will be selected (if the types match; see below).

```
<ww:select label=" 'Users' "  
  name=" 'userId' "  
  listKey="id"  
  listValue="name"  
  list="app.users"  
  value="app.user.id"  
  onchange=" 'chooseUser(this) ' "  
>
```

will create the following (if `getApp().getUser().getId() == 2`):

```
<tr><td>Users</td><td><select name="userId" onchange="chooseUser(this)"><option  
value="1">  
    User Number One  
</option><option value="2" selected="selected">  
    User Number Two  
</option></select></td></tr>
```

Of course, the <td> formatting and such depends on the [template](#) you are using.

Sample Usages

```
<ww:select label=" 'Pets' "  
  name=" 'petIds' "  
  list="petDao.pets"  
  listKey="id"  
  listValue="name"  
  multiple="true"  
  size="3"  
  required="true"  
>
```

```

<ww:select label="'Months'"
    name="'months'"
    list="#{'01':'Jan', '02':'Feb', [...]}"
    value="selectedMonth"
    required="true"
/>

// The month id (01, 02, ...) returned by the getSelectedMonth() call
// against the stack will be auto-selected

```

Note: For any of the tags that use lists (select probably being the most ubiquitous), which uses the OGNL list notation (see the "months" example above), it should be noted that the map key created (in the months example, the '01', '02', etc.) is typed. '1' is a char, '01' is a String, "1" is a String. This is important since if the value returned by your "value" attribute is NOT the same type as the key in the "list" attribute, they WILL NOT MATCH, even though their String values may be equivalent. If they don't match, nothing in your list will be auto-selected.

Attributes

Name	Required	Description
id	no	HTML id attribute
name	yes	HTML name attribute
value	no	Data to pass as field value
required	no	Is field required for form submission
list	no	Iterable source to populate from. If the list is a Map (key, value), the Map key will become the option "value" parameter and the Map value will become the option body.
listKey	no	Property of list objects to get field value from
listValue	no	Property of list objects to get field content from
emptyOption	no	Whether or not to add an

		empty (--) option after the header option
multiple	no	Creates a multiple select. The tag will pre-select multiple values if the values are passed as an Array (of appropriate types) via the value attribute. Passing a Collection may work too? Haven't tested this.
size	no	Size of the element box (# of elements to show)
disabled	no	HTML disabled attribute
tabindex	no	HTML tabindex attribute
onchange	no	HTML onchange attribute
onclick	no	HTML onclick attribute
headerKey	no	Key for first item in list
headerValue	no	Value for first item in list
label	no	Text used as label in template
labelposition	no	Alignment of label (left,right,center)
cssClass	no	HTML class attribute
cssStyle	no	HTML style attribute
theme	no	Theme to use
template	no	Name of template to use

Submit tag

This page last changed on Dec 14, 2004 by [casey](#).

<ww:submit />

Submit button.

Sample Usages

```
<ww:submit value="'Submit'"/>
```

Name	Required	Description
id	no	HTML id attribute
name	no	HTML name attribute
value	yes	Data to pass as text
align	no	HTML align attribute
label	no	Text used as label in template
labelposition	no	Alignment of label (left,right,center)
cssClass	no	HTML class attribute
cssStyle	no	HTML style attribute
theme	no	Theme to use
template	no	Name of template to use

Tabbedpane tag

This page last changed on Dec 14, 2004 by [casey](#).

<ww:tabbedpane />

Tabbed pane allows you to associated tabs with different views. When the user clicks the tab, that view will render and become current. Tabbed pane is basically a table which includes the selected tab's page in its bottom row.

In this example, a tabbed pane is rendered with tabs aligned right. See WW's example for a more complete picture.

```
<p><ww:tabbedpane id="tp1" contentName="'tabs1'" tabAlign="'RIGHT'"/></p>
```

Name	Required	Description
id	no	HTML id attribute
theme	no	Theme to use
contentName	yes	???
tabAlign	no	Tab horizontal alignment: RIGHT,LEFT,CENTER

Table tag

This page last changed on Dec 14, 2004 by [casey](#).

<ww:table />

HTML table using the displaytag library.

Sample Usages

```
<ww:table modelName="/result" sortable="true" sortColumn="0"
sortOrder="ASC"><ww:param name="'columnHidden(1)'" value="true"/><ww:param
name="'columnDisplayName(2)'" value="'New Display Name'"/><ww:param
name="'columnRenderer(0)'" value="#dateRenderer"/><ww:param
name="'columnRenderer(2)'" value="#linkRenderer"/><ww:param
name="'columnRenderer(4)'" value="#intRenderer"/></ww:table>
```

Name	Required	Description
theme	no	Theme to use
modelName	yes	???
sortable	no	Whether the columns are sortable
sortColumn	no	Index of initial sorted column
sortOrder	no	ASC,DESC,NONE

Text tag

This page last changed on Aug 15, 2005 by [digi9ten](#).

<ww:text />

Print out an internationalized string. It is used in conjunction with the `i18n` tag. The text tag gets a specific message from the bundle specified in the surrounding `i18n` tag. Values can be passed into the message for parsing, for instance to format a date or currency. If the text tag is not used in conjunction with a `i18n` tag, it will search through the class hierarchy. Please look at [Internationalization](#) for more details.

Attributes

Name	Required	Description
name	yes	Name of property to fetch
id	no	When specified, causes output to be stored in the <code>ApplicationContext</code> using the <code>id</code> as the key rather than printing out the text
value0	no	Pass data to param 0 in message
value1	no	Pass data to param 1 in message
value2	no	Pass data to param 2 in message
value3	no	Pass data to param 3 in message

Sample Usages

```
Accessing messages from a given bundle (the i18n Shop example bundle in this case)
<br><ww:i18n name="'webwork.action.test.i18n.Shop'"><ww:text
name="'main.title'"/></ww:i18n>
```

```
<ww:il8n id="foo" name="'webwork.action.test.il8n.Shop'"><ww:text  
name="'main.title'" /></ww:il8n><ww:property value="#foo" />
```

Note that instead of using value0..value4, you may also:

```
<ww:text name="'someKey'"><ww:param value="'Hello'" /></ww:text>
```

OR

```
<ww:text name="'someKey'"><ww:param>Hello</ww:param></ww:text>
```

The last format is particularly good when you are embedding HTML in to the message, since you don't need to worry about escaping the various quotes that might be there.

Textarea tag

This page last changed on Dec 14, 2004 by [casey](#).

<ww:textarea />

Renders a textarea tag.

Sample Usages

```
<ww:textarea label="'Comments'" name="'comments'" cols="30" rows="8"/>
```

Attributes

Name	Required	Description
id	no	HTML id attribute
name	yes	HTML name attribute
value	no	Data to pass as field value
required	no	Is field required for form submission
rows	no	HTML rows attribute
cols	no	HTML cols attribute
wrap	no	HTML wrap attribute
readonly	no	HTML readonly attribute
disabled	no	HTML disabled attribute
tabindex	no	HTML tabindex attribute
onkeyup	no	HTML onkeyup attribute
label	no	Text used as label in template
labelposition	no	Alignment of label (left,right,center)
cssClass	no	HTML class attribute

cssStyle	no	HTML style attribute
theme	no	Theme to use
template	no	Name of template to use

Textfield tag

This page last changed on Dec 14, 2004 by [casey](#).

<ww:textfield />

In this example, a text control is rendered. The label is retrieved from a ResourceBundle by calling ActionSupport's `getText()` method.

```
<ww:textfield label="text('user_name')" name="'user'"/>
```

Attributes

Name	Required	Description
id	no	HTML id attribute
name	yes	HTML name attribute
value	no	Data to pass as field value
required	no	Indicates if this field is required for form submission
size	no	HTML size attribute
maxlength	no	HTML maxlength attribute
readonly	no	HTML readonly attribute
disabled	no	HTML disabled attribute
tabindex	no	HTML tabindex attribute
onkeyup	no	HTML onkeyup attribute
onchange	no	HTML onchange attribute
onclick	no	HTML onclick attribute
label	no	Text used as label in template
labelposition	no	Alignment of label (left,right,center)

cssClass	no	HTML class attribute
cssStyle	no	HTML style attribute
theme	no	Theme to use
template	no	Name of template to use

Note that the default template name for this tag is `text.vm` and not `textfield.vm`.

Token tag

This page last changed on Jun 18, 2005 by [mnrsiat](#).

<ww:token />

The token tag is used to help with the "double click" submission problem. It is needed if you are using the [TokenInterceptor](#) or the [TokenSessionInterceptor](#). They are documented somewhere in the wiki. The ww:token tag merely places a hidden element that contains the unique token.

Sample Usages

```
<ww:token />
```

Attributes

Name	Required	Description
name	no	Name of token
theme	no	Theme to use
template	no	Name of template to use

TokenInterceptor

This page last changed on Jun 18, 2005 by [mnrsiat](#).

The TokenInterceptor, or TokenSessionInterceptor, is useful in preventing double submission of a form. The difference between the two is merely that the TokenSessionInterceptor places the double-submission data into the HttpSession, so that you may access it again if needed. If your application is content to simply discard this data as an error, just use the TokenInterceptor. It is never necessary to use both.

In order to use it on a particular action, you must do several things:

- include an interceptor-ref description on the action in xwork.xml that contains one of the two interceptors
- include a result mapping for the "invalid.token" result, which may be returned by the interceptor
- on each page that references the action, include a `<ww:token/>` tag (or the Velocity version)

Let's say you have an Action called DeleteImportantItemAction. You might use the following action configuration in your xwork.xml:

```
<action name="delete" class="com.example.action.DeleteImportantItemAction"><result
name="input" type="dispatcher">/WEB-INF/views/delete.jsp</result><result
name="success" type="chain">view-item</result><result name="invalid.token"
type="chain">view-item</result><interceptor-ref
name="defaultStack"/><interceptor-ref name="token"/></action>
```

You'll need to put a `<ww:token/>` tag on the page that initially calls your action, if it's not delete.jsp, as well as on delete.jsp itself. Otherwise, the Interceptor will short-circuit your action and instead, return the invalid.token result. Additionally, it will place an ActionError, "The form has already been processed or no token was supplied" on the Action.

A slightly less secure but simpler alternative to using tokens is to use a Redirect result:

```
<action name="delete" class="com.example.action.DeleteImportantItemAction"><result
name="input" type="dispatcher">/WEB-INF/views/delete.jsp</result><result
name="success" type="redirect">view-item.action</result></action>
```


If you use a workflow, with a middle-stage of confirming that the user wants to do the delete, you may wish to break that out into two actions. A token isn't needed for the confirm step, but there is no way to configure a token for some results and not others in the same action (except possibly by writing your own `TokenInterceptor`, but why bother when it already exists?). You can certainly do something like:

```
<action name="delete-confirm"
class="com.example.action.DeleteImportantItemAction"><result name="input"
type="dispatcher">/WEB-INF/views/delete.jsp</result><result name="success"
type="chain">view-item</result></action><action name="delete"
class="com.example.action.DeleteImportantItemAction"><result name="success"
type="chain">view-item</result><result name="invalid.token"
type="chain">view-item</result><interceptor-ref
name="defaultStack"/><interceptor-ref name="token"/></action>
```

Velocity Tags

This page last changed on Oct 06, 2005 by [digi9ten](#).

Velocity tags are extensions of the generic [Tags and UI Components](#) provided by WebWork. You can get started almost immediately by simply knowing the generic structure in which the tags can be accessed: **#wwxxx (...) ... #end**, where xxx is any of the tags supported by WebWork.



As of WebWork 2.2, Velocity support in WebWork has been deprecated. Many of the new 2.2 features were not built for Velocity, and moving forward Velocity support may be removed entirely. We highly recommend you look at FreeMarker.

Syntax

For example, in JSP you might create a form like so:

```
<ww:form action="updatePerson"><ww:textfield label="First name"
name="firstName"/><ww:submit value="Update"/></ww:form>
```

In Velocity the same form is built like so:

```
#wwform ("action=updatePerson")
  #wwtextfield ("label=First name" "name=firstName")
  #wwsubmit ("value=Update")
#end
```

Block and Inline Tags

You may notice that some tags require an `#end` statement, while others do not. Due to a limitation in Velocity, tags must declare if they are a *block* or *inline* tag up front.

As such, by default all tags are *inline* except for a few key ones, such as the [form tag](#). We **strongly** encourage you to look at FreeMarker, which provides much better flexibility in this area as well as others.

Velocity Tags - Old

This page last changed on Sep 12, 2005 by [plightbo](#).

[Note: this page is being actively worked on as of June 15 2005. I'll remove this notice when it's complete enough to be depended on.]

WebWork's Velocity tags correspond nearly exactly to the [JSP Tags](#). The syntax is, of course, very different, but the functionality is the same. This page is a reference for the Velocity syntax, but does not attempt to discuss the functionality in any detail. If you are unfamiliar with the tags, please read the [JSP Tags](#) page first for a solid discussion of the available features.

Much of this information is available elsewhere on the wiki, but it is all mixed in with the JSP discussion and not well labelled.

Name	Tag	Description
Checkbox tag	#tag(Checkbox)	render a checkbox input field
Checkboxlist tag	#tag(Checkboxlist)	render a list of checkboxes
Combobox tag	#tag(ComboBox)	widget that fills a text box from a select
Component tag	#tag(Component) OR #bodytag(Component) #end	render a custom ui widget
File tag	#tag(File)	renders a file select input field
Form tag	#bodytag(Form) #end	defines an input form
Hidden tag	#tag(Hidden)	render a hidden field
Label tag	#tag(Label)	render a label that displays read-only information
Password tag	#tag(Password)	render a password input field
Radio tag	#tag(Radio)	renders a radio button input field

Select tag	#tag(Select)	renders a select element
Submit tag	#tag(Submit)	renders a submit button
Table tag	#tag(Table) OR #bodytag(Table) #end	renders a table
Tabbedpane tag	#tag(Tabbedpane)	renders a tabbedpane
Textarea tag	#tag(Textarea)	renders a text area input field
Textfield tag	#tag(TextField)	renders an input field of type text
Token tag	#tag(Token)	stop double-submission of forms

You'll notice that some of the tags use a #tag(Name) syntax, and others use a #bodytag(Name) #end syntax. Tags that are parameterizable must use the second, which allows for body content. In general, the syntax is:

```
#bodytag( Name )
  #param( "name" "value" )
#end
```

WebWork has one of the most advanced type conversion abilities in any web-based framework in any Java language. Generally, you don't need to do anything to take advantage of it, other than name your HTML inputs (form elements and other GET/POST parameters) names that are valid [OGNL](#) expressions.

A Simple Example

Content pulled from external source. Click [here](#) to refresh.

Type conversion is great for situations where you need to turn a String in to a more complex object. Because the web is type-agnostic (everything is a string in HTTP), WebWork's type conversion features are very useful. For instance, if you were prompting a user to enter in coordinates in the form of a string (such as "3, 22"), you could have WebWork do the conversion both from String to Point and from Point to String.

Using this "point" example, if your action (or another compound object in which you are setting properties on) has a corresponding ClassName-conversion.properties file, WebWork will use the configured type converters for conversion to and from strings. So turning "3, 22" in to new Point(3, 22) is done by merely adding the following entry to **ClassName-conversion.properties** (Note that the PointConverter should impl the ognl.TypeConverter interface):

point = com.acme.PointConverter

Your type converter should be sure to check what class type it is being requested to convert. Because it is used for both to and from strings, you will need to split the conversion method in to two parts: one that turns Strings in to Points, and one that turns Points in to Strings.

After this is done, you can now reference your point (using <ww:property value="post"/> in JSP or \${point} in FreeMarker) and it will be printed as "3, 22" again. As such, if you submit this back to an action, it will be converted back to a

Point once again.

In some situations you may wish to apply a type converter globally. This can be done by editing the file **xwork-conversion.properties** in the root of your class path (typically WEB-INF/classes) and providing a property in the form of the class name of the object you wish to convert on the left hand side and the class name of the type converter on the right hand side. For example, providing a type converter for all Point objects would mean adding the following entry:

com.acme.Point = com.acme.PointConverter



Content pulled from external source.
Click [here](#) to refresh.

Type conversion should not be used as a substitute for i18n. It is not recommended to use this feature to print out properly formatted dates. Rather, you should use the i18n features of WebWork (and consult the JavaDocs for JDK's MessageFormat object) to see how a properly formatted date should be displayed.

WebWork ships with a helper base class that makes converting to and from Strings very easy. The class is **com.opensymphony.webwork.util.WebWorkTypeConverter**. This class makes it very easy for you to write type converters that handle converting objects to Strings as well as from Strings. From the JavaDocs for this class:

Content pulled from external source. Click [here](#) to refresh.

Base class for type converters used in WebWork. This class provides two abstract methods that are used to convert both to and from strings – the critical functionality that is core to WebWork's type conversion system.

Type converters do not have to use this class. It is merely a helper base class,

although it is recommended that you use this class as it provides the common type conversion contract required for all web-based type conversion.

Built in Type Conversion Support

Content pulled from external source. Click [here](#) to refresh.

WebWork will automatically handle the most common type conversion for you. This includes support for converting to and from Strings for each of the following:

- String
- boolean / Boolean
- char / Character
- int / Integer, float / Float, long / Long, double / Double
- dates - uses the SHORT format for the Locale associated with the current request
- arrays - assuming the individual strings can be converted to the individual items
- collections - if not object type can be determined, it is assumed to be a String and a new ArrayList is created

Note that with arrays the type conversion will defer to the type of the array elements and try to convert each item individually. As with any other type conversion, if the conversion can't be performed the standard type conversion error reporting is used to indicate a problem occurred while processing the type conversion.

Relationship to Parameter Names

The best way to take advantage of WebWork's type conversion is to utilize complete objects (ideally your domain objects directly), rather than submitting form values on to intermediate primitives and strings in your action and then converting those values to full objects in the execute() method. Some tips for achieving this are:

- Use complex OGNL expressions - WebWork will automatically take care of creating the actual objects for you.
- Use JavaBeans! WebWork can only create objects for you if your objects obey the JavaBean specification and provide no-arg constructions, as well as getters and

setters where appropriate.

- Remember that *person.name* will call **getPerson().setName()**, but if you are expecting WebWork to create the Person object for you, a **setPerson()** **must also exist**.
- For lists and maps, use index notation, such as *people[0].name* or *friends['patrick'].name*. Often these HTML form elements are being rendered inside a loop, so you can use the iterator tag's status attribute if you're using [JSP Tags](#) or the `${foo_index}` special property if you're using [FreeMarker Tags](#).
- For multiple select boxes, you obviously can't name each individual item using index notation. Instead, name your element simply *people.name* and WebWork will understand that it should create a new Person object for each selected item and set its name accordingly.

Advanced Type Conversion

WebWork also has some very advanced, yet easy-to-use, type conversion features. Null property handling will automatically create objects where null references are found. Collection and map support provides intelligent null handling and type conversion for Java Collections. Type conversion error handling provides an easy way to distinguish the difference between an input validation problem from an input type conversion problem.

Null Property Handling

Content pulled from external source. Click [here](#) to refresh.

Provided that the key `#CREATE_NULL_OBJECTS` is in the action context with a value of true (this key is set only during the execution of the `com.opensymphony.xwork.interceptor.ParametersInterceptor`), OGNL expressions that have caused a `NullPointerException` will be temporarily stopped for evaluation while the system automatically tries to solve the null references by automatically creating the object.

The following rules are used when handling null references:

- If the property is declared *exactly* as a Collection or List, then an `ArrayList` shall be returned and assigned to the null references.

- If the property is declared as a Map, then a HashMap will be returned and assigned to the null references.
- If the null property is a simple bean with a no-arg constructor, it will simply be created using the `{@link ObjectFactory#buildBean(Class)}` method.

Content pulled from external source. Click [here](#) to refresh.

For example, if a form element has a text field named **person.name** and the expression *person* evaluates to null, then this class will be invoked. Because the *person* expression evaluates to a *Person* class, a new Person is created and assigned to the null reference. Finally, the name is set on that object and the overall effect is that the system automatically created a Person object for you, set it by calling `setPerson()` and then finally called `getPerson().setName()` as you would typically expect.

Collection and Map Support

WebWork supports ways to determine the object type found in collections. This is done via an *ObjectTypeDetermine*. The default implementation is provided. The JavaDocs explain how map and collection support is determined:

Content pulled from external source. Click [here](#) to refresh.

This *ObjectTypeDeterminer* looks at the **Class-conversion.properties** for entries that indicated what objects are contained within Maps and Collections. For Collections, such as Lists, the element is specified using the pattern **Element_xxx**, where xxx is the field name of the collection property in your action or object. For Maps, both the key and the value may be specified by using the pattern **Key_xxx** and **Element_xxx**, respectively.

From WebWork 2.1.x, the **Collection_xxx** format is still supported and honored, although it is deprecated and will be removed eventually.

There is also an optional *ObjectTypeDeterminer* that utilizes Java 5 generics. See the [J2SE 5 Support](#) page for more information.

Type Conversion Error Handling

Content pulled from external source. Click [here](#) to refresh.

Any error that occurs during type conversion may or may not wish to be reported. For example, reporting that the input "abc" could not be converted to a number might be important. On the other hand, reporting that an empty string, "", cannot be converted to a number might not be important - especially in a web environment where it is hard to distinguish between a user not entering a value vs. entering a blank value.

By default, all conversion errors are reported using the generic i18n key **xwork.default.invalid.fieldvalue**, which you can override (the default text is *Invalid field value for field "xxx"*, where xxx is the field name) in your global i18n resource bundle.

However, sometimes you may wish to override this message on a per-field basis. You can do this by adding an i18n key associated with just your action (Action.properties) using the pattern **invalid.fieldvalue.xxx**, where xxx is the field name.

It is important to know that none of these errors are actually reported directly. Rather, they are added to a map called *conversionErrors* in the ActionContext. There are several ways this map can then be accessed and the errors can be reported accordingly.

There are two ways the error reporting can occur:

1. globally, using the [Conversion Error Interceptor](#)
2. on a per-field basis, using the Conversion Field Validator

By default, the conversion interceptor is included in [webwork-default.xml](#) in the default stack, so if you don't want conversion errors reporting globally, you'll need to change the interceptor stack and add additional validation rules.

Validation

This page last changed on Oct 16, 2005 by [roughley](#).

WebWork relies on [XWork's](#) validation framework to enable the application of input validation rules to your Actions before they are executed. This section only provides the bare minimum to get you started and focuses on WebWork's extension of the XWork validators to support client-side validation.

Please consult XWork's [validation framework documentation](#) for complete details.



There is also an option for AJAX based asynchronous validation, please see [Remote Form Validation](#) for more information.

Reference pages

1. [Simple validators](#)
2. [Visitor validation](#)
3. [Client-Side Validation](#)
4. [Validation Examples](#)

Registering Validators

Validation rules are handled by validators, which must be registered with the ValidatorFactory. The simplest way to do so is to add a file name **validators.xml** in the root of the classpath (/WEB-INF/classes) that declares all the validators you intend to use. The syntax of the file is as follows:

```
<validators>
  <validator name="required"
    class="com.opensymphony.webwork.validators.JavaScriptRequiredFieldValidator"/>
  <validator name="requiredstring"
    class="com.opensymphony.webwork.validators.JavaScriptRequiredStringValidator"/>
  <validator name="stringlength"
    class="com.opensymphony.xwork.validator.validators.StringLengthFieldValidator"/>
  <validator name="int"
    class="com.opensymphony.webwork.validators.JavaScriptIntRangeFieldValidator"/>
  <validator name="date"
    class="com.opensymphony.webwork.validators.JavaScriptDateRangeFieldValidator"/>
```

```

<validator name="expression"
  class="com.opensymphony.xwork.validator.validators.ExpressionValidator"/>
<validator name="fieldexpression"
  class="com.opensymphony.xwork.validator.validators.FieldExpressionValidator"/>
<validator name="email"
  class="com.opensymphony.webwork.validators.JavaScriptEmailValidator"/>
<validator name="url"
  class="com.opensymphony.webwork.validators.JavaScriptURLValidator"/>
<validator name="visitor"
  class="com.opensymphony.xwork.validator.validators.VisitorFieldValidator"/>
<validator name="conversion"
  class="com.opensymphony.xwork.validator.validators.ConversionErrorFieldValidator"/>
<validator name="regex"
  class="com.opensymphony.xwork.validator.validators.RegexFieldValidator"/>
</validators>

```

This list declares all the validators that comes with WebWork.

Turning on Validation

All that is required to enable validation for an Action is to put the ValidationInterceptor in the interceptor refs of the action (see [xwork.xml](#)) like so:

```

<interceptor name="validator"
  class="com.opensymphony.xwork.validator.ValidationInterceptor"/>

```

Note: The default **validationWorkflowStack** already includes this.

Defining Validation Rules

Validation rules can be specified:

1. Per Action class: in a file named ActionName-validation.xml
2. Per Action alias: in a file named ActionName-alias-validation.xml
3. Inheritance hierarchy and interfaces implemented by Action class: WebWork searches up the inheritance tree of the action to find default validations for parent classes of the Action and interfaces implemented

Here is an example for SimpleAction-validation.xml:

```
<!DOCTYPE validators PUBLIC "-//OpenSymphony Group//XWork Validator
1.0//EN" "http://www.opensymphony.com/xwork/xwork-validator-1.0.dtd">
<validators><field name="bar"><field-validator type="required"><message>You must
enter a value for bar.</message></field-validator><field-validator type="int"><param
name="min">6</param><param name="max">10</param><message>bar must be between ${min}
and ${max}, current value is ${bar}.</message></field-validator></field><field
name="bar2"><field-validator type="regex"><param
name="regex">[0-9],[0-9]</param><message>The value of bar2 must be in the format "x,
y", where x and y are between 0 and 9</message></field-validator></field><field
name="date"><field-validator type="date"><param name="min">12/22/2002</param><param
name="max">12/25/2002</param><message>The date must be between 12-22-2002 and
12-25-2002.</message></field-validator></field><field name="foo"><field-validator
type="int"><param name="min">0</param><param name="max">100</param><message
key="foo.range">Could not find
foo.range!</message></field-validator></field><validator type="expression"><param
name="expression">foo > bar</param><message>Foo must be greater than Bar. Foo =
${foo}, Bar = ${bar}.</message></validator></validators>
```

Here we can see the configuration of validators for the SimpleAction class. Validators (and field-validators) must have a **type** attribute, which refers to a name of an Validator registered with the ValidatorFactory as above. Validator elements may also have <param> elements with name and value attributes to set arbitrary parameters into the Validator instance. See below for discussion of the message element.

Each Validator or Field-Validator element must define one message element inside the validator element body. The message element has 1 attributes, key which is not required. The body of the message tag is taken as the default message which should be added to the Action if the validator fails.

Key gives a message key to look up in the Action's ResourceBundle using getText() from LocaleAware if the Action implements that interface (as ActionSupport does). This provides for Localized messages based on the Locale of the user making the request (or whatever Locale you've set into the LocaleAware Action).

After either retrieving the message from the ResourceBundle using the Key value, or using the Default message, the current Validator is pushed onto the ValueStack, then the message is parsed for \${...} sections which are replaced with the evaluated value of the string between the \${ and }. This allows you to parameterize your messages with values from the Validator, the Action, or both. Here is an example of a parameterized message:

```
bar must be between ${min} and ${max}, current value is ${bar}.
```

This will pull the min and max parameters from the IntRangeFieldValidator and the value of bar from the Action.

Client-Side Validation

This page last changed on Jul 22, 2004 by [unkyaku](#).

WebWork adds support for client-side validation on top of XWork's standard validation framework. You can enable it on a per-form basis by specifying **validate="true"** in the `<ww:form>` tag:

```
<ww:form name="'test'" action="'javascriptValidation'" validate="true">
    ...
</ww:form>
```

You must specify a *name* for the form in order for client-side validation.


You should also make sure you provide the correct *action* and *namespace* attributes to the `<ww:form>` tag. For example, if you have an Action named "submitProfile" in the "/user" namespace, you must use


```
<ww:form namespace="/user" action="'submitProfile'" validate="true">
    ...
</ww:form>
```


While the following will "work" in the sense that the form will function correctly, client-side validation will not:

```
<ww:form action="/user/submitProfile.action" validate="true">
    ...
</ww:form>
```

Of course, all the standard [validation configuration](#) steps still apply. Client-side validation uses the same validation rules as server-side validation. If server-side validation doesn't work, then client-side validation won't work either.

 Not all validators support client-side validation. Only validators that implement `ScriptValidationAware` support this feature. Refer to the list of WebWork validators to see which ones do so.

 Note that the *required* attribute on many WebWork [UI tags](#) has nothing to do with client-side validation.

 **Upgrade Alert:** This feature was introduced in WebWork 2.1. If upgrading from a previous version, make sure you are using the correct validators in [validators.xml](#). You *must* be using the **com.opensymphony.webwork.validators.JavaScriptRequired*Validator** version of the standard XWork validators.

Building a Validator that supports client-side validation

Any validator can be extended to support client-side validation by implementing the **com.opensymphony.webwork.validators.ScriptValidationAware** interface:

```
public interface ScriptValidationAware extends FieldValidator {
    public String validationScript(Map parameters);
}
```

The value returned by **validationScript** will be executed on the client-side before the form is submitted if client-side validation is enabled. For example, the **requiredstring** validator has the following code:

```
public String validationScript(Map parameters) {
    String field = (String) parameters.get("name");
    StringBuffer js = new StringBuffer();

    js.append("value = form.elements['" + field + "'].value;\n");
    js.append("if (value == \"\") {\n");
    js.append("\talert('\" + getMessage(null) + '\");\n");
    js.append("\treturn '\" + field + '\";\n");
    js.append("}\n");
    js.append("\n");

    return js.toString();
}
```

Only JavaScript is supported at this time.

Simple validators

This page last changed on Sep 23, 2005 by [jhouse](#).

The following validators are included in the default validators.xml:

Name	JavaScript aware	Description
required		Field value must have a value (non-null)
requiredstring	x	Field value is non-null and has a length > 0
regex		If not empty, field value must match a regular expression
int	x	Field value must be an integer and within a range
date		Field value must be a date (the format is based on locale) and within a range
expression		A given OGNL expression is evaluated against the value stack and must return true. This is mostly usefully for cross-field validation. Errors are added as action errors
fieldexpression		A given OGNL expression is evaluated against the value stack and must return true. This is similar to expression but errors are added as field errors
email	x	Field value must be a valid e-mail address
url	x	Field value must be a valid url

visitor		Allows you to forward validation to object properties of your action using the objects own validation files
conversion		Add conversion errors from ActionContext to field errors of the action. This does the same thing as WebWorkConversionErrorInterceptor

Note: the above name can be changed if you supply your own validators.xml.

required

In SimpleAction-validation.xml:

```
<validators><field name="bar"><field-validator type="required"><message>You must enter a value for bar.</message></field-validator></field></validators>
```

[top](#)

requiredstring

In LoginAction-validation.xml:

```
<validators><field name="userName"><field-validator type="requiredstring"><message>You must enter an username.</message></field-validator></field></validators>
```

The error is shown if request parameter **userName** is missing or an empty string

[top](#)

regex

```
<validators><field name="phone"><field-validator type="regex"><param
name="regex">\([\d][\d][\d]\) [\d][\d][\d]-[\d][\d][\d][\d]</param><message>Phone
number must be in the format (XXX)
XXX-XXXX</message></field-validator></field></validators>
```

[top](#)

int

```
<validators><field name="foo"><field-validator type="int"><param
name="min">0</param><param name="max">100</param><message key="foo.range">Could not
find foo.range!</message></field-validator></field></validators>
```

[top](#)

date

```
<validators><field name="startDate"><field-validator type="date"><param
name="min">12/22/2002</param><param name="max">12/25/2002</param><message>The date
must be between 12-22-2002 and
12-25-2002.</message></field-validator></field></validators>
```

[top](#)

expression

```
<validators><validator type="expression"><param name="expression">foo >
bar</param><message>Foo must be greater than Bar. Foo = ${foo}, Bar =
${bar}.</message></validator></validators>
```

The validator is not associated with a single field. You may need to place your expression within a CDATA if it contains bad xml characters.

[top](#)

fieldexpression

```
<validators><field name="productCode"><field-validator type="fieldexpression"><param
name="expression">name.length() == 5</param><message>Product code must be 5
characters, it is currently
'${productCode}'</message></field-validator></field></validators>
```

[top](#)

email

```
<validators><field name="email"><field-validator type="email"><message>You must
enter a valid email address.</message></field-validator></field></validators>
```

The address must be in the format xxx@yyy.com|net|gov|org|edu|info|mil|biz|tv|...

[top](#)

url

```
<validators><field name="companyUrl"><field-validator type="url"><message>You must
enter a valid URL.</message></field-validator></field></validators>
```

[top](#)

Validation Examples

This page last changed on Sep 23, 2005 by [jhouse](#).

Included in the [WW:WebWork](#) example war file is an example of using the [XW:Validation Framework](#) in WebWork2. This example consists of three links which all use the same Action Class and view pages (Velocity).

The sources

First, I had to add the **validators.xml** file to the root of the source tree for the example app

```
<validators><validator name="required"
class="com.opensymphony.xwork.validator.validators.RequiredFieldValidator"/><validator
name="requiredstring"
class="com.opensymphony.xwork.validator.validators.RequiredStringValidator"/><validator
name="int"
class="com.opensymphony.xwork.validator.validators.IntRangeFieldValidator"/><validator
name="date"
class="com.opensymphony.xwork.validator.validators.DateRangeFieldValidator"/><validator
name="expression"
class="com.opensymphony.xwork.validator.validators.ExpressionValidator"/><validator
name="fieldexpression"
class="com.opensymphony.xwork.validator.validators.FieldExpressionValidator"/><validator
name="email"
class="com.opensymphony.xwork.validator.validators.EmailValidator"/><validator
name="url"
class="com.opensymphony.xwork.validator.validators.URLValidator"/><validator
name="visitor"
class="com.opensymphony.xwork.validator.validators.VisitorFieldValidator"/><validator
name="regex"
class="com.opensymphony.xwork.validator.validators.RegexFieldValidator"/></validators>
```

The Action class used by all of the validation examples is **ValidatedAction**

```
package com.opensymphony.webwork.example;

import com.opensymphony.xwork.ActionSupport;

/**
 * ValidatedAction
 * @author Jason Carreira
 * Created Sep 12, 2003 9:23:38 PM
 */
public class ValidatedAction extends ActionSupport {
    private ValidatedBean bean = new ValidatedBean();
    private String name;
    private String validationAction = "basicValidation.action";
```

```

public ValidatedBean getBean() {
    return bean;
}

public void setBean(ValidatedBean bean) {
    this.bean = bean;
}

public String getName() {
    return name;
}

public void setName(String name) {
    this.name = name;
}

public String getValidationAction() {
    return validationAction;
}

public void setValidationAction(String validationAction) {
    this.validationAction = validationAction;
}
}

```

The base validation file for the ValidatedAction is **ValidatedAction-validation.xml**

```

<!DOCTYPE validators PUBLIC "-//OpenSymphony Group//XWork Validator
1.0//EN" "http://www.opensymphony.com/xwork/xwork-validator-1.0.dtd"><validators><field
name="name"><field-validator type="requiredstring"><message>You must enter a
name.</message></field-validator></field></validators>

```

This and all other validation files are placed in the same package as the classes to which they apply.

The form for this Action is **validationForm.vm**

```

<html><head><title>Webwork Validation Example</title></head><body>
#if( $actionErrors.size() > 0 )
<p><font color="red"><b>ERRORS:</b><br><ul>
#foreach( $error in $actionErrors )
<li>$error</li>
#end
</ul></font></p>
#end
<p><form name="myForm" action="${validationAction}" method="POST"><input
type="hidden" name="validationAction" value="${validationAction}"/>
Action Properties:
<br><table>
#tag( TextField "label=Name" "name=name" "value=name" )
</table>
Bean Properties:
#if( $stack.findValue("fieldErrors") )

```

```

    #set( $beanErrors = $stack.findValue("fieldErrors.get('bean')") )
    #if( $beanErrors.size() > 0 )
    <br><font color="red"><b>Bean Errors:</b><br><ul>
    #foreach( $beanError in $beanErrors )
    <li>$beanError</li>
    #end
    </ul></font>
    #end
#end
<table>
#tag( TextField "label=Bean.Text" "name=bean.text" "value=bean.text" )<br>
#tag( TextField "label=Bean.Date" "name=bean.date" "value=bean.date" )<br>
#tag( TextField "label=Bean.Number" "name=bean.number" "value=bean.number" )<br>
#tag( TextField "label=Bean.Number2" "name=bean.number2" "value=bean.number2"
)<br></table><input type="submit" value="Test Validation"/></form></body>

```

The success page for these examples is a very simple page, **valid.vm**

```

<html><head><title>WebWork Validation Test: Valid</title></head><body>
Input was valid!
</body></html>

```

We'll look at any other example-specific configuration files as we get to them.

Basic Validation

The BasicValidation example is defined in the example **xwork.xml** file like this

```

<action name="basicValidation"
class="com.opensymphony.webwork.example.ValidatedAction"><interceptor-ref
name="validationWorkflowStack"/><result name="success"
type="dispatcher">valid.vm</result><result name="input"
type="dispatcher">validationForm.vm</result><result name="error"
type="dispatcher">validationForm.vm</result></action>

```

The **interceptor-ref** here, to **"validationWorkflowStack"**, is defined in `webwork-default.xml` and provides the parameter interceptors as well as the `ValidationInterceptor` (see [XW:Validation Framework](#) and the `DefaultWorkflowInterceptor` (see [XW:Interceptors#DefaultWorkflow](#)). All of the parameters from the configuration file (there are none in this case) followed by the parameters from the request will be set onto the Action. Next, the validations will be run, and finally the `DefaultWorkflow` will be applied (see [XW:Interceptors#DefaultWorkflow](#)).

This example is very simple, and the ValidatedAction-validation.xml file is the only set of Validations which will be applied. This means that the only validation done is that you enter some text for the name field.

Visitor Validation Example

note: check out another [Visitor Field Validator Example](#) .

The **ValidatedAction** holds a reference to a plain Java bean, **ValidatedBean**:

```
package com.opensymphony.webwork.example;

import java.util.Date;

/**
 * ValidatedBean
 * @author Jason Carreira
 * Created Sep 12, 2003 9:24:18 PM
 */
public class ValidatedBean {
    privateString text;
    private Date date = new Date(System.currentTimeMillis());
    privateint number;
    privateint number2;
    publicstaticfinalint MAX_TOTAL = 12;

    publicString getText() {
        return text;
    }

    public void setText(String text) {
        this.text = text;
    }

    public Date getDate() {
        return date;
    }

    public void setDate(Date date) {
        this.date = date;
    }

    publicint getNumber() {
        return number;
    }

    public void setNumber(int number) {
        this.number = number;
    }

    publicint getNumber2() {
```

```

        return number2;
    }

    public void setNumber2(int number2) {
        this.number2 = number2;
    }
}

```

The base validation file for the ValidatedBean is **ValidatedBean-validation.xml**

```

<!DOCTYPE validators PUBLIC "-//OpenSymphony Group//XWork Validator
1.0//EN" "http://www.opensymphony.com/xwork/xwork-validator-1.0.dtd"><validators><field
name="text"><field-validator type="requiredstring"><message key="invalid.text">Empty
Text!</message></field-validator></field><field name="date"><field-validator
type="date"><param name="min">01/01/1970</param><message key="invalid.date">Invalid
Date!</message></field-validator></field><field name="number"><field-validator
type="int"><param name="min">1</param><param name="max">10</param><message
key="invalid.number">Invalid
Number!</message></field-validator></field></validators>

```

In the Visitor Validation Example, we add a **VisitorFieldValidator** to apply these validations to our **ValidatedBean**. The Action is defined in our **xwork.xml** file like this:

```

<action name="visitorValidation"
class="com.opensymphony.webwork.example.ValidatedAction"><interceptor-ref
name="validationWorkflowStack"/><param
name="validationAction">visitorValidation.action</param><result
name="success">valid.vm</result><result
name="input">validationForm.vm</result><result
name="error">validationForm.vm</result></action>

```

Here we see a slight difference from the basic validation example above. I've added a static param to the Action which will be applied to the Action by the static-param interceptor. This parameter only sets the value for the action to post the form to for validation (see **validationForm.vm**).

The Action name above, *visitorValidation* is mapped to another set of validations defined in the file **ValidatedAction-visitorValidation-validation.xml**

```

<!DOCTYPE validators PUBLIC "-//OpenSymphony Group//XWork Validator
1.0//EN" "http://www.opensymphony.com/xwork/xwork-validator-1.0.dtd"><validators><field
name="bean"><field-validator type="required"><message>The bean must not be
null.</message></field-validator><field-validator type="visitor"><message>bean:
</message></field-validator></field></validators>

```

This file is found automatically by the validation framework based on the class name (**ValidatedAction**), the action alias, which is used as the validation context (**visitorValidation**) and the standard suffix (**-validation.xml**) to form the filename **ValidatedAction-visitorValidation-validation.xml**.

This file defines two validators for the "bean" field, a required validator which makes sure the bean is not null, and a **VisitorFieldValidator**. The **VisitorFieldValidator** will apply the validators for the **ValidatedBean** using the same validation context as is used in validating **ValidatedAction**, **visitorValidation**. It therefore looks for the validation files **ValidatedBean-validation.xml** (the default validations for the **ValidatedBean**) and **ValidatedBean-visitorValidation-validation.xml** (the validations specific to this validation context) , in that order.

The **ValidatedBean-validation.xml** looks like this:

```
<!DOCTYPE validators PUBLIC "-//OpenSymphony Group//XWork Validator
1.0//EN" "http://www.opensymphony.com/xwork/xwork-validator-1.0.dtd"><validators><field
name="text"><field-validator type="requiredstring"><message key="invalid.text">Empty
Text!</message></field-validator></field><field name="date"><field-validator
type="date"><param name="min">01/01/1970</param><message key="invalid.date">Invalid
Date!</message></field-validator></field><field name="number"><field-validator
type="int"><param name="min">1</param><param name="max">10</param><message
key="invalid.number">Invalid
Number!</message></field-validator></field></validators>
```

This file applies validations for three fields (text, date, and number) and gives message keys and default messages for each of them. These message keys will be used to look up messages from a resource bundle specific to the **ValidatedBean** class. In the same package as the **ValidatedBean** is a file named **ValidatedBean.properties**

```
invalid.date=You must enter a date after ${min}.
invalid.text=You must enter some text.
invalid.number=You must enter a number between ${min} and ${max}.
invalid.total=The total of number and number2 must be less than
${@com.opensymphony.webwork.example.ValidatedBean@MAX_TOTAL}.
```

These messages will be used for any errors added for the **ValidatedBean** using a message key. As you can see from the body of the messages, they can be parameterized with properties from the Bean, the **Interceptor**, and the **Action** (and they will be searched in that order). There is also an example of using a **Static** field

`${@com.opensymphony.webwork.example.ValidatedBean@MAX_TOTAL}`.

The **ValidatedBean-visitorValidation-validation.xml** file would define validations specific for the *visitorValidation* validation context, but it is not there, so it is ignored.

Visitor Validation with the Expression Validator

The final example shows a similar setup to the previous visitor validation example. The **xwork.xml** configuration for this example is very similar to the **visitorValidation** example:

```
<action name="expressionValidation"
class="com.opensymphony.webwork.example.ValidatedAction"><interceptor-ref
name="validationWorkflowStack"/><param
name="validationAction">expressionValidation.action</param><result
name="success">valid.vm</result><result
name="input">validationForm.vm</result><result
name="error">validationForm.vm</result></action>
```

The **ValidatedAction-expressionValidation-validation.xml** file defines the validations specific to this validation context:

```
<!DOCTYPE validators PUBLIC "-//OpenSymphony Group//XWork Validator
1.0//EN" "http://www.opensymphony.com/xwork/xwork-validator-1.0.dtd"><validators><field
name="bean"><field-validator type="required"><message>The bean must not be
null.</message></field-validator><field-validator type="visitor"><param
name="context">expression</param><message>bean:
</message></field-validator></field></validators>
```

This is almost identical to the **ValidatedAction-visitorValidation-validation.xml** file, but shows an example of passing a context param to the **VisitorFieldValidator**. In this case, rather than using the same validation context as is used for the **ValidatedAction** (*expressionValidation*), it passes another context (*expression*) to be used instead.

In this case, the validation context specific validations for the **ValidatedBean** is present, and it's named **ValidatedBean-expression-validation.xml**

```
<!DOCTYPE validators PUBLIC "-//OpenSymphony Group//XWork Validator
1.0//EN" "http://www.opensymphony.com/xwork/xwork-validator-1.0.dtd"><validators><validator
type="expression"><param
name="expression">@com.opensymphony.webwork.example.ValidatedBean@MAX_TOTAL >
(number + number2)</param><message key="invalid.total">Invalid
total!</message></validator></validators>
```

This adds an object-level (as opposed to field-level) ExpressionValidator which checks the total of the number and number2 fields against a static constant, and adds an error message if the total is more than the constant.

A note about error messages with the VisitorFieldValidator

With the VisitorFieldValidator, message field names are appended with the field name of the field in the Action. In this case, the fields *text*, *date*, and *number* in the **ValidatedBean** would be added as field error messages to the Action with field names *bean.text*, *bean.date*, and *bean.number*. The error messages added for the object-level ExpressionValidator applied in the last example will be added as field-level errors to the Action with the name *bean*.

VisitorFieldValidatorExample

This page last changed on Nov 29, 2004 by [jcarreira](#).

I've been using the validator in webwork and have found myself duplicating a few of the validations throughout the app, so after asking a few questions I was told what to do.

basically I have a creation form to create a Car, and then I have another to Edit and Update the Car, I wanted to create just one validation file that would validate that Car, I was using 2 validation files, CreateCarAction-validation.xml and UpdateCarAction-validation.xml and both were identical.

what I had to do was:

first i had to expose the Car object in my action, then I had to create a validation file for the action, and placed this file in the same package as the CarAction in this example;

CarAction-validation.xml

```
<!DOCTYPE validators PUBLIC "-//OpenSymphony Group//XWork Validator
1.0//EN" "http://www.opensymphony.com/xwork/xwork-validator-1.0.dtd">
<validators>
  <field name="car">
    <field-validator type="visitor">
      <message>Car Visitor: </message>
    </field-validator>
  </field>
</validators>
```

next I had to create Car-validation.xml file and put this file in the same package as the Car class

Car-validation.xml

```
<!DOCTYPE validators PUBLIC "-//OpenSymphony Group//XWork Validator
1.0//EN" "http://www.opensymphony.com/xwork/xwork-validator-1.0.dtd">
<validators>

  <field name="name">
    <field-validator type="requiredstring">
      <message>Name is REQUIRED</message>
```

```

        </field-validator>
    </field>
    <field name="model">
        <field-validator type="requiredstring">
            <message>Model is REQUIRED</message>
        </field-validator>
    </field>
</validators>

```

so now if the input strings car.model and car.name are empty you should get fieldErrors added to the car.name and car.model fields, you just need to make sure that in your action you expose the Car object.

heres the CarAction.java file:

```

public class CarAction extends ActionSupport {

    private Car car = new Car();

    public Car getCar() {
        return car;
    }

    public void setCar(Car car) {
        this.car = car;
    }

    publicString execute() {
        return SUCCESS;
    }

}

```

heres my Car bean:

```

public class Car {
    privateString name;
    privateString model;

    public Car() {}

    publicString getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    publicInteger getId() {
        return id;
    }
}

```

```

    }

    public void setId(Integer id) {
        this.id = id;
    }

    public String getModel() {
        return model;
    }

    public void setModel(String model) {
        this.model = model;
    }
}

```

and then the car.vm file:

```

<form method="get">

<table border="1" align="center">

    #tag( TextField "label=Car ID" "name=car.id" "value=car.id" )
    #tag( TextField "label=Car Name" "name=car.name" "value=car.name" )
    #tag( TextField "label=Car Model" "name=car.model" "value=car.model" )

    #tag( Submit "value='Submit Car'" "align=center" )

</table>

</form>

```


Visitor validation

This page last changed on Jul 10, 2004 by [unkyaku](#).

The VisitorFieldValidator allows you to forward validation to object properties of your action using the objects own validation files. This allows you to use the ModelDriven development pattern and manage your validations for your models in one place, where they belong, next to your model classes. The VisitorFieldValidator can handle either simple Object properties, Collections of Objects, or Arrays. See [XW:Standard Validators#VisitorFieldValidator](#)

Velocity

This page last changed on Aug 30, 2005 by [plightbo](#).

TODO: General document describing how Velocity works with WebWork. It should cover:

- What result type to use (velocity)
- How to configure velocity (see [velocity.properties](#))
- What tags are available (refer to the general tag documentation, don't re-document them here)
- Items available in the velocity context
- Compatibility issues (see [web.xml 2.1.x compatibility](#))
- General tips and tricks w/ Velocity (such as utilizing VelocityManager)
- Common pitfalls

Related Projects

This page last changed on Oct 30, 2005 by [plightbo](#).

1. [EclipseWork](#)
2. [IDEA Plugin](#)
3. [Struts Ti](#)

EclipseWork

This page last changed on Oct 30, 2005 by [plightbo](#).

EclipseWork is a third-party project created by Ricardo Lecheta. It is an Eclipse plugin for WebWork. Future plans include abstracting the plugin and providing for a plugin base that can be used to build both the Eclipse plugin and a new [IDEA Plugin](#). For more information, check out the [EclipseWork home page](#).



This document is an MRD (Marketing Requirements Document) for the ideal vision of what the WebWork IDEA Plugin will be. At this point, no WebWork IDEA plugin has been created. When it is created, it will be based on [EclipseWork](#).

A key success for WebWork is its ability to provide tools that make developers lives easier. This isn't just some configuration editor, but rather a complex integrated environment where developers can write their code, build JSPs, and have much of the redundant work related to WebWork automated or assisted. WebWork's IDEA integration enables developers to not only write WebWork-based applications faster, but also with much higher quality.

The IDEA Plugin has the following features:

- **Configuration browser** - you can browse namespaces and see both actions and views as well as their relationships. The browser lets you avoid hunting through xwork.xml and all the included files and instead presents a logical view of all your actions and their associated views. The browser also allows for very quick access to an action if you know it by name (and optional namespace).
- **Find by usages** - finding usages on a field will show you where all the views that use that field are located.
- **Refactoring support** - similar to find by usages, it is very useful to rename a field in an action (or even an object that is someone in the stack/object graph) and know that all the views that reference it have been updated as well
- **Internationalization support** - you can select text in a JSP or other views and automatically extract it into the a choice of the correct resource bundles (package.properties, Action.properties, etc). Conversely, you can press Ctrl-Q on `<ww:text/>` tags to see what the actual text for a particular i18n key are.
- **Error reporting** - the plugin is excellent at notifying you of errors before you deploy your application. Since many aspects of WebWork are based around loosely coupled and dynamic interactions, the plugin helps you see when you've had a typo such as "document" instead of "document". It highlights errors in OGNL expressions, i18n locations, and configuration mixups automatically for you. When these errors happen the plugin also gives you a choice of possible fixes (Ctrl-Enter).
- **Code completion** - when using WebWork tags such as `<ww:select value="..."/>`

pressing Ctrl-Space pops up a code completion dialog that introspects through all the available properties that are in the value stack, including those in your action.

- **OGNL expression evaluator** - When you're editing a page you should be able to have an OGNL expression evaluator that can execute your action and use its properties to evaluate OGNL expressions to show you what you get.

Struts Ti

This page last changed on Oct 30, 2005 by [plightbo](#).

Struts Ti is a new version of Struts created by the Struts developers to succeed Struts 1.x. It is based on WebWork and aims to make web development as simple as possible by taking advantage of the newest Java 5 features. You can find out more at the [Struts Ti homepage](#).

This page contains my suggestions on documentation style. I will *try* to demonstrate my suggestions by writing a document that justifies them. I'm an advocate of **learning by example**. As Mark Twain said, "*Few things are harder to put up with than the annoyance of a good example*" ([Samuel Langhorne Clemens \(1835-1910\)](#)).

Pulling content from CVS

A large part of the WebWork 2.2 documentation effort is to make documentation easier to handle in the future. At on the onset of this effort, everyone agreed that pulling as much source code, examples, and documentation from source control would help greatly. As such, we've installed a modified version of the **snippet macro** in the OpenSymphony wiki.



Important!

Whenever you write documentation, ask yourself if you can somehow have this documentation checked in to source control in the form of example code or JavaDocs. This will make the documentation much more likely to be useful for years to come.

The standard way to use it is to do the following:

```
{snippet:id=description|javadoc=true|url=com.opensymphony.xwork.interceptor.LoggingInterce
```

or

```
{snippet:id=example|javadoc=true|lang=xml|url=com.opensymphony.xwork.interceptor.Logging
```

or


```
{snippet:id=sitegraph-usage|lang=none|url=webwork/src/java/com/opensymphony/webwork/si
```

or

```
{snippet:id=ajax-validation-example|lang=xml|url=webwork/webapps/ajax/src/webapp/lesson1.
```

Where:

- id is the *name* of the snippet (*required).
- url is the URL where the snippet can be found (**required**).
- lang is the language that the code block should be required as. If this snippet is simply text, don't include this parameter and the content will be printed outside of a code block.
- javadoc indicates if the content is within a JavaDoc block. If this is set to true, then the preceeding "* " (asterisk-space) characters will be stripped before printing the content out. Also, the content is assumed to be HTML escaped already and therefore won't be escaped again.

All snippets are marked off by the pattern **START SNIPPET: XXX** and **END SNIPPET: XXX** where **XXX** is the *name* of the snippet that is assigned in the id attribute of the macro. The URL is typically a location that points to the project's source control contents.

A note about URLs

As you probably noticed in the examples, there are several formats that the URL patterns can be in. A fully-qualified URL is always allowed, though that is often not practical. We've customized the macro to be a bit more intelligent with the URL attribute:

- If the URL appears to be a class, we assume it lives in *src/java*, convert all the dots to slashes, and then append *.java* to it.
- If the URL doesn't start with "http", then it is assumed to start with https://opensymphony.dev.java.net/source/browse/*checkout*/, as you saw in the third example.

Note that the short-hand class notation will only work for top-level projects

(WebWork, OSWorkflow, etc) and not any sub-projects within the projects, such as example webapps in WebWork. If you wish to include content from a class in a sub-project, you'll need to list out the full path, like in the fourth example.

A note about snippet markers

All snippet markers should be commented out if possible. How they are commented out depends on where the snippet is. If the snippet is for HTML or XML, you can do:

```
<!-- START SNIPPET: xxx -->
...
<!-- END SNIPPET: xxx -->
```

If the snippet is in Java code, you can do:

```
if (true != false) {
    // START SNIPPET: xxx
    System.out.println("This is some silly code!");
    // END SNIPPET: xxx
}
```

If the snippet is found in JavaDocs, you should use HTML comments as they won't render in the JavaDocs. For XML examples in JavaDocs (see [Timer Interceptor](#) for an example), it can be a bit tricky. This is because in your JavaDocs you want to use the `<pre>` tag, but you don't want the wiki to display it. A good technique is to embed the snippet markers *inside* the `<pre>` tag:

```
* <pre>
* <!-- START SNIPPET: example -->
* &lt;!-- records only the action's execution time --&gt;
* &lt;action name="someAction" class="com.examples.SomeAction"&gt;
*     &lt;interceptor-ref name="completeStack"/&gt;
*     &lt;interceptor-ref name="timer"/&gt;
*     &lt;result name="success"&gt;good_result.ftl&lt;/result&gt;
* &lt;/action&gt;
* <!-- END SNIPPET: example -->
```

About Headings

★ This section refers to: [Notation Guide >> Headings](#).

Headings should definitely be used. This section tries to justify why.

First rule: don't use "h1" at the top of each page. The page title serves as the "top level header" already, so there is no need to duplicate that information again. Also, when the docs end up on the website, SiteMesh will place a top level "h1" element using the page title.

Document sections

Headings can help you divide your document into sections, subsections, sub-subsections and so forth.

Advantages

Your document becomes more organized.

Disadvantages

If you exaggerate you could fragment your text too much.



Warning

This is definitely an example of this, since this whole "Headings" section has such few paragraphs that it really should have been written in just one section.

Aren't warning boxes neat?

Headings capitalization

I think headers `h1` and `h2` should have all words capitalized (such as "Vitor's Suggestions on Documentation Style" and "About Headings"), but `h3` and smaller

would have just the first word (such as "Headings capitalization"). Except, of course, for words that are always capitalized (eg. "Understanding WebWork's importance to OpenSymphony and its community"). This gives even more importance to bigger headings.

Avoid skipping headers

I mean, avoid going from a `h1` directly to a `h3` without using `h2` before. This would be like havin a section 1.1.1 directly below section 1, without the existence of section 1.1.



Handy Hint

One thing that I like to do is leave **five** blank lines before `h1` headings, **three** before `h2`'s and **two** before `h3`'s. Also, in Portuguese (I'm Brazilian), we write small numbers using their names instead of numeric representation (**five** instead of **5**). I don't know if this is also a good practice in written english.

Aren't tip boxes neat?



Be Careful

If you find yourself writting too many `h1` headings in a single page, consider breaking the page into child pages and linking to them.


Aren't note boxes neat?

More on Text Effects

★ This section refers to: [Notation Guide >> Text Effects](#).

Text effects should be largely used, although I have some questions on some of them. **Strong**, *emphasis*, and inserted can be used to denote importante parts of a sentence. But I really think inserted should have been called underline in [the notation guide](#). I don't see the point of using ~~deleted~~, since when someone changes a page and deletes stuff, [Confluence](#) keeps the old versions in history.

I can't think of a situation in WebWork's doc for superscript and subscript, but it doesn't hurt to mention them. I can't say anything about %span% because I frankly don't know what it does. Monospaced is heavily used, for instance, to refer to webwork-default.xml file or items in source code examples: <xmltag />, JavaClass or javaVariable.

	<p>Boxes vs. Block Quotes</p> <p>I think boxes and block quotes do the same job, but boxes are better. Therefore, I suggest we don't use block quotes.</p> <p><i>Aren't info boxes neat? Aren't them all neat? By now you may have realized I think we should definetly use them...</i></p>
---	--

Colors should be used in very specific cases, or else each documentation writers would color his/her pages the way he/she thinks it's better, and it would look like a mess. One such specific case in which colors can help is when you want them to work as tags or captions. For (a lame) example, in this paragraph, guidelines are in red and justifications are in blue. Yes, it's a really really lame example, I know. 😊

Text Breaks

★ This section refers to: [Notation Guide >> Text Breaks](#).

Text breaks shouldn't be used. If you'd like paragraphs or headings to have more spacing (before or after), the style sheet should be changed, not the contents. Patrick explained this a long time ago. Other stuff in this section (paragraphs, horizontal ruler, — symbol and – symbol) can be used when necessary.

Links

★ This section refers to: [Notation Guide >> Links](#).

All types of links can and should be used. I already used a few in this document. Just watch out for links to non-existing pages when writing on the official documentation.

Lists

★ This section refers to: [Notation Guide >> Lists](#).

Lists can be used for many purposes. Every time we list some things that are in order, ordered lists are used. If they don't have a specific order, unordered lists are the case. List should be nested if needed for a better organization. Unordered lists should be created only with the * (star) notation only, so all pages use the same style of bullet.

- This is an unordered list in star notation;
- Items can have sub-items
 - That can have sub-items
 - That can have sub-items...
 - What is the limit?
- Mixing ordered and unordered lists is possible:
 1. One;
 2. Two;
 3. Three.



List indentation

Use tabs to indent nested lists. This way your page's markup is more readable

	and easier to maintain.
--	-------------------------







Images and Icons

★ This section refers to: [Notation Guide >> Images](#) and [Notation Guide >> Misc](#).

External images should be used only when strictly necessary (meaning, don't use images as list bullets or box icons). Also, try to use only images that are very unlikely to be removed from its current URL, to reduce document maintenance. Pay attention on copyright issues too! Attached images are less prone to become missing links, however, we should not clutter the documentation with unnecessary attachments and copyrights are also a issue here.

Example:



Icons are cool in a number of situations. Some of them, such as , ,  or  can make the documentation look professional, but some others, such as  and  may give a feeling of amateurship and I wouldn't advise them for pages that are exported to form the official documentation.

Tables

★ This section refers to: [Notation Guide >> Tables](#).

Tables are very useful when lists just don't do it. Meaning: don't write a table when a list suffices. Tables are more organized, because you can align the text in columns. Since the markup text for tables in confluence is not very easy to read, remember complex and big tables are very hard to maintain.

The table below was copied from a reference page on WebWork's configuration (just the first two lines were copied). This is an example where tables are good: a list

wouldn't be as organized as this table to display these files and their properties.

File	Optional	Location (relative to webapp)	Purpose
web.xml	no	/WEB-INF/	Web deployment descriptor to include all necessary WebWork components
xwork.xml	no	/WEB-INF/classes/	Main configuration, contains result/view types, action mappings, interceptors, etc

Advanced Formatting

★ This section refers to: [Notation Guide >> Advanced Formatting](#).

I've already made my point about info, warning, tip and note boxes. Other interesting markups are `noformat` and `code`. The former can be used for general purpose text while the latter is used to display example source code, be it HTML, XML, Java or anything that is part of a software solution. When displaying something that has a name, use a title, as the example below demonstrates.

HelloWorld.java
<pre>/** Hello World class. */ public class HelloWorld { /** Main method. */ public static void main(String[] args) { System.out.println("Hello, World!"); } }</pre>

A typical example of `noformat` would be the command line statements to compile and run the code above. We should also standardize terminal notation (`{ $ }`) for command

prompt).

```
$ javac HelloWorld.java  
  
$ java HelloWorld  
Hello, World!
```

Do not use tabs inside `noformat` and `code`, use two spaces instead. This way your code is indented but keeps lines short. Large lines should be splitted as to fit in a 800x600 resolution screen without horizontal scroll bars.

Your Comments Please

Please contribute to this page. Let me know if you have a different opinion on something (please justify it). Please warn me if I wrote something wrong or if this proposed Style Guide is missing something. Feel free to correct my english, since I'm not a native speaker.

Tutorial

This page last changed on Oct 30, 2005 by [plightbo](#).

TODO: we need to merge the old tutorials in with the new ones.

TODO: write up some actual tutorials!

TODO: maybe create a template for tutorial pages?

1. [Getting Started](#) / [Quick Start Guide](#) (TODO: consolidate these articles and incorporate "prototype")
2. Downloading and installing WebWork
3. Setting up the test environment (to test tutorial source code)
4. Basic configuration and your first action (Hello WebWorld)
5. Understanding actions
6. Understanding results
7. Meet WebWork tag library (*would also explain a little bit of OGNL*)
8. Evaluating other view options: Velocity
9. Evaluating other view options: FreeMarker
10. Understanding interceptors
11. Performing validation
12. Performing dependency injection (IoC) through components
13. Going i18n (internationalization)
14. Retrieving data without a full request using XHR Ajax

Old tutorials:

1. [Lesson 1 - Setting up webwork in a web application](#)
2. [Lesson 2 - An html form with no data](#)
3. [Lesson 3 - An html form with data](#)
4. [Lesson 4 - An html form with data, without getters or setters](#)
5. [Lesson 5 - Views](#) (JSP, Velocity, Freemarker)
6. [Lesson 6 - Interceptors](#)

Getting Started

This page last changed on Sep 10, 2004 by [plightbo](#).

This site is geared towards developers that have an understanding towards certain technologies. Before diving into how Webwork works and running demos, it is recommended that you review the concepts below:

- Java
- Servlets, JSP, and Tag Libraries
- JavaBeans
- HTML and HTTP
- Web Containers (ex. Tomcat)
- XML

Website & downloads

This site is set up with many features. Here are links to help you around:

- [Download Webwork](#) - download Webwork Distribution
- [Webwork Mailing List](#) - [Browse mail archive](#) or [post a question](#). The list is full of active developers, contributors, and power users. This is the best and quickest way to get a question answered.
- [CVS](#) - Browse CVS and source at java.net
- [Webwork Wiki](#) - Powered by [Confluence](#), the professional J2EE wiki
- [Webwork Bugs & Issues](#) - Powered by [JIRA:Bug & Issue Traking System](#)
- [OpenSymphony Home](#)

What's included in the distro

The distribution contains the following directory layout:

```
docs/  
lib/  
src/  
src/java/template/  
webwork-(VERSION).jar  
webwork-example.war  
webwork-migration.jar
```

The docs directory contains the current Javadocs, the document you are reading, as

well as JUnit reports for the build. The lib directory contains the required as well as the optional dependencies for Webwork:

```
lib/  
  core/  
  migration/  
  optional/
```

Note that none of the optional packages are required to use Webwork. If you wish to use certain features such as JasperReports and FreeMarker results, you must include the optional packages.

Webwork also comes packaged with all the source files and the templates for the JSP tags.

Installing

The following illustrates how your web application should be set up. Copy the webwork-(VERSION).jar, all the *.jar files in /lib/core and any necessary optional *.jar files in /lib/optional to your webapp/lib directory. If you need to customize your own templates (how HTML is rendered from webwork UI tags), copy the /src/java/template directory into your webapp/ directory. Your webapp should look similar to this:

```
/mywebapp/  
/mywebapp/template/  
/mywebapp/META-INF/  
/mywebapp/WEB-INF/  
/mywebapp/WEB-INF/classes/  
/mywebapp/WEB-INF/lib/  
/mywebapp/WEB-INF/lib/CORE&OPTIONAL *.jar  
/mywebapp/WEB-INF/web.xml
```

Onward to [Configuration](#) or the [Webwork Tutorial](#)

Running demos

In order to run webwork applications and demos, you need to have a servlet/jsp engine. If you don't, we suggest you learn about [Apache Tomcat](#), which is a free Servlet container from the Apache Jakarta Project, or Resin, from [Caucho Technology](#),

which is free for non-commercial use. Once you have a Servlet container setup, you can install the webwork example applications (*.war) and any other demos by placing the .war file inside the containers webapp directory. Example of location with tomcat:


```
<TOMCAT_HOME>/webapps/webwork-example.war
```

After the war file is in the correct location, start your web container and access your application through a web browser with the following url.

[http://??HOST:PORT??*/webwork-example*](http://*??HOST:PORT??*/webwork-example*)*

Lesson 1: Setting up webwork in a web application

For this lesson, you need to have a Servlet container set up and know how to create a web application. If you don't, we suggest you learn about [Apache Tomcat](#), which is a free Servlet container from the Apache Jakarta Project, or Resin, from [Caucho Technology](#), which is free for noncommercial use.

	Notation Throughout these lessons, we'll assume that your web application root is the directory [webapp], and that your Java source files are kept in [src].
---	--

To install WebWork in a web application:


1. Download WebWork. The current version can be found at [WebWork's home page](#). This tutorial is based on version 2.1.7.
2. Set up an empty web application. For example, if you are using Tomcat, this will have something like the following directories (the directory "webwork-lessons" is referred to as [webapp] in these lessons):

```
[the tomcat root directory]
\|_webapps
  \|_webwork-lessons
    \|_WEB-INF
      \|_classes
      \|_lib
```

3. Copy the required WebWork libraries to your web application:

- copy webwork-2.1.7.jar to [webapp]/WEB-INF/lib ,
- copy lib/core/*.jar to [webapp]/WEB-INF/lib (not to [webapp]/WEB-INF/lib/core).

1. Configure `[webapp]/WEB-INF/web.xml`, and create `[webapp]/WEB-INF/classes/xwork.xml` and `[webapp]/WEB-INF/classes/validators.xml`, as described below.

	WebWork jar name If you have a later version of WebWork than 2.1.7, the WebWork jar will not be named <code>webwork-2.1.7.jar</code> . Be sure to replace all occurrences of this jar's name below with the name of the jar you are using.
---	--

web.xml:

Create the following `web.xml` file in `[webapp]/WEB-INF`. If you already have a `web.xml` file, just add the content of the `<web-app>` tag below to your existing `<web-app>` tag.

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

<web-app><display-name>My WebWork
Application</display-name><servlet><servlet-name>webwork</servlet-name><servlet-class>com.opensymphony.webwork.servlet.ServletDispatcher</servlet-class></servlet></web-app>
```

This registers `ServletDispatcher` as a servlet, and maps it to the suffix `*.action`. We will go into this more in the section on Actions in the [next lesson](#). WebWork's taglib descriptor allows WebWork tags to be used (see [lesson 4.1](#)).

Read more: [web.xml](#)

xwork.xml:

Create the following file `xwork.xml` in `[webapp]/WEB-INF/classes/`.

```
<!DOCTYPE xwork PUBLIC "-//OpenSymphony Group//XWork
1.0//EN" "http://www.opensymphony.com/xwork/xwork-1.0.dtd">

<xwork><!-- Include webwork defaults (from WebWork JAR). --><include
file="webwork-default.xml" /><!-- Configuration for the default package. --><package
name="default" extends="webwork-default"></package></xwork>
```

For now, this xwork.xml does only two things:

- It informs WebWork that it should import the configuration information from webwork-default.xml. (This file is located at the root of the webwork-2.1.7.jar, so it is sure to be found.)
- It defines a default package (with the <package> section) where WebWork elements like actions, results and interceptors are registered.

Read more: [xwork.xml](#)

validators.xml:

Create a file validators.xml in [webapp]/WEB-INF/classes/ with the following content:

```
<validators>
  <validator name="required"
    class="com.opensymphony.xwork.validator.validators.RequiredFieldValidator"/>
  <validator name="requiredstring"
    class="com.opensymphony.xwork.validator.validators.RequiredStringValidator"/>
  <validator name="int"
    class="com.opensymphony.xwork.validator.validators.IntRangeFieldValidator"/>
  <validator name="date"
    class="com.opensymphony.xwork.validator.validators.DateRangeFieldValidator"/>
  <validator name="expression"
    class="com.opensymphony.xwork.validator.validators.ExpressionValidator"/>
  <validator name="fieldexpression"
    class="com.opensymphony.xwork.validator.validators.FieldExpressionValidator"/>
  <validator name="email"
    class="com.opensymphony.xwork.validator.validators.EmailValidator"/>
  <validator name="url"
    class="com.opensymphony.xwork.validator.validators.URLValidator"/>
  <validator name="visitor"
    class="com.opensymphony.xwork.validator.validators.VisitorFieldValidator"/>
  <validator name="conversion"
    class="com.opensymphony.xwork.validator.validators.ConversionErrorFieldValidator"/>
</validators>
```


This file defines the validators used, for example, for validating html form fields.

Read more: [Validation](#)

All Set Up!

Restart your servlet container (for example, restart Tomcat), and your webapp should be ready for use as a skeleton WebWork application.

To test whether everything is working, create `[webapp]/test.jsp`:

```
<html>
<body>
Hello html world
<hr/>
<%
    out.println("Hello jsp world.");
%>
</body>
</html>
```

If you can load this file in your browser and see the two Hello messages, your web application is working.

[Next Lesson](#)

Lesson 2: An html form with no data

In this lesson, we are going to create a JSP with a form which, when submitted, loads a different JSP page saying "Hello, WebWorld!". To do that, we are going to write our first WebWork **action**.

Background: what are actions?

In JSP programming, submitting a form typically loads another JSP page where the form is processed using `request.getProperty()`. The form html looks like: `<form action="foo.jsp">`.

When you submit an html form using WebWork, the form is sent to a Java class that you write yourself, not to a JSP page. These classes are called WebWork **actions**. The form html typically looks like: `<form action="foo.action">`.

In a Model-View-Controller approach, the WebWork action is part of the Controller, leaving to JSP pages what they do best: the View. (If you don't know what a Model-View-Controller is, don't worry about this.)

The code

These are typical steps for creating a form and its action:

1. Create a JSP page with a form that calls the action.
2. Create the action class.
3. Register the action in `xwork.xml`.
4. Create a JSP page that will display the result.

5. Compile the action class. If necessary, restart your webapp.



Watch out for typos!

If something doesn't work properly, the first thing you'll want to do in this lesson is check all the files for typos, both in the files themselves as well as in the file names. This is a common source of errors.

1. Create a JSP page with a form that calls the action

Past this code into file called `page02.jsp`.

```
<html><head><title>A simple form with no data</title></head><body><p>Click the  
button below to activate Form02Action.</p><form action="form02.action"  
method="post"><p><input type="submit" value="Click me." /></p></form></body></html>
```

This is a form with no entry fields, just a submit button. Notice that the form's action attribute doesn't point to a jsp page, but to something strange called `form02.action`. We'll soon see why.

2. Create the action class

We are now going to create a Java class that will be part of the Java package "lessons". It doesn't matter where you keep this and other .java files; for example, they could be in these directories (if you are using Windows):

```
c:  
 \ |_java  
   \ |_src  
     \ |_lessons
```

In these lessons, the above "src" directory is referred to as [src].

All our Java classes will be compiled to [webapp]/WEB-INF/classes. You'll have to

include all the [webapp]/WEB-INF/lib/*.jar files in your CLASSPATH in order to compile these classes.

Paste this code into a file [src]/lessons/Form02Action.java:

```
package lessons;

import com.opensymphony.xwork.ActionSupport;

public class Form02Action extends ActionSupport {
    String hello;

    publicString getHello() {
        return hello;
    }

    publicString execute() throws Exception {
        hello = "Hello, WebWorld!";
        return SUCCESS;
    }
}
```

3. Register the action in xwork.xml

Edit the xwork.xml file as shown below, adding the form02 action and something called an interceptor to the default package.

```
<!DOCTYPE xwork PUBLIC "-//OpenSymphony Group//XWork
1.0//EN" "http://www.opensymphony.com/xwork/xwork-1.0.dtd">

<xwork><!-- Include webwork defaults (from WebWork JAR). --><include
file="webwork-default.xml" /><!-- Configuration for the default package. --><package
name="default" extends="webwork-default"><!-- Default interceptor stack.
--><default-interceptor-ref name="defaultStack" /><!-- 02 --><action name="form02"
class="lessons.Form02Action"><result name="success"
type="dispatcher">page02-success.jsp</result></action></package></xwork>
```

Read more: [xwork.xml](#)

4. Create a JSP page that will display the result

Paste this code into a file [webapp]/page02-success.jsp:

```
<%@ taglib uri="webwork" prefix="ww" %><html><head><title>Success page for form with  
no data</title></head><body><ww:property value="hello" /></body></html>
```

Try it

Don't forget to compile your action to [webapp]/WEB-INF/classes, and to restart your web application if necessary.

Go ahead and try it now: click the form submit button on page02.jsp and see what happens. You should see a page that says "Hello, WebWorld!".

How the code works

The above four files work together like this.

- You click the form submit button on page02.jsp, sending it to your web application server.
- The server receives the request for `helloWebWebWorld.action`. Looking in [webapp]/WEB-INF/web.xml, it sees that all *.action requests are to be handed off to `com.opensymphony.webwork.dispatcher.ServletDispatcher`. Essentially, the request is handed to WebWork now.
- WebWork looks in `xwork.xml` for an action named "form02". There it finds that this corresponds to the class "lessons/Form02Action," instantiates it, and calls its `execute()` method.
- `execute()` returns SUCCESS, and WebWork looks again in `xwork.xml` to see what page to load if SUCCESS is returned. It finds the page "form02-success.jsp".
- The page page02.jsp is processed (the `<ww:property value="hello" />` tag calls the getter `getHello()` of Form02Action) and sent back to the browser.

To sum up: with WebWork, all html forms are sent to actions. The actions return constants like SUCCESS to specify (via `xwork.xml`) what page to return.

In this example, the form contained no data. In the next lesson, we'll see how to send form data to an action. Since page02-success.jsp called a getter of the action, you

might guess that the form fields are going to call setters. You'd be right.

[Previous Lesson](#) | [Next Lesson](#)

Lesson 3: An html form with data

In this lesson, we will create a form in which you can enter your name. For example, if you enter "Bob" and click the submit button, you'll get a page saying "Hello, Bob!". If you don't enter a name, you'll get a screen saying: "Hmm, you don't seem to have entered a name. Go back and try again please."

As before, we set everything up in four steps: create the form, create the action, register the action, and create the landing page (or in this case, pages).

1. Create the form

Paste this html into [webapp]/page03.jsp:

```
<html><head><title>A simple form with data</title></head><body><p>What is your
name?</p><form action="form03.action" method="post"><p><input type="text"
name="yourName"></p><p><input type="submit" value="Submit your name."
/></p></form></body></html>
```

2. Create the form action

Paste this code into [src]/lessons/Form03Action.java:

```
package lessons;

import com.opensymphony.xwork.ActionSupport;

public class Form03Action extends ActionSupport {

    String yourName;

    public void setYourName(String p_yourName) {
        yourName = p_yourName;
    }
}
```

```

publicString getYourName() {
    return yourName;
}

publicString execute() throws Exception {
    if (yourName == null || yourName.length() == 0)
        return ERROR;
    else return SUCCESS;
}
}

```

3. Register the action in xwork.xml:

Edit [webapp]/WEB-INF/classes/xwork.xml:

```

<!DOCTYPE xwork PUBLIC "-//OpenSymphony Group//XWork
1.0//EN" "http://www.opensymphony.com/xwork/xwork-1.0.dtd">

<xwork><!-- Include webwork defaults (from WebWork JAR). --><include
file="webwork-default.xml" /><!-- Configuration for the default package. --><package
name="default" extends="webwork-default"><!-- Default interceptor stack.
--><default-interceptor-ref name="defaultStack" /><!-- 02 --><action name="form02"
class="lessons.Form02Action"><result name="success"
type="dispatcher">page02-success.jsp</result></action><!-- 03 --><action
name="form03" class="lessons.Form03Action"><result name="success"
type="dispatcher">page03-success.jsp</result><result name="error"
type="dispatcher">page03-error.jsp</result></action></package></xwork>

```

4. Create the success and error pages

Create [webapp]/page03-success.jsp:

```

<%@ taglib uri="webwork" prefix="ww" %><html><head><title>Success page for form with
data</title></head><body>

Hello, <ww:property value="yourName" />!

</body></html>

```

Create [webapp]/page03-error.jsp:

```

<html><head><title>Error page for form with data</title></head><body>

Hmm, you don't seem to have entered a name. Go back and try again please.

</body></html>

```


Try it

Don't forget to compile your action to [webapp]/WEB-INF/classes, and to restart your web application if necessary.

Go ahead and try it now: click the form submit button and see what happens. Try it with and without entering a name.

How the code works

There are only two differences between this example and the previous lesson.

- When the action is called, its `setYourName()` setter is called with the contents of the form field named `yourName`.
- After the action has been called (which is when its `execute()` method returns), `WebWork` has two options. If `ERROR` is returned, `WebWork` will return `page03-error.jsp`; if `SUCCESS`, `page03-success.jsp`. Just as in the last lesson, the `<ww:property>` tag calls the action's getter (in this case, `getYourName()`).

[Lesson 2 - An html form with no data](#) | [Lesson 4 - An html form with data, without getters or setters](#)

Lesson 4: An html form with data, without getters or setters

For the form field named "yourName" in the previous lesson, we also had to create the getters and setters `getYourName()` and `setYourName()` in the action, as well as the private variable `yourName`. With dozens of forms and hundreds of form fields, you'll be typing thousands of getters and setters. That can get old fast. In this lesson, we'll repeat the last lesson, but without any of that extra typing.

1. Create the html form

Use the same JSP form from the previous lesson, but change the form action to `page04.action`:

```
<html><head><title>A simple form with data</title></head><body><p>What is your
name?</p><form action="form04.action" method="post"><p><input type="text"
name="yourName"></p><p><input type="submit" value="Submit your name."
/></p></form></body></html>
```

2. Create the form action

Paste this code into `[src]/lessons/Form04Action.java`:

```
package lessons;

import com.opensymphony.xwork.ActionSupport;
import com.opensymphony.webwork.interceptor.ParameterAware;

import java.util.Map;

public class Form04Action extends ActionSupport implements ParameterAware {

    Map parameters;

    public Map getParameters() {
```

```

    return parameters;
}

public void setParameters(Map parameters) {
    this.parameters = parameters;
}

public String execute() {
    String[] yourName = (String[]) parameters.get("yourName");
    if(yourName == null || yourName[0] == null || yourName[0].length() == 0)
        return ERROR;
    else return SUCCESS;
}
}

```

Register the action in xwork.xml:

Edit [webapp]/WEB-INF/classes/xwork.xml:

```

<!DOCTYPE xwork PUBLIC "-//OpenSymphony Group//XWork
1.0//EN" "http://www.opensymphony.com/xwork/xwork-1.0.dtd">

<xwork><!-- Include webwork defaults (from WebWork JAR). --><include
file="webwork-default.xml" /><!-- Configuration for the default package. --><package
name="default" extends="webwork-default"><!-- Default interceptor stack.
--><default-interceptor-ref name="defaultStack" /><!-- 02 --><action name="form02"
class="lessons.Form02Action"><result name="success"
type="dispatcher">page02-success.jsp</result></action><!-- 03 --><action
name="form03" class="lessons.Form03Action"><result name="success"
type="dispatcher">page03-success.jsp</result><result name="error"
type="dispatcher">page03-error.jsp</result></action><!-- 04 --><action name="form04"
class="lessons.Form04Action"><result name="success"
type="dispatcher">page04-success.jsp</result><result name="error"
type="dispatcher">page03-error.jsp</result><interceptor-ref
name="servlet-config" /></action></package></xwork>

```

Create the success and error pages

We'll use the same error page, but create a slightly different success page page04-success.jsp. The only difference is the <ww:property> tag.

```

<%@ taglib uri="webwork" prefix="ww" %><html><head><title>Success page for form with
data</title></head><body>

Hello, <ww:property value="parameters.yourName" />!

</body></html>

```

Try it

Don't forget to compile your action to [webapp]/WEB-INF/classes, and to restart your web application if necessary.

Go ahead and try it now. Load `page04.jsp`, enter "Bob" in the text field, and click the form submit button. You should see `page04-success.jsp` saying "Hello, Bob!"

How the code works

You've probably figured out what is going on just from looking at the code.

Instead of a setter `setYourName()` setting a private variable `yourName` in the action, `setParameters()` magically extracts everything from the JSP request object and puts into a private local Map `parameters`. Then `execute()`, instead of looking for a `yourName` variable, is able to get the value of the "yourName" field from `parameters`. So far so good .

Back on the `page04-success.jsp` page, `<ww:property value="yourName" />` isn't going to work any more, because there is no `getYourName()` getter in the action. Instead, `<ww:property value="parameters.yourName" />` calls the `getParameters()` getter, and is able to get the value of the "yourName" field. Pretty neat!

We haven't covered how to handle radio buttons, checkboxes, and other strange html form fields. That involves dealing with the fact that every entry in the `parameters` Map is a `String[]`. We'll cover this in a later lesson.

[Previous Lesson](#) | [Next Lesson](#)

Quick Start Guide

This page last changed on Oct 08, 2005 by [plightbo](#).

WebWork is a popular, easy-to-use MVC framework, for more information on the WebWork project, please visit [WW:WebWork](#). This guide should be helpful to seasoned Java developers with previous experience in MVC frameworks. We will briefly cover the three main components on a WebWork2 based application, the configuration the action classes, and the views.

GETTING STARTED

To use WebWork as your framework for writing Java-based web applications, you need to start by installing the various libraries (these are all available in the [webwork 2.x distribution](#)):

Core JAR files

- [webwork-2.1.5.jar](#)
- [xwork-1.0.3.jar](#)
- ognl-2.6.5.jar
- commons-logging.jar
- oscore-2.2.4.jar
- velocity-dep-1.3.1.jar

Optional JAR files

- jstl.jar (needed for Standard Tag Libraries)
- cos-multipart.jar
- pell-multipart.jar
- standard.jar
- mail.jar

CONFIGURATION

WebWork is built upon the Xwork framework. Xwork handles the translation requests to commands execution, but let's not worry about that right now. You need to know this information in case you were curious about the xwork JAR file, and if you want to learn some of the more advanced features of the WebWork command structure, you can visit the [XW:XWork](#) site.

Example web.xml file

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
    PUBLIC
        "-//Sun Microsystems, Inc.//DTD Web Application
        2.3//EN" "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app><display-name>WebWork 2.0 Quick
Start</display-name><servlet><servlet-name>webwork</servlet-name><servlet-class>com.opensymphony
```

Servlet mappings

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

The above section will map ANY servlet called with an extension of **.action** to the WebWork base servlet, and assume it is a WebWork action class.

Taglibs

```
<taglib><taglib-uri>webwork</taglib-uri><taglib-location>/WEB-INF/lib/webwork-2.1.5.jar</taglib-location></taglib>
```

The above section will load the standard WebWork tag libraries. This is required for JSP 1.1 compatible containers. To load more or different libraries, add more **<taglib>** calls.

Example Xwork config file (xwork.xml)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE xwork
    PUBLIC
        "-//OpenSymphony Group//XWork
```

```
"http://www.opensymphony.com/xwork/xwork-1.0.dtd">
<xwork><include file="webwork-default.xml"/><package name="default"
extends="webwork-default"><interceptors><interceptor name="security"
class="com.acme.LoginCheck"/></interceptors><default-interceptor-ref
name="security"/><action name="showForm" class="com.acme.FormAction"><result
name="success" type="dispatcher"><param
name="location"/>/form.jsp</param></result></action><action name="saveForm"
class="com.acme.FormAction" method="processForm"><result name="success"
type="dispatcher"><param name="location"/>/form.jsp</param></result><result
name="missing-data" type="dispatcher"><param
name="location"/>/form.jsp</param></result></interceptor-ref
name="security"/></action></package></xwork>
```

ACTIONS

```
<action name="saveForm" class="com.acme.FormAction" method="processForm">
```

Actions are the basic "unit-of-work" in WebWork, they define, well, actions. An action will usually be a request, (and usually a button click, or form submit). The main action element (tag is too synonymous with JSP) has two parts, the friendly name (referenced in the URL, i.e. **saveForm.action**) and the corresponding "handler" class.

The method parameter tells WebWork which method to call based upon this action. If you leave the method parameter blank, WebWork will call the method **execute()** from the Action Interface by default. Since every Action must implement the Action Interface, this method will always be available.

RESULTS

```
<result name="missing-data" type="dispatcher"><param
name="location"/>/form.jsp</param></result>
```

Result tags tell WebWork what to do next after the action has been called. There are a standard set of result codes built-in to WebWork, (in the Action interface) they include:

```
String SUCCESS = "success";
String NONE    = "none";
String ERROR   = "error";
String INPUT   = "input";
```

```
String LOGIN    = "login";
```

You can extend these as you see fit. Most of the time you will have either **SUCCESS** or **ERROR**, with **SUCCESS** moving on to the next page in your application;

```
<result name="success" type="dispatcher"><param  
name="location">/thank_you.jsp</param></result>
```

...and **ERROR** moving on to an error page, or the preceding page;

```
<result name="error" type="dispatcher"><param  
name="location">/error.jsp</param></result>
```

You can stack as many result tags within a single action tag as you wish.

For further reading on Result types, see [WW:Result Types](#).

INTERCEPTORS

Interceptors allow you to define code to be executed before and/or after the execution of an action. Interceptors can be a powerful tool when writing web applications. Some of the most common implementations of an Interceptor might be:

- Security Checking (ensuring the user is logged in)
- Trace Logging (logging every action)
- Bottleneck Checking (start a timer before and after every action, to check bottlenecks in your application)

You can also group Interceptors together to create "stacks". If you wanted to do a login check, security check, and logging all before an Action call, this could easily be done with an interceptor stack.

For further Reading on Interceptors, see [XW:Interceptors](#)

For further Reading on Xwork configuration files, see [XW:Configuration](#)

ACTION CLASSES

The action classes do what they say, they handle the action. They are called by the actions specified in the **xwork.xml** file and initiated by the user in the "views". To turn your class into an action class, you simply need to extend the class **ActionSupport** or implement the **Action** interface.

Here is what our **saveForm** action looks like in its Java form (NOTE: If you look in the **xwork.xml** file above, we've overridden the action to call the **processForm** method):

```
package com.acme;
import com.opensymphony.xwork.*;

public class FormAction extends ActionSupport {

    private FormBean myFormBean = new FormBean();

    public void setFormBean(FormBean inBean) {
        myFormBean = inBean;
    }

    public FormBean getFormBean() {
        return myFormBean;
    }

    public String processForm() {

        FormParameters formParams = this.getFormBean();
        checkBizRules(formParams);
        this.saveParamsToDb(formParams);

        return SUCCESS;
    }
    ...
}
```

VIEWS

WebWork supports JSP, Velocity, and FreeMarker for your application presentation layer. For this example, we will use a JSP file. The appropriate presentation layer is denoted by the result type specified. For JSPs, the result type is the name mapped to the **com.opensymphony.webwork.dispatcher.ServletDispatcherResult**. Typically, this is "dispatcher".

WebWork comes packaged with a tag library (taglibs). You can use these taglibs as components in your JSP file. Here is an section of our **form.jsp** page:

Unable to find source-code formatter for language: jsp. Available languages are: xhtml, javascript, java, none, html, actionscript, xml, sql

```
<%@ taglib prefix="ww" uri="webwork" %>
<html>
<head><title>Webwork Form Example</title></head>
<body>

<ww:form name="myForm" action="saveForm.action" method="POST">
  <table>

    <ww:textfield label="First Name" name="formBean.firstName"
value="formBean.firstName"/>
    <ww:textfield label="Last Name" name="formBean.lastName"
value="formBean.lastName"/>

  </table>
  <input type="submit" value="Save Form"/>
</ww:form>
</body>
```

The process of events will go as follows:

1. WebWork will take notice since the URI ends in **.action** (defined in our **web.xml** files)
2. WebWork will look up the action **saveForm** in its action hierarchy and call any Interceptors that we might have defined.
3. WebWork will translate **saveForm** and decide we would like to call the method **processForm** in our class **com.acme.FormAction** as defined in our **xwork.xml** file.
4. Our method will process successfully and give WebWork the **SUCCESS** return parameter.
5. WebWork will translate the **SUCCESS** return parameter into the location **thank_you.jsp** (as defined in **xwork.xml**) and redirect us accordingly.

SUMMARY

The purpose of this guide is to provide the user with a quick-and-dirty understanding of WebWork2. I hope we successfully briefed you on the three most important components of any WebWork based application, the configuration (including **web.xml** and **xwork.xml**), the action classes, and the views. This information should give you a starting point to experiment and become more familiar with the rising star of the open source, Model2, web frameworks.

Miscellaneous Notes

It is not necessary to map and load the `WebWorkVelocityServlet` in in the **web.xml** in order to use Velocity rendered pages. WebWork includes a built-in Velocity result type. Using the result type should be faster than sending the result to the servlet dispatcher which then delegates to the servlet mapping for `WebWorkVelocityServlet`.

Originally by:

Matt Dowell

matt.dowell@notiva.com

September 15, 2003

Lesson 5: Views

There are some different technologies that you could use as the view, i.e., to construct the user interface:

Lesson 5.1 - Java Server Pages

JSP is the common choice, because most Java web developers are already familiar with the technology. This lesson assumes you already have experience with Java Server Pages and demonstrates how you can use the WebWork features in JSP, mostly by using WebWork tags.

[Go to lesson 4.1](#) (Currently named 4.x while documentation is being rewritten.)

Lesson 5.2 - Velocity

Velocity is a Java-based template engine that provides a simple, but powerful, template language that replaces JSP and allows for separation of concerns. This lesson assumes that you are already familiar with Velocity and teaches you how to use WebWork features from it.

[Go to lesson 4.2](#) (Currently named 4.x while documentation is being rewritten.)

Lesson 5.3 - Freemarker

Designed for MVC pattern, Freemarker is another Java-based template engine that provides a powerful template language that replaces JSP, but can remain JSP-compatible with a JSP taglib support. This lesson teaches you how to use

WebWork and Freemarker together.

[Go to lesson 5.3](#) (Currently named 4.x while documentation is being rewritten.)

[Previous Lesson](#) | [Next Lesson](#)

Lesson 4.1: Using JSP as the View

When using JSP to render the views, you can choose to access the action's data using scriptlets or tags. Tags are the recommended approach.

Accessing Action Data through Scriptlets:

Action data can be accessed through an object called *Value Stack*. The example below does the same thing as the result page of [lesson 3](#)'s second example (*Supplying Data to the Action*), but using scriptlets:

```
<%@ page import="com.opensymphony.xwork.util.OgnlValueStack"
%><html><head><title>WebWork Tutorial - Lesson 4.1 - Lesson 3's example
modified</title></head><body>

<%
OgnlValueStack stack = (OgnlValueStack)request.getAttribute("webwork.valueStack");
out.write("Hello, " + stack.findValue("person"));
%>

</body></html>
```

WebWork tags, however, are recommended over scriptlets. For instance, `<ww:property />` tags do exactly what the scriptlet above does, with a cleaner syntax and also handles the case where the Value Stack doesn't exist.

WebWork Tag Library:

We've already showed in [lesson 3](#)'s example how to access an action's property using tags. This section describes and exemplifies the use of the WebWork Tag Library, which can be divided in seven categories:

- **Common tags:** the most frequently used, basic tags;

- **Componentisation tags:** foster componentisation within your views;
- **Flow control tags:** govern the flow of control within the JSP page;
- **Iteration tags:** iterate over elements and manipulate iterable objects;
- **UI tags:** generate HTML form fields and controls;
- **VUI tags:** *volunteers needed to write this part*;
- **Internationalisation tags:** internationalise your views.

Common tags

<code><ww:property /></code>	Gets the value of a result attribute. If the value isn't given, the top of the stack will be returned.
<code><ww:push /></code>	Pushes a value onto the Value Stack.
<code><ww:param /></code>	Sets a parent tag's parameter. This tag is used only inside another tag to set the value of some property of the parent tag.
<code><ww:set /></code>	Sets the value of an object in the Value Stack to a scope (page, stack, application, session). If the value is not given, the top of the stack is used. If the scope is not given, the default scope of "webwork" is used.
<code><ww:url /></code>	Builds an encoded URL.

EXAMPLE NEEDED.

Read more: [Non-UI Tags](#)

Componentisation tags

<code><ww:action /></code>	Executes an Action from within the context of a taglib. The body of the tag is used to display the results of the action invocation.
<code><ww:bean /></code>	Creates a JavaBean, instantiate its

	properties and place it in the ActionContext for later use.
<code><ww:include /></code>	Includes another page or action.

EXAMPLE NEEDED.

Read more: [Non-UI Tags](#)

Flow control tags

This if-else set of tags works just like if-else scriptlets.

<code><ww:if /></code>	Conditional execution path. That is, evaluates the tag body if a boolean expression is true.
<code><ww:else /></code>	Negative execution path for the if tag. That is, if the preceeding conditional tag's boolean expression evaluated to false, then evaluate this tag's body.
<code><ww:elseif /></code>	Negative conditional execution path for the if tag. That is, if the preceeding conditional tag's boolean expression evaluated to false and if this tag's boolean expression evaluates to true, then evaluate this tag's body.

EXAMPLE NEEDED.

Read more: [Non-UI Tags](#)

Iteration tags

<code><ww:iterator /></code>	Iterates over a collection.
------------------------------------	-----------------------------

<code><ww:generator /></code>	Generates iterators.
<code><ww:append /></code>	Appends several iterators.
<code><ww:subset /></code>	Gets a subset of an iterator.
<code><ww:merge /></code>	Merges several iterators into one.
<code><ww:sort /></code>	Sorts an iterator.

EXAMPLE NEEDED.

Read more: [Non-UI Tags](#)

UI tags

The UI tags wrap generic HTML controls while providing tight integration with the core framework. The tags have been designed to minimize the amount of logic in compiled code and delegate the actual rendering of HTML to a template system. The UI tags attempt to cover the most common scenarios, while providing a Component Tag for creating custom components. The UI tags also provide built-in support for displaying inline error messages.

There is a separate lesson about WebWork UI Tags which explains in detail how they work, how you could customize their appearance through the use of templates, how to create custom components, etc.

[Go to WebWork UI Tags Lesson.](#)

VUI(Voice UI) tags

<code><ww:audio /></code>	???
<code><ww:prompt /></code>	???
<code><ww:filled /></code>	???
<code><ww:log /></code>	???

Volunteers needed to write this part.

Internationalisation tags

<code><ww:text /></code>	Prints out an internationalized string.
<code><ww:i18n /></code>	Places a resource bundle on the Value Stack, for access by the text tag.

Read more: [UI Tags](#)

[Previous Lesson](#) | [Next Lesson](#)

Lesson 4.1.1: WebWork UI Tags

In WebWork, the UI tags wrap generic HTML controls while providing tight integration with the core framework. The tags have been designed to minimize the amount of logic in compiled code and delegate the actual rendering of HTML to a template system. The UI tags attempt to cover the most common scenarios, while providing a Component Tag for creating custom components. The UI tags also provide built-in support for displaying inline error messages.

This lesson tries to explain how to take advantage of the UI tags to build forms and other graphical controls and, by explaining how the template system works, teaches you how to change the look of existing components and create your own UI components.

Building forms:

WebWork comes with ready-to-use tags to construct forms. Some of these tags relate directly to HTML tags that are used to make forms and you probably can figure them out by their names: `<ww:checkbox />`, `<ww:file />`, `<ww:form />`, `<ww:hidden />`, `<ww:label />`, `<ww:password />`, `<ww:radio />`, `<ww:select />`, `<ww:submit />`, `<ww:textarea />` and `<ww:textfield />`.

To build forms with these tags, place them in your page as you would do with the HTML tags. The only difference is that the parameters should be enclosed in double quotes and single quotes (`key='value'`). That's because names that are not single-quoted are evaluated against the Value Stack.

Let's check out an example:

ex01-index.jsp:

```

<%@ taglib uri="webwork" prefix="ww" %><html><head><title>WebWork Tutorial - Lesson
4.1.1 - Example 1</title><meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1"><style type="text/css">
    .errorMessage { color: red; }
</style></head><body><p>UI Form Tags Example:</p><ww:form
action="'formProcessing.action'" method="'post'"><ww:checkbox name="'checkbox'"
label="'A checkbox'" fieldValue="'checkbox_value'" /><ww:file name="'file'"
label="'A file field'" /><ww:hidden name="'hidden'" value="'hidden_value'"
/><ww:label label="'A label'" /><ww:password name="'password'" label="'A password
field'" /><ww:radio name="'radio'" label="'Radio buttons'" list="{ 'One', 'Two',
'Three' }" />
    <ww:select name="'select'" label="'A select list'" list="{ 'One', 'Two',
'Three' }"
        emptyOption="true" />
    <ww:textarea name="'textarea'" label="'A text area'" rows="3" cols="40"
/><ww:textfield name="'textfield'" label="'A text field'" /><ww:submit value="'Send
Form'" /></ww:form></body></html>

```

HTML result after processing ex01-index.jsp:

```

<html><head><title>WebWork Tutorial - Lesson 4.1.1 - Example 1</title><style
type="text/css">
    .errorMessage { color: red; }
</style></head><body><p>UI Form Tags Example:</p><table>
<form
action="formProcessing.action" method="post" >

<tr><td valign="top" colspan="2"><table width="100%" border="0" cellpadding="0"
cellspacing="0"><tr><td valign="top">
<input type="checkbox"
name="checkbox"
value="checkbox_value"
/>
</td><td width="100%" valign="top"><span class="checkboxLabel">
A checkbox
</span></td></tr></table></td></tr><tr><td align="right" valign="top"><span
class="label">

A file field:
</span></td><td>

<input type="file"
name="file"
/>

</td></tr>

    <input
type="hidden"
name="hidden" value="hidden_value" />

```

```

<tr><td align="right" valign="top"><span class="label">

A label:
</span></td><td><label></label></td></tr><tr><td align="right" valign="top"><span
class="label">

A password field:
</span></td><td>

<input type="password"
name="password"

/>

</td></tr><tr><td align="right" valign="top"><span class="label">

Radio buttons:
</span></td><td>

<input
type="radio"
name="radio"
id="radioOne"
value="One" />
<label for="radioOne">One</label>

<input
type="radio"
name="radio"
id="radioTwo"
value="Two" />
<label for="radioTwo">Two</label>

<input
type="radio"
name="radio"
id="radioThree"
value="Three" />
<label for="radioThree">Three</label></td></tr><tr><td align="right"
valign="top"><span class="label">

A select list:
</span></td><td>

<select name="select"
>

<option value=""></option>

```

```

<option value="One"
>One</option>

<option value="Two"
>Two</option>

<option value="Three"
>Three</option></select></td></tr><tr><td align="right" valign="top"><span
class="label">

A text area:
</span></td><td>

<textarea name="textarea"
cols="40"
rows="3"
></textarea></td></tr><tr><td align="right" valign="top"><span class="label">

A text field:
</span></td><td>

<input type="text"
name="textfield"
/>

</td></tr><tr><td colspan="2"><div
align="right" ><input
type="submit"
value="Send Form" /></div></td></tr></form></table></body></html>

```

xwork.xml:

```

<!DOCTYPE xwork PUBLIC "-//OpenSymphony Group//XWork
1.0//EN" "http://www.opensymphony.com/xwork/xwork-1.0.dtd">

<xwork><!-- Include webwork defaults (from WebWork JAR). --><include
file="webwork-default.xml" /><!-- Configuration for the default package. --><package
name="default" extends="webwork-default"><action name="formProcessing"
class="lesson04_01_01.FormProcessingAction"><result name="input"
type="dispatcher">ex01-index.jsp</result><result name="success"
type="dispatcher">ex01-success.jsp</result><interceptor-ref
name="validationWorkflowStack" /></action></package></xwork>

```

FormProcessingAction.java:

```
package lesson04_01_01;

import com.opensymphony.xwork.ActionSupport;

public class FormProcessingAction extends ActionSupport {
    privateString checkbox;
    privateString file;
    privateString hidden;
    privateString password;
    privateString radio;
    privateString select;
    privateString textarea;
    privateString textfield;

    publicString getCheckbox() { return checkbox; }
    publicString getFile() { return file; }
    publicString getHidden() { return hidden; }
    publicString getPassword() { return password; }
    publicString getRadio() { return radio; }
    publicString getSelect() { return select; }
    publicString getTextArea() { return textarea; }
    publicString getTextfield() { return textfield; }

    public void setCheckbox(String checkbox) { this.checkbox = checkbox; }
    public void setFile(String file) { this.file = file; }
    public void setHidden(String hidden) { this.hidden = hidden; }
    public void setPassword(String password) { this.password = password; }
    public void setRadio(String radio) { this.radio = radio; }
    public void setSelect(String select) { this.select = select; }
    public void setTextarea(String textarea) { this.textarea = textarea; }
    public void setTextfield(String textfield) { this.textfield = textfield; }

    publicString execute() throws Exception {
        return SUCCESS;
    }
}
```

FormProcessingAction-validation.xml:

```
<!DOCTYPE validators PUBLIC "-//OpenSymphony Group//XWork Validator
1.0//EN" "http://www.opensymphony.com/xwork/xwork-validator-1.0.dtd">

<validators><field name="checkbox"><field-validator
type="requiredstring"><message>Please, check the
checkbox.</message></field-validator></field><field name="file"><field-validator
type="requiredstring"><message>Please select a
file.</message></field-validator></field><field name="password"><field-validator
type="requiredstring"><message>Please type something in the password
field.</message></field-validator></field><field name="radio"><field-validator
type="requiredstring"><message>Please select a radio
button.</message></field-validator></field><field name="select"><field-validator
type="requiredstring"><message>Please select an option from the
```

```
</field-validator></field><field name="textarea"><field-validator  
type="requiredstring"><message>Please type something in the text  
area.</message></field-validator></field><field name="textfield"><field-validator  
type="requiredstring"><message>Please type something in the text  
field.</message></field-validator></field></validators>
```

ex01-success.jsp:

```
<%@ taglib uri="webwork" prefix="ww" %><html><head><title>WebWork Tutorial - Lesson  
4.1.1 - Example 1</title></head><body><p>UI Form Tags Example  
result:</p><ul><li>checkbox: <ww:property value="checkbox" /></li><li>file:  
<ww:property value="file" /></li><li>hidden: <ww:property value="hidden"  
</li><li>password: <ww:property value="password" /></li><li>radio: <ww:property  
value="radio" /></li><li>select: <ww:property value="select" /></li><li>textarea:  
<ww:property value="textarea" /></li><li>textfield: <ww:property value="textfield"  
</li></ul></body></html>
```

Notice how much cleaner `ex01-index.jsp` is, compared to its HTML result. The default layout of the form components is a table layout, with the label on the left column and the field to the right. You can learn how to create your own layouts when we explain the template system, below.

Another thing to notice is the reference to the `validationWorkflowStack` in the action's configuration. This makes WebWork validate the parameters that are sent to our actions according to a configuration file we place in the same location as the action class – in our case, `FormProcessingAction-validation.xml` (see [Validation](#)). In case something is not valid, it prevents the action from executing and dispatches the request to the input result with error messages attached to each field (using the method `addFieldError(String fieldName, String errorMessage)`).

But don't worry about how the validation framework works for now. Run the example and try leaving some fields blank. You will see that the UI tags provide error messages that integrate with the validation framework and that's what we want to demonstrate here. This separation of concerns can help programmers and designers concentrate more on their part of the work.

Read more: [UI Tags](#)

Try the example!

Other UI Controls:

Besides the standard form controls that HTML designers are already familiar with, WebWork provides some other controls and also the ability to create a custom control. Let's take a look at the custom controls that are already provided by WebWork:

<code><ww:checkboxlist /></code>	Works just like the <code><ww:radio /></code> tag, but with check boxes instead of radio buttons. It gets the keys and values from a collection and creates a list of checkboxes, all with the same name.
<code><ww:combobox /></code>	Simulates a combo box, which is a control that mixes a selection list with a text field. It does this by placing a text field with a <code><select /></code> list right below it and a JavaScript code that fills the text field with the selection of the list every time it changes.
<code><ww:tabbedpane /></code>	<i>Help needed here.</i>
<code><ww:token /></code>	<i>Help needed here.</i>

Read more: [UI Tags](#)

The Template System:

WebWork uses the Velocity template system to render the actual HTML output for all UI tags. A default implementation of all templates has been included with the core distribution allowing users to use WebWork's UI tags "out of the box". Templates can be edited individually or replaced entirely allowing for complete customization of the resulting HTML output. In addition, the default template can be overridden on a per tag basis allowing for a very fine level of control. The default templates are located in the `webwork-2.1.1.jar` file under `/template/xhtmll`.

If you unpack `webwork-2.1.1.jar` and look under the `/template/xhtmll` directory you will see a bunch of velocity templates. Most of them correspond to a specific UI Tag, and those have the name of the tag they render. If you're familiar with Velocity, I recommend you analyse the template files to see what you're capable of doing with them. Since version 2.1, there's also a `/template/simple` directory, which is a simpler version of the HTML form controls (just the control, no table or label).

If you want to display your UI components in a different layout than the one that comes with WebWork, you can:

- Edit and replace the files in `/template/xhtmll` (repack the JAR or create the same directory structure somewhere else and make sure your container looks that path before the JAR);
- Change the location of the templates by editing the `webwork.ui.theme` property in `webwork.properties` (file that should be placed in the root of your classpath);
- Specifying the location of the templates for each tag individually using the theme or the template property. The former allows you to specify the directory where all templates are (thus, WebWork looks for templates with the same name as the ones in `/template/xhtmll`), while the latter allows you to indicate the exact template to be used for that component.

Read more: [Templates](#), [Themes](#)

The third approach is demonstrated in the example below. Note that, by default, the specified theme directory should be under `/template` and the specified template file should be under `/template/xhtmll`.

ex02.jsp:

```
<%@ taglib uri="webwork" prefix="ww" %><html><head><title>WebWork Tutorial - Lesson
4.1.1 - Example 2</title></head><body><p>Template Change Example:</p><p><ww:checkbox
name="checkbox" label="A checkbox" fieldValue="checkbox_value"
theme="mytheme" /></p><p><ww:textfield name="textfield" label="A text field"
template="mytextfield.vm" /></p></body></html>
```

/template/mytheme/checkbox.vm:

```
<div align="center">
  <input type="checkbox"
    name="`${webwork.htmlEncode($parameters.name)}`"
    value="`${webwork.htmlEncode($parameters.fieldValue)}`"
    #if ($parameters.nameValue) checked="checked" #end
    #if ($parameters.disabled == true) disabled="disabled" #end
    #if ($parameters.tabindex) tabindex="`${webwork.htmlEncode($parameters.tabindex)}`"
  #end
  #if ($parameters.onChange) onChange="`${webwork.htmlEncode($parameters.onChange)}`"
  #end
  #if ($parameters.id) id="`${webwork.htmlEncode($parameters.id)}`" #end
  /><br />
  `${webwork.htmlEncode($parameters.label)}`
</div>
```

/template/xhtml/mytextfield.vm:

```
<div align="center">
  <input type="text"
    name="`${webwork.htmlEncode($parameters.name)}`"
    #if ($parameters.size) size="`${webwork.htmlEncode($parameters.size)}`" #end
    #if ($parameters.maxLength)
maxLength="`${webwork.htmlEncode($parameters.maxLength)}`" #end
    #if ($parameters.nameValue) value="`${webwork.htmlEncode($parameters.nameValue)}`"
  #end
    #if ($parameters.disabled == true) disabled="disabled" #end
    #if ($parameters.readonly) readonly="readonly" #end
    #if ($parameters.onkeyup) onkeyup="`${webwork.htmlEncode($parameters.onkeyup)}`"
  #end
    #if ($parameters.tabindex) tabindex="`${webwork.htmlEncode($parameters.tabindex)}`"
  #end
    #if ($parameters.onChange) onChange="`${webwork.htmlEncode($parameters.onChange)}`"
  #end
    #if ($parameters.id) id="`${webwork.htmlEncode($parameters.id)}`" #end
  /><br />
  `${webwork.htmlEncode($parameters.label)}`
</div>
```

HTML result after processing ex02.jsp:

```
<html><head><title>WebWork Tutorial - Lesson 4.1.1 - Example
2</title></head><body><p>Template Change Example:</p><p><div align="center">
  <input type="checkbox"
    name="checkbox"
    value="checkbox_value"
  /><br />
  A checkbox
</div></p><p><div align="center">
```

```
<input type="text"
                                name="textfield"
/><br />
A text field
</div></p></body></html>
```

Try the example!

Building Customized UI Components:

There are some situations in which none of the UI Components that come bundled with WebWork fit your requirements. In this case, the recommended approach would be to create your own custom component. In this way, you keep your web page clean of layout and error-checking issues and also promote component reuse.

To create a custom component, just create a Velocity template for it, just like the ones that already exist. To place it in a web page, use the `<ww:component />` tag and specify the location of the template in its `template` parameter.

To pass parameters to be used by your template, use the `<ww:param />` tag (see [lesson 4.1](#)). The example below demonstrates the creation of a custom date field.

Read more: [UI Tags](#)

ex03.jsp:

```
<%@ taglib uri="webwork" prefix="ww" %><html><head><title>WebWork Tutorial - Lesson
4.1.1 - Example 3</title></head><body><p>Custom Component
Example:</p><p><ww:component template="datefield.vm"><ww:param name="'label'"
value="'Date'" /><ww:param name="'name'" value="'mydatefield'" /><ww:param
name="'size'" value="3" /></ww:component></p></body></html>
```

/template/xhtml/datefield.vm:

```
#set ($name = $parameters.get('name'))
#set ($size = $parameters.get('size'))
#set ($yearSize = $size * 2)

$parameters.get('label'):

```

HTML result after processing ex03.jsp:

```
<html><head><title>WebWork Tutorial - Lesson 4.1.1 - Example
3</title></head><body><p>Custom Component Example:</p><p>
Date:

```

Try the example!

[Previous Lesson](#) | [Next Lesson](#)

Lesson 4.2: Using Velocity with WebWork

There are two ways of using Velocity as the view.

- Using the `velocity` result-type to render velocity templates;
- Registering `WebWorkVelocityServlet` in your `web.xml` file to render Velocity templates accessed directly through browser requests.

To use the second approach, we have to modify `web.xml` and add a servlet and a servlet mapping for `WebWorkVelocityServlet`, as demonstrated below:

```
<servlet><servlet-name>velocity</servlet-name><servlet-class>com.opensymphony.webwork.views.velo
```

Read more: [xwork.xml](#)

Using `velocity` result-type means that Velocity templates can only be rendered through an action, i.e., request to `.vm` pages will not render the file and it will be returned as plain text. If you choose this approach, it's recommended that you place your Velocity files under `WEB-INF` so they become unaccessible.

Using `WebWorkVelocityServlet` means that Velocity templates can be rendered through requests to `.vm` pages. That also means that you should implement security checks in your templates so an user doesn't access it directly without going through an action first (if that is required).

No matter which approach you choose (and you can choose to use both at the same time), not only all the features from Velocity are available to you when you're writing templates, but also some other functionalities, specific of WebWork, are available. It is supposed that you are already familiar with Velocity, so we will focus only in the

WebWork-specific features. If that's not the case, please [get started with Velocity](#) before continuing.

The main feature of it is to provide easy access to objects that are on the Value Stack, which contains some things that WebWork provides to you automatically, because you may find them useful at some point. These are some of the things that are available in the value stack:

- The current `HttpServletRequest`;
- The current `HttpServletResponse`;
- The current `OgnlValueStack`;
- An instance of `OgnlTool`;
- All the properties of the current action class.

To access the objects in the value stack, all you have to do is use appropriate Velocity references:

- `$req` = `HttpServletRequest`;
- `$res` = `HttpServletResponse`;
- `$stack` = `OgnlValueStack`;
- `$ognl` = `OgnlTool`;
- `$name-of-property` = property of the current action class.

The example below does the same thing as the Hello example from [lesson 3](#), but now, using a Velocity template as the result. Notice that the `<property value="person" />` tag was replaced by the `$person` reference, which returns the same thing: a property from the action class. In this example we chose to use the *velocity result-type* approach.

xwork.xml:

```
<!DOCTYPE xwork PUBLIC "-//OpenSymphony Group//XWork
1.0//EN" "http://www.opensymphony.com/xwork/xwork-1.0.dtd">

<xwork><!-- Include webwork defaults (from WebWork JAR). --><include
file="webwork-default.xml" /><!-- Configuration for the default package. --><package
name="default" extends="webwork-default"><!-- Default interceptor stack.
--><default-interceptor-ref name="defaultStack" /><!-- Action: Lesson 4.2:
HelloAction using Velocity as result. --><action name="helloVelocity"
class="lesson03.HelloAction"><result name="error"
```

```
</result><result name="success"
type="velocity">ex01-success.vm</result></action></package></xwork>
```

HelloAction.java (same as lesson 3):

```
package lesson03;

import com.opensymphony.xwork.ActionSupport;

public class HelloAction extends ActionSupport {
    String person;
    publicString getPerson() {
        return person;
    }
    public void setPerson(String person) {
        this.person = person;
    }
    publicString execute() throws Exception {
        if ((person == null) || (person.length() == 0)) return ERROR;
        elsereturn SUCCESS;
    }
}
```

ex01-index.jsp (same as lesson 3):

```
<html><head><title>WebWork Tutorial - Lesson 3 - Example
2</title></head><body><p>What's your name?</p><form action="hello.action"
method="post"><p><input type="text" name="person" /><input type="submit"
/></p></form></body></html>
```

ex01-success.vm:

```
<html><head><title>WebWork Tutorial - Lesson 4.2 - Example 1</title></head><body>

Hello, $person

</body></html>
```

Try the example!

Using WebWork Tags from Velocity:

As you already know, when you switch from JSP to Velocity you lose the ability of using JSP Tags. But WebWork's Velocity Servlet provides a way of doing this through the use of #tag, #bodytag and #param velocimacros. The general syntax is:

```
#tag (name-of-tag list-of-attributes)
```

– or –

```
#bodytag (name-of-tag list-of-attributes)
    #param (key value)
    #param (key value)
    ...
#end
```

Let's revisit [lesson 4.1.1](#)'s form example to demonstrate the usage of the UI tags from velocity:

xwork.xml:

```
<!DOCTYPE xwork PUBLIC "-//OpenSymphony Group//XWork
1.0//EN" "http://www.opensymphony.com/xwork/xwork-1.0.dtd">

<xwork><!-- Include webwork defaults (from WebWork JAR). --><include
file="webwork-default.xml" /><!-- Configuration for the default package. --><package
name="default" extends="webwork-default"><!-- Default interceptor stack.
--><default-interceptor-ref name="defaultStack" /><!-- Actions: Lesson 4.2:
FormProcessingAction using Velocity. --><action name="formProcessingVelocityIndex"
class="lesson04_02.FormProcessingIndexAction"><result name="success"
type="velocity">ex02-index.vm</result></action><action name="formProcessingVelocity"
class="lesson04_01_01.FormProcessingAction"><result name="input"
type="velocity">ex02-index.vm</result><result name="success"
type="velocity">ex02-success.vm</result><interceptor-ref
name="validationWorkflowStack" /></action></package></xwork>
```

ex02-index.vm:

```
<html><head><title>WebWork Tutorial - Lesson 4.2 - Example 2</title><style
type="text/css">
    .errorMessage { color: red; }
</style></head><body><p>UI Form Tags Example using Velocity:</p>

#bodytag (Form "action='formProcessingVelocity.action'" "method='post'")
    #tag (Checkbox "name='checkbox'" "label='A
checkbox'" "fieldValue='checkbox_value'")
    #tag (File "name='file'" "label='A file field'")
```

```

#tag (Hidden "name='hidden'" "value='hidden_value'")
#tag (Label "label='A label'")
#tag (Password "name='password'" "label='A password field'")
#tag (Radio "name='radio'" "label='Radio buttons'" "list={'One', 'Two', 'Three'}")
#tag (Select "name='select'" "label='A select list'" "list={'One', 'Two',
'Three'}" "emptyOption=true")
#tag (Textarea "name='textarea'" "label='A text area'" "rows='3'" "cols='40'")
#tag (TextField "name='textfield'" "label='A text field'")
#tag (Submit "value='Send Form'")
#end

</body></html>

```

ex02-success.vm:

```

<html><head><title>WebWork Tutorial Lesson 4.2 - Example 2</title></head><body><p>UI
Form Tags Example result using Velocity:</p><ul><li>checkbox:
$!checkbox</li><li>file: $!file</li><li>hidden: $!hidden</li><li>password:
$!password</li><li>radio: $!radio</li><li>select: $!select</li><li>textarea:
$!textarea</li><li>textfield: $!textfield</li></ul></body></html>

```

FormProcessingAction.java (same as lesson 4.1.1):

```

package lesson04_01_01;

import com.opensymphony.xwork.ActionSupport;

public class FormProcessingAction extends ActionSupport {
    privateString checkbox;
    privateString file;
    privateString hidden;
    privateString password;
    privateString radio;
    privateString select;
    privateString textarea;
    privateString textfield;

    publicString getCheckbox() { return checkbox; }
    publicString getFile() { return file; }
    publicString getHidden() { return hidden; }
    publicString getPassword() { return password; }
    publicString getRadio() { return radio; }
    publicString getSelect() { return select; }
    publicString getTextArea() { return textarea; }
    publicString getTextField() { return textfield; }

    public void setCheckbox(String checkbox) { this.checkbox = checkbox; }
    public void setFile(String file) { this.file = file; }
    public void setHidden(String hidden) { this.hidden = hidden; }
    public void setPassword(String password) { this.password = password; }
    public void setRadio(String radio) { this.radio = radio; }
    public void setSelect(String select) { this.select = select; }
    public void setTextArea(String textarea) { this.textarea = textarea; }
    public void setTextField(String textfield) { this.textfield = textfield; }
}

```

```

    publicString execute() throws Exception {
        return SUCCESS;
    }
}

```

FormProcessingAction-validation.xml (same as lesson 4.1.1):

```

<!DOCTYPE validators PUBLIC "-//OpenSymphony Group//XWork Validator
1.0//EN" "http://www.opensymphony.com/xwork/xwork-validator-1.0.dtd">

<validators><field name="checkbox"><field-validator
type="requiredstring"><message>Please, check the
checkbox.</message></field-validator></field><field name="file"><field-validator
type="requiredstring"><message>Please select a
file.</message></field-validator></field><field name="password"><field-validator
type="requiredstring"><message>Please type something in the password
field.</message></field-validator></field><field name="radio"><field-validator
type="requiredstring"><message>Please select a radio
button.</message></field-validator></field><field name="select"><field-validator
type="requiredstring"><message>Please select an option from the
list.</message></field-validator></field><field name="textarea"><field-validator
type="requiredstring"><message>Please type something in the text
area.</message></field-validator></field><field name="textfield"><field-validator
type="requiredstring"><message>Please type something in the text
field.</message></field-validator></field></validators>

```

Try the example!

The example above does not use the `#param` tag. So, let's revisit another example from [lesson 4.1.1](#) - custom components:

ex03.vm:

```

<html><head><title>WebWork Tutorial - Lesson 4.2 - Example
3</title></head><body><p>Custom Component Example:</p><p>
#bodytag (Component "template=datefield.vm")
    #param ("label" "Date")
    #param ("name" "mydatefield")
    #param ("size" "3")
#end
</p></body></html>

```

/template/xhtml/datefield.vm (same as lesson 4.1.1):

```
#set ($name = $parameters.get('name'))
#set ($size = $parameters.get('size'))
#set ($yearSize = $size * 2)

$parameters.get('label'):

```

Notice that, this time, we did not enclose `Date` and `mydatefield` with single quotes, as we had to do when we used the JSP tag.

Try the example!

[Previous Lesson](#) | [Next Lesson](#)

- `$stack` = `OgnlValueStack`;
- `$webwork` = `FreemarkerWebWorkUtil`, a toolbox providing services like formatting url, accessing the value stack, etc;
- `$name-of-property` = property retrieved from the value stack. If that fails, it looks up an attribute with that name in the `HttpServletRequest`, `HttpSession` and `ServletContext`, in that order;
- `$Request` = `HttpServletRequest`;
- `$Session` = `HttpServletResponse`;
- `$Application` = `OgnlValueStack`.

The example below does the same thing as example 2 from [lesson 3](#), but now, using Freemarker templates.

xwork.xml:

```
<!DOCTYPE xwork PUBLIC "-//OpenSymphony Group//XWork
1.0//EN" "http://www.opensymphony.com/xwork/xwork-1.0.dtd">

<xwork><!-- Include webwork defaults (from WebWork JAR). --><include
file="webwork-default.xml" /><!-- Configuration for the default package. --><package
name="default" extends="webwork-default"><!-- Default interceptor stack.
--><default-interceptor-ref name="defaultStack" /><!-- Action: Lesson 4.3:
HelloAction. --><action name="indexFreemarker"
class="com.opensymphony.xwork.ActionSupport"><result name="success"
type="dispatcher">/WEB-INF/ftl/lesson3/index.ftl</result></action><action
name="helloFreemarker" class="lesson03.HelloAction"><result name="error"
type="dispatcher">/WEB-INF/ftl/lesson3/index.ftl</result><result name="success"
type="dispatcher">/WEB-INF/ftl/lesson3/success.ftl</result></action></package></xwork>
```

HelloAction.java (same as lesson 3):

```
package lesson03;

import com.opensymphony.xwork.ActionSupport;

public class HelloAction extends ActionSupport {
    String person;
    public String getPerson() {
        return person;
    }
    public void setPerson(String person) {
        this.person = person;
    }
    public String execute() throws Exception {
        if ((person == null) || (person.length() == 0)) return ERROR;
        else return SUCCESS;
    }
}
```

ex02-index.ftl

```
<#assign ww=JspTaglibs["/WEB-INF/lib/webwork.tld"] /><html><head><title>WebWork  
Tutorial - Lesson 4.3 - Example 1</title></head><body><p>Click <a  
href="${wwUtil.buildUrl('indexFreemarker.action')}">here</a> to reload this  
page.</p><@ww.form name="nameForm" action="helloFreemarker.action"  
method="POST"><@ww.textfield label="What is your name ?" name="person"  
value="person" size="20"/><@ww.submit name="submit"  
value="Submit"/></@ww.form></body></html>
```

If you don't want to use WebWork's UI Tags, you could do it like this:

ex02-index-notags.ftl

```
<html><head><title>WebWork Tutorial - Lesson 4.3 - Example  
1</title></head><body><p>Click <a  
href="${wwUtil.buildUrl('indexFreemarker.action')}">here</a> to reload this  
page.</p><form name="nameForm" action="${wwUtil.buildUrl('helloFreemarker.action')}"  
method="POST">  
    What is your name ?  
    <input type="text" name="person" value="${person}" size="20"><input  
type="submit" name="submit" value="Submit"></form></body></html>
```

However, if you choose not to use tags, it's recommended that you use Freemarker Macros to write the form elements.

ex02-success.ftl:

```
<#assign ww=JspTaglibs["/WEB-INF/lib/webwork.tld"] /><html><head><title>WebWork  
Tutorial - Lesson 4.3 - Example 1</title></head><body>  
  
Come from the property WW tag (taglibs support) : <@ww.property value="person"/><br>  
Come from the Freemarker lookup in the WW stack : ${person}  
  
</body></html>
```

You can use either WebWork `property` tag or the Freemarker `$person` reference. Both of them return the same thing: a property from the action class.

[Previous Lesson](#) | [Next Lesson](#)

Lesson 5: Interceptors

Interceptors allow arbitrary code to be included in the call stack for your action before and/or after processing the action, which can vastly simplify your code itself and provide excellent opportunities for code reuse. Many of the features of XWork and WebWork are implemented as interceptors and can be applied via external configuration along with your own Interceptors in whatever order you specify for any set of actions you define.

In other words, when you access a *.action URL, WebWork's `ServletDispatcher` proceeds to the invocation of the an action object. Before it is executed, however, the invocation can be intercepted by another object, that is hence called interceptor. To have an interceptor executed before (or after) a given action, just configure `xwork.xml` properly, like the example below, taken from [lesson 4.1.1](#):

Interceptor configuration from lesson 4.1.1:

```
<action name="formProcessing" class="lesson04_01_01.FormProcessingAction"><result
name="input" type="dispatcher">ex01-index.jsp</result><result name="success"
type="dispatcher">ex01-success.jsp</result><interceptor-ref
name="validationWorkflowStack" /></action>
```

As you can see, lesson 4.1.1's `formProcessing` Action uses the `validationWorkflowStack`. That is an interceptor stack, which organizes a bunch of interceptors in the order in which they are to be executed. That stack is configured in `webwork-default.xml`, so all we have to do to use it is declare a `<interceptor-ref />` under the action configuration or a `<default-interceptor-ref />`, under package configuration, as seen in [lesson 3](#)'s first example:

Interceptor configuration from lesson 3.1:


```
<!DOCTYPE xwork PUBLIC "-//OpenSymphony Group//XWork
1.0//EN" "http://www.opensymphony.com/xwork/xwork-1.0.dtd">

<xwork><!-- Include webwork defaults (from WebWork JAR). --><include
file="webwork-default.xml" /><!-- Configuration for the default package. --><package
name="default" extends="webwork-default"><!-- Default interceptor stack.
--><default-interceptor-ref name="defaultStack" /><!-- Action: Lesson 03:
HelloWebWorldAction. --><action name="helloWebWorld"
class="lesson03.HelloWebWorldAction"><result name="success"
type="dispatcher">ex01-success.jsp</result></action></package></xwork>
```

But let's see how it works from scratch:

1. Create an interceptor class, which is a class that implements the `com.opensymphony.xwork.interceptor.Interceptor` interface (bundled in `xwork-1.0.jar`);
2. Declare the class in your XML configuration file (`xwork.xml`) using the element `<interceptor />` nested within `<interceptors />`;
3. Create stacks of interceptors, using the `<interceptor-stack />` element (*optional*);
4. Determine which interceptors are used by which action, using `<interceptor-ref />` or `<default-interceptor-ref />`. The former defines the interceptors to be used in a specific action, while the latter determines the default interceptor stack to be used by all actions that do not specify their own `<interceptor-ref />`.

Looking inside `webwork-default.xml` we can see how it's done:

webwork-default.xml:

```
<!DOCTYPE xwork PUBLIC "-//OpenSymphony Group//XWork
1.0//EN" "http://www.opensymphony.com/xwork/xwork-1.0.dtd">

<xwork><package name="webwork-default"><result-types>
  <result-type name="dispatcher" default="true"
    class="com.opensymphony.webwork.dispatcher.ServletDispatcherResult"/>
  <result-type name="redirect"
    class="com.opensymphony.webwork.dispatcher.ServletRedirectResult"/>
  <result-type name="velocity"
    class="com.opensymphony.webwork.dispatcher.VelocityResult"/>
  <result-type name="chain"
    class="com.opensymphony.xwork.ActionChainResult"/>
  <result-type name="xslt"
    class="com.opensymphony.webwork.views.xslt.XSLTResult"/>
</result-types><interceptors>
  <interceptor name="timer"
    class="com.opensymphony.xwork.interceptor.TimerInterceptor"/>
  <interceptor name="logger"
    class="com.opensymphony.xwork.interceptor.LoggingInterceptor"/>
```

```

<interceptor name="chain"
  class="com.opensymphony.xwork.interceptor.ChainingInterceptor"/>
<interceptor name="static-params"
  class="com.opensymphony.xwork.interceptor.StaticParametersInterceptor"/>
<interceptor name="params"
  class="com.opensymphony.xwork.interceptor.ParametersInterceptor"/>
<interceptor name="model-driven"
  class="com.opensymphony.xwork.interceptor.ModelDrivenInterceptor"/>
<interceptor name="component"
  class="com.opensymphony.xwork.interceptor.component.ComponentInterceptor"/>
<interceptor name="token"
  class="com.opensymphony.webwork.interceptor.TokenInterceptor"/>
<interceptor name="token-session"
  class="com.opensymphony.webwork.interceptor.TokenSessionStoreInterceptor"/>
<interceptor name="validation"
  class="com.opensymphony.xwork.validator.ValidationInterceptor"/>
<interceptor name="workflow"
  class="com.opensymphony.xwork.interceptor.DefaultWorkflowInterceptor"/>
<interceptor name="servlet-config"
  class="com.opensymphony.webwork.interceptor.ServletConfigInterceptor"/>
<interceptor name="prepare"
  class="com.opensymphony.xwork.interceptor.PrepareInterceptor"/>
<interceptor name="conversionError"
  class="com.opensymphony.webwork.interceptor.WebWorkConversionErrorInterceptor"/>
<interceptor-stack name="defaultStack"><interceptor-ref
name="static-params"/><interceptor-ref name="params"/><interceptor-ref
name="conversionError"/></interceptor-stack><interceptor-stack
name="validationWorkflowStack"><interceptor-ref
name="defaultStack"/><interceptor-ref name="validation"/><interceptor-ref
name="workflow"/></interceptor-stack></interceptors></package></xwork>

```

Since we included `webwork-default.xml` in our `xwork.xml`, all the interceptors and stacks above are available for us to use in our actions. Here's what these interceptors do:

- **timer**: clocks how long the action (including nested interceptors and view) takes to execute;
- **logger**: logs the action being executed;
- **chain**: makes the previous action's properties available to the current action. Used to make action chaining (reference: [Result Types](#));
- **static-params**: sets the parameters defined in `xwork.xml` onto the action. These are the `<param />` tags that are direct children of the `<action />` tag;
- **params**: sets the request (POST and GET) parameters onto the action class. We have seen an example of this in [lesson 3](#);
- **model-driven**: if the action implements `ModelDriven`, pushes the `getModel()` result onto the Value Stack;
- **component**: enables and makes registered components available to the actions. (reference: [\[IoC & Components\]](#));
- **token**: checks for valid token presence in action, prevents duplicate form submission;
- **token-session**: same as above, but storing the submitted data in session when handed an invalid token;

- **validation:** performs validation using the validators defined in {Action}-validation.xml (reference: [Validation](#)). We've seen an example of this in [lesson 4.1.1](#);
- **workflow:** calls the validate method in your action class. If action errors created then it returns the INPUT view. Good to use together with the validation interceptor (reference: [Validation](#));
- **servlet-config:** give access to HttpServletRequest and HttpServletResponse (think twice before using this since this ties you to the Servlet API);
- **prepare:** allows you to programmatic access to your Action class before the parameters are set on it.;
- **conversionError:** *help needed here.*

Building your own Interceptor

If none of the above interceptors suit your particular need, you will have to implement your own interceptor. Fortunately, this is an easy task to accomplish. Suppose we need an interceptor that places a greeting in the Session according to the time of the day (morning, afternoon or evening). Here's how we could implement it:

GreetingInterceptor.java:

```
package lesson05;

import java.util.Calendar;
import com.opensymphony.xwork.interceptor.Interceptor;
import com.opensymphony.xwork.ActionInvocation;

public class GreetingInterceptor implements Interceptor {
    public void init() { }
    public void destroy() { }
    public String intercept(ActionInvocation invocation) throws Exception {
        Calendar calendar = Calendar.getInstance();
        int hour = calendar.get(Calendar.HOUR_OF_DAY);
        String greeting = (hour < 6) ? "Good evening" :
            ((hour < 12) ? "Good morning" :
            ((hour < 18) ? "Good afternoon" : "Good evening"));

        invocation.getInvocationContext().getSession().put("greeting", greeting);

        String result = invocation.invoke();

        return result;
    }
}
```

xwork.xml:

```
<!DOCTYPE xwork PUBLIC "-//OpenSymphony Group//XWork
1.0//EN" "http://www.opensymphony.com/xwork/xwork-1.0.dtd">

<xwork><!-- Include webwork defaults (from WebWork JAR). --><include
file="webwork-default.xml" /><!-- Configuration for the default package. --><package
name="default" extends="webwork-default"><interceptors><interceptor name="greeting"
class="section02.lesson05.GreetingInterceptor" /></interceptors><!-- Action: Lesson
5: GreetingInterceptor. --><action name="greetingAction"
class="lesson05.GreetingAction"><result name="success"
type="velocity">ex01-result.vm</result><interceptor-ref name="greeting"
/></action></package></xwork>
```

GreetingAction.java:

```
package lesson05;

import com.opensymphony.xwork.ActionSupport;

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

ex01-result.vm:

```
<html><head><title>WebWork Tutorial - Lesson 5 - Example 1</title></head><body>

#set ($ses = $req.getSession())
<p><b>${ses.getAttribute('greeting')}!</b></p></body></html>
```

Let's take a look at our interceptor class first. As explained before, the interceptor must implement `com.opensymphony.xwork.interceptor.Interceptor`'s methods: `init()`, called during interceptor initialization, `destroy()`, called during destruction, and most importantly, `intercept(ActionInvocation invocation)`, which is where we place the code that does the work.

Notice that our interceptor returns the result from `invocation.invoke()` which is the method responsible for executing the next interceptor in the stack or, if this is the last one, the action. This means that the interceptor has the power of short-circuiting the action invocation and return a result string without executing the action at all! Use this

with caution, though.

One other thing that interceptors can do is execute code after the action has executed. To do that, just place code after the `invocation.invoke()` call. WebWork provides an abstract class that already implements this kind of behaviour: `com.opensymphony.xwork.interceptor.AroundInterceptor`. Just extend it and implement the methods `before(ActionInvocation invocation)` and `after(ActionInvocation dispatcher, String result)`.

The `xwork.xml` configuration, the action class and the result page are pretty straightforward and require no further explanation.

Try the example!

[Previous Lesson](#) | End of Tutorial