


Space Details

Key:	XW
Name:	XWork
Description:	
Creator (Creation Date):	plightbo (Apr 18, 2004)
Last Modifier (Mod. Date):	plightbo (Apr 25, 2004)

Available Pages

- XWork 
 - Documentation
 - Basics
 - Components
 - Configuration
 - Logging
 - Core Concepts
 - Dependencies
 - Interceptors
 - DefaultWorkflowInterceptor
 - PrepareInterceptor
 - ValidationInterceptor
 - Introduction
 - Localization
 - Ognl
 - Previous releases
 - Upgrading from 1.0
 - Upgrading from 1.0.1
 - Upgrading from 1.0.2
 - Upgrading from 1.0.3
 - Upgrading from 1.0.4
 - XWork 1.0.1
 - XWork 1.0.2
 - XWork 1.0.3
 - XWork 1.0.4
 - XWork 1.0.5
 - XWork 1.0.6
 - Type Conversion
 - Null Property Access
 - Type Conversion Error Handling
 - Type Conversion In Collections
 - Upgrading from 1.1
 - Validation Framework

- Building a Validator
 - Custom Action Validation
 - ExpressionValidator Tips
 - Generic Object Validation
 - Sample Validation Rules
 - Standard Validators
- XWork layers
- Press Releases
 - 1.0.1 Press Release
 - 1.0.2 Press Release
 - 1.0.3 Press Release
 - 1.0.4 Press Release
 - 1.0.5 Press Release
 - 1.1.2 Press Release
 - 1.2.0 Press Release
 - About
- Sandbox
 - XML Invocation
- XWork-1_2
 - Building XWork1_2
 - XWork1_2 Actions
 - XWork1_2 Annotations
 - XWork1_2 Architecture
 - XWork1_2 Articles
 - XWork1_2 Configurations
 - XWork1_2 Conversion
 - XWork1_2 FAQs
 - XWork1_2 Features
 - XWork1_2 Hibernate Integration
 - XWork1_2 Installation
 - XWork1_2 Interceptors
 - XWork1_2 Object Factory
 - XWork1_2 package
 - XWork1_2 PreResultListeners
 - XWork1_2 Profiling
 - XWork1_2 Reloading
 - XWork1_2 Requirements And Dependencies
 - Xwork1_2 Results
 - Xwork1_2 Spring Integration
 - XWork1_2 Tutorial
 - XWork1_2 Validation
 - XWork1_2 Value Stack
- XWork2
 - Annotations
 - AnnotationWorkflowInterceptor

- Building XWork
- Reloading
- XWork Actions
- XWork Architecture
- XWork Articles
- XWork Configurations
- XWork Conversion
- XWork FAQs
- XWork Features
- XWork Hibernate Integration
- XWork Installation
- XWork Interceptors
- XWork Object Factory
- XWork package
- XWork PreResultListeners
- XWork Profiling
- XWork Requirements And Dependencies
- Xwork Results
- Xwork Spring Integration
- XWork Tutorial
 - XWork2 Hello World Tutorial
- XWork Validation
- XWork Value Stack

XWork

This page last changed on Nov 22, 2006 by [tm_je](#).

XWork is a generic command pattern framework. It was split out of WebWork 1.x and forms the core of [WW:WebWork](#) 2.0. It features:

- Flexible and customizable configuration based on a simple Configuration interface, allowing you to use XML , programmatic, or even product-integrated configuration
- Core command pattern framework which can be customized and extended through the use of interceptors to fit any request / response environment
- Built in type conversion and action property validation using [OGNL](#)
- Powerful validation framework based on runtime attributes and a validation interceptor

Useful links:

- [Documentation](#)
 - [API Javadocs](#)
- [Press Releases](#)
- [Download Binaries](#)
- [CVS](#)
- [XWork2](#)
- [XWork\1\ 2](#)

In addition to the documentation, it might be useful to see [Rickard Oberg's thoughts](#) on the future directions of XWork, especially as it relates to Portal development

XWork-Optional

XWork optional is a repository where optional modules for XWork may be created. Just about anyone can get developers access to this repository and is encouraged to use this area as a staging zone for developing projects based upon XWork. Currently a mail dispatcher project is there which uses a mail folder to dispatch actions for processing the emails. For download instructions code go to <https://xwork-optional.dev.java.net/servlets/ProjectSource>

Documentation

This page last changed on Apr 11, 2006 by [mrdon](#).



These documents are very out of date. As the majority of XWork users are also WebWork users, we've decided to put all our effort in to enhancing the WebWork documentation. As such, these documents may be out of date. However, as of XWork 1.1, a large amount of JavaDocs have been added to XWork. We hope they are useful to you. You can also read the WebWork documentation, as it may provide useful elements about XWork

Overview

- [Introduction](#)
 - What is Xwork?
 - How does Webwork 2.0 relate to Xwork?

Reference Guide

- [Core Concepts](#): Terminology and an introduction to XWork
- [XWork layers](#)
- [Configuration](#): xwork.xml
- [Ognl](#)
- [Localization](#)
- [Type Conversion](#)
- [Interceptors](#)
- [Validation Framework](#)
- [Components](#): Inversion of Control

XWork Versions

- Current Release
 - Release Notes
 - [XWork 1.0.6](#)
 - [XWork 1.0.5](#)
 - [XWork 1.0.4](#)
 - [XWork 1.0.3](#)
 - [XWork 1.0.2](#)
 - [XWork 1.0.1](#)
 - [Dependencies](#)
- Upgrading from previous versions
 - [Upgrading from 1.1](#)
 - [Upgrading from 1.0.4](#)
 - [Upgrading from 1.0.3](#)
 - [Upgrading from 1.0.2](#)
 - [Upgrading from 1.0.1](#)
 - [Upgrading from 1.0](#)

Documentation Tasks Remaining

- Merge XWork specific docs from WebWork. In particular:
 - config docs
- Description for all interceptors
- Beef up [Basics](#)
- Make sure we document ActionChaining somewhere

Actions

Actions are the basic unit of execution...

The Action Interface

The basic interface which all XWork actions must implement. It provides several standard return values like SUCCESS and INPUT, and only contains one method:

```
public String execute() throws Exception;
```

In general, Actions should simply extend ActionSupport, which provides a default implementation for the most common actions.

ActionContext

The ActionContext provides access to the execution environment in the form of named objects during an Action invocation. A new ActionContext is created for each invocation allowing developers to access/modify these properties in a thread safe manner. The ActionContext makes a number of properties available that are typically set to appropriate values by the framework. In WebWork 2 for example, the ActionContext session map wraps an underlying HttpSession object. This allows access to environment specific properties without tying the core framework to a specific execution environment.

The ActionContext is acquired through the static ActionContext.getContext() method. The ActionContext is a thread local variable and thus the properties of the ActionContext will be relative to the current request thread. The ActionContext has several methods for commonly used properties as well as get() and set() methods which can be used for application specific properties.

Overview

[XWork](#) provides the `ComponentManager` interface (and a corresponding implementation in the `DefaultComponentManager` class) to allow a design pattern known as **Inversion of Control** (or **IoC** for short) to be applied to your actions or other arbitrary objects. In a nutshell, the IoC pattern allows a parent object (in this case XWork's `ComponentManager` instance) to control a client object (usually an action, but it could be any object that implements the appropriate *enabler*).

You may also want to look at [WW:Components](#) to see how WW2 uses XWork's IoC architecture.

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()`)

XWork takes care of passing components through to enabled action classes or other components.

Some other benefits that IoC can provide include:

- 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.

- 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).
- 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'.
- 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.
- 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).

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 [XW: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.

Writing Component Classes

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

Configuration

This page last changed on Apr 25, 2005 by [rgielen](#).

- `xwork.xml`
- [Logging](#)

xwork.xml

XWork is configured through the use of a file named `xwork.xml` in the root of the classpath which conforms to the DTD. This file defines the action, interceptor, and result configurations and mappings. The following is a sample `xwork.xml` file; it's a condensed version of the one used in the XWork test cases. It's a long example, but it exhibits all the major configuration possibilities:

```
<!DOCTYPE xwork PUBLIC "-//OpenSymphony Group//XWork 1.0//EN"
"http://www.opensymphony.com/xwork/xwork-1.0.dtd">

<xwork>
  <package name="default">
    <result-types>
      <result-type name="chain" class="com.opensymphony.xwork.ActionChainResult"/>
    </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="params"
class="com.opensymphony.xwork.interceptor.ParametersInterceptor"/>
      <interceptor name="static-params"
class="com.opensymphony.xwork.interceptor.StaticParametersInterceptor"/>
      <interceptor name="test" class="com.opensymphony.xwork.TestInterceptor">
        <param name="foo">expectedFoo</param>
      </interceptor>

      <interceptor-stack name="defaultStack">
        <interceptor-ref name="static-params"/>
        <interceptor-ref name="params"/>
      </interceptor-stack>
      <interceptor-stack name="debugStack">
        <interceptor-ref name="timer"/>
        <interceptor-ref name="logger"/>
      </interceptor-stack>
    </interceptors>

    <global-results>
      <result name="login" type="chain">
        <param name="actionName">login</param>
      </result>
    </global-results>

    <action name="Foo" class="com.opensymphony.xwork.SimpleAction">
      <param name="foo">17</param>
      <param name="bar">23</param>
      <result name="success" type="chain">
        <param name="actionName">Bar</param>
      </result>
      <interceptor-ref name="debugStack"/>
      <interceptor-ref name="defaultStack"/>
    </action>

    <action name="Bar" class="com.opensymphony.xwork.SimpleAction">
      <param name="foo">17</param>
      <param name="bar">23</param>
    </action>
  </package>
</xwork>
```

```

<action name="TestInterceptorParam" class="com.opensymphony.xwork.SimpleAction">
  <interceptor-ref name="test">
    <param name="expectedFoo">expectedFoo</param>
  </interceptor-ref>
</action>

<action name="TestInterceptorParamOverride" class="com.opensymphony.xwork.SimpleAction">
  <interceptor-ref name="test">
    <param name="foo">fool23</param>
    <param name="expectedFoo">fool23</param>
  </interceptor-ref>
</action>
</package>

<package name="bar" extends="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>
</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="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>

<include file="includeTest.xml"/>
</xwork>

```

xwork.xml elements

Package

The package element has one required attribute, "name", which acts as the key to later reference 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. ([see above for example](#))

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
abstract	no	declares package to be abstract (no action configurations required in package)

Namespace

The optional namespace attribute warrants its own discussion section. 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. ([see above for example](#))

Result-Type

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.

Interceptors

Interceptors also serve as a name-value pairing for referring to specific Interceptor classes by a shorthand name where we use interceptor-ref elements, such as in Interceptor stacks and action configurations.

Interceptor-Stack

The interceptor-stack element allows you to assign a name for later referencing via interceptor-ref to an ordered list of interceptors. This allows you to create standard groups of interceptors to be used in a certain order. The interceptor stack name/value pairs are stored in the same map as the interceptor definitions, so anywhere you use an interceptor-ref to refer to an interceptor you may also refer to an interceptor stack.

Interceptor Parameterization

Interceptors are instantiated and stored at the ActionConfig level, i.e. there will be one instance per

action configured in xwork.xml. These Interceptor instances may be parameterized at either the declaration level, where you map an Interceptor class to a name to be referenced later, like so:

```
<interceptor name="test" class="com.opensymphony.xwork.TestInterceptor">
  <param name="foo">expectedFoo</param>
</interceptor>
```

or at the interceptor-ref level, either inside an interceptor-stack or in an action declaration, like so:

```
<interceptor-ref name="test">
  <param name="expectedFoo">expectedFoo</param>
</interceptor-ref>
```

Although it is allowed by the DTD, parameters applied to interceptor-refs which refer to interceptor-stack elements will NOT be applied, and will cause a warning message.

Global-results

The 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.

Action

The action element allows the mapping of names to action classes. These names are used, along with the namespace described above, to retrieve the action config from the configuration. The action may also be parameterized by using the param elements, as above, which will set parameters into the Action at execution (with the help of the StaticParametersInterceptor).

When defining actions, sometimes you actually don't need any class, but rather are just putting in a placeholder for accessing interceptors and results. You can use ActionSupport for a simple action that has all of the XWork features but doesn't actually do anything. In fact, rather than defining your action class, if you leave it empty it defaults to ActionSupport.

```
<action name="foo">
  ...
</action>
```

The action may also have one or more results mapped to string return codes, such as "success", "error", or "input", which are the default 3 return states for actions, although ANY return string may be used and mapped to a result. The result, which is mapped to a result class by the "type" attribute which refers to the result-type name defined above, may also be parameterized by using the param element.

There is one shortcut when defining results, as a lot of the time a result will have only one parameter (which is often a location to redirect to (when using XWork in the web sense)).

Here is the long form of a result with a single location parameter:

```
<result name="test">
  <param name="location">foo.jsp</param>
</result>
```


and this is the 'shortcut' form:

```
<result name="test">foo.jsp</result>
```

Note that this shortcut **only** works when there is a single parameter for the result and the result has defined what its default parameter is.

If your result name is "success", then you can take this shortcut even further by just doing:

```
<result>success.jsp</result>
```

 **Tip:** using the action class shorthand and the result shortcuts, you can do something as simple as this to define an XWork front-end to a result:

```
<action name="foo">
  <result>foo.jsp</result>
</action>
```

Similar to the interceptor Syntax shown above, actions may be parametrized:

```
<action name="Foo" class="com.opensymphony.xwork.SimpleAction">
  <param name="foo">17</param>
  ...
</action>
```

In this example, it would be tried to call the setFoo() method in SimpleAction with the given value when Foo.action is called. If SimpleAction is model driven, setFoo() is invoked on the model.

The action may also define one or more interceptor refs to either interceptors or interceptor stacks to be applied before and after the action execution (see the section on Interceptors below). The ordering of these interceptor refs determines the order in which they will be applied.

Includes - using multiple configuration files

The xwork.xml configuration file may be broken up into several files, each with its own set of package declarations, by using the <include> element zero or more times at the bottom of your xwork.xml file, like so:

```
<include file="includeTest.xml"/>
```

These files will be processed, in order, in the same manner as the main xwork.xml, thus may have all of

the same structures, including the `<include>` element. Although it is, therefore, possible to have packages in configuration files listed later in an include element extend packages defined in an earlier included file, it is not recommended. It should be considered best practice, in the case of using multiple configuration files, to declare default packages meant to be extended in the `xwork.xml` and to have no dependencies between sub-configuration files.

Configuration Providers

XWork configuration is handled through classes which implement the `ConfigurationProvider` interface. The default implementation is the `XmlConfigurationProvider` class. You can either create a new provider by implementing the `ConfigurationProvider` interface or you can extend the `XmlConfigurationProvider` class. The `XmlConfigurationProvider` class includes a protected method called `getInputStream()` which is called to acquire the configuration `InputStream` which is expected to be an XML data stream. The default implementation looks for a file called `xwork.xml` in the class path but by overriding the `getInputStream()` method you can pull configuration data from any source.

Custom configuration providers must be registered with the `ConfigurationManager` before they will be used to load configuration data. If no custom configuration providers are registered then the default configuration provider is used. If any custom configuration providers are registered then the default configuration provider will no longer be used (although you could add a new instance of it yourself to include it in the list of providers which is searched). To add a configuration provider just call the `ConfigurationManager.addConfigurationProvider()` method with the custom configuration provider as the argument.

Logging

This page last changed on Jun 05, 2004 by [unkyaku](#).

Logging in XWork is handled by [Commons-Logging](#). If Log4J is present in the classpath, logging tasks will be passed through to Log4J.

The logger names are the class names. The pattern used is:

```
Log log = LogFactory.getLog(ThisClass.class);
```

For details on configuring commons-logging, see
<http://jakarta.apache.org/commons/logging/guide.html#Configuration>.

XWork Core Concepts

XWork is based on a number of core concepts that helps to explain how the framework works. The core concepts can be broken down into two parts: Architecture Concepts and Terminology.

Architecture Concepts

- Explain Command Driven Architecture (in general)
- Explain the implementation in XWork

Terminology

Actions

Actions are classes that get invoked in response to a request, execute some code and return a Result. Actions implement at a minimum a single method, `execute()`, that defines the entry point called by the framework. This method allows developers to define a unit of work that will be executed each time the Action is called.

ActionContext

The `ActionContext` provides access to the execution environment in the form of named objects during an Action invocation. A new `ActionContext` is created for each invocation allowing developers to access/modify these properties in a thread safe manner. The `ActionContext` makes a number of properties available that are typically set to appropriate values by the framework. In WebWork 2 for example, the `ActionContext` session map wraps an underlying `HttpSession` object. This allows access to environment specific properties without tying the core framework to a specific execution environment. For more information, see `ActionContext` in [Basics](#).

Interceptors

In XWork, 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 [Interceptors](#) for further details.

Stacks

To handle the case where developers want to apply more than a single Interceptor to an Action, Stacks have been introduced. Stacks are an ordered list of Interceptors and/or other Stacks that get applied when an Action is invoked. Stacks centralize the declaration of Interceptors and provide a convenient way to configure multiple actions.

Results

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.

Result Types

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.

Packages

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.

ValueStack

The ValueStack is a stack implementation built on top of an OGNL core. The OGNL expression language can be used to traverse the stack and retrieve the desired object. The OGNL expression language provides a number of additional features including: automatic type conversion, method invocation and object comparisons. For more information, see the [OGNL Website](#).

Components

XWork provides the ComponentManager interface (and a corresponding implementation in the DefaultComponentManager class) to apply a design pattern known as **Inversion of Control** (or IoC for short). In a nutshell, the IoC pattern allows a parent object (in this case XWork's ComponentManager instance) to control a client object (usually an action, but it could be any object that implements the appropriate *enabler*). See [Components](#) for further details.

Dependencies

This page last changed on Aug 26, 2004 by [plightbo](#).

Dependencies are split into runtime and compile time dependencies.

- Runtime dependencies are stored in CVS in the **lib/core** directory.
- Compile time dependencies are stored in CVS in the **lib/build** directory.

You can see all the dependencies as well as their version numbers by looking at the **libraries.txt** file located in each directory.

Interceptors

This page last changed on Aug 18, 2004 by [unkyaku](#).

- Overview
- [PrepareInterceptor](#)
- [ValidationInterceptor](#)
- [DefaultWorkflowInterceptor](#)

Overview

Interceptors allow you to define code to be executed before and/or after the execution of an action. They are defined outside the action class, yet have access to the action and the action execution environment at runtime, allowing you to encapsulate cross-cutting code and provide separation of concerns.

Interceptors are given the `ActionInvocation` object at runtime, and may do whatever processing needed, then forward processing to the rest of the `ActionInvocation`, which will either call the next `Interceptor` or the `Action`, if there are no more `Interceptors`, and do whatever post-processing needed.

Interceptors may also decide to short-circuit processing and return whatever result string desired WITHOUT forwarding processing, thus keeping the `Action` from executing. This ability should be used with caution, however, as any data loading or processing expected to be done by the `Action` will not happen.

Here is the `invoke()` method from `ActionInvocation`, which calls the `Interceptors` and the `Action`:

```
public String invoke() throws Exception {
    if (executed) {
        throw new IllegalStateException("Action has already executed");
    }

    if (interceptors.hasNext()) {
        Interceptor interceptor = (Interceptor) interceptors.next();
        result = interceptor.intercept(this);
    } else {
        result = action.execute();
        executed = true;
    }

    return result;
}
```

It may not be immediately apparent how the rest of the `Interceptors` and the `Action` come to be called from the code snippet. For this we need to look at the `Interceptor` implementation in

AroundInterceptor:

```
public String intercept(ActionInvocation invocation) throws Exception {
    before(invocation);

    result = invocation.invoke();
    after(invocation);

    return result;
}
```

Here we can see that the `Interceptor` calls back into the `ActionInvocation.invoke()` to tell the

ActionInvocation to continue down the chain and eventually executes the Action. It is here that the Interceptor can decide not to forward to the rest of the Interceptors and the Action, and choose instead to return a return code.

It is also important to know what the AroundInterceptor is doing when you extend it to implement your own Interceptors.

The AroundInterceptor defines a base class for interceptor implementations. It delegates calls to subclasses, which must implement the abstract methods before() and after(). The before() call is first called, then the rest of the ActionInvocation is called and the String result is saved (and is available to the Interceptor implementation during the after() method). Finally, the after() method is called and the result is returned.

 Note that all Interceptor implementations must be threadsafe.

Utility Interceptors

The TimerInterceptor and LoggingInterceptor are provided as simple examples and utilities.

- The **LoggingInterceptor** simply logs before and after executing the rest of the ActionInvocation.
- The **TimerInterceptor** times the execution of the remainder of the ActionInvocation.

The TimerInterceptor does not extend AroundInterceptor because it needs to keep some state (the start time) from before the rest of the execution. Interceptors must be stateless, so it is impossible to save this in an instance field. It is a good rule of thumb to say that if your interceptor needs to maintain information from the beginning to the end of the interceptor call, it should implement Interceptor directly, not subclass AroundInterceptor. Here is the code for **intercept()** in TimerInterceptor:

```
public String intercept(ActionInvocation dispatcher) throws Exception {
    long startTime = System.currentTimeMillis();
    String result = dispatcher.invoke();
    long executionTime = System.currentTimeMillis() - startTime;
    log.info("Processed action " + dispatcher.getProxy().getActionName() + " in " +
        executionTime + "ms.");

    return result;
}
```

It is important to remember to call **invoke()** on the ActionInvocation if you directly implement Interceptor, otherwise the rest of the Interceptors and the Action will not be executed.

Parameter Interceptors - populating your Action

The StaticParametersInterceptor and ParametersInterceptor populate your Action fields during the ActionInvocation execution.

- The **StaticParametersInterceptor** applies the parameters defined in the Action configuration with the <param> elements.
- The **ParametersInterceptor** populates the Action with the parameters passed in as part of the

request.

The `StaticParametersInterceptor` should be applied before the `ParametersInterceptor` so that the static parameters may be set as the defaults and overridden by the request parameters.

ModelDrivenInterceptor - choosing your model

Normally, the **StaticParameterInterceptor** and the **ParametersInterceptor** apply themselves directly to the Action. Using the `ModelDrivenInterceptor`, you can specify an alternate object to have the parameters applied to instead.

Consider the following Action:

```
public class AddContactAction implements Action {
    private String name;
    private String addr;
    private String city;

    public void setName(String name) { this.name = name ; }
    public void setAddr(String addr) { this.addr = addr ; }
    public void setCity(String city) { this.city = city ; }

    public String execute() {
        Contact contact = new Contact();
        contact.setName(name);
        contact.setAddr(addr);
        contact.setCity(city);

        // save contact information here
    }
}
```

We can see that our action will be populated with name, addr, and city parameters if they are passed in. In the execute we copy these values to a contact object and save the contact.

Here's the `ModelDriven` interface:

```
public interface ModelDriven {
    public Object getModel();
}
```

Let's apply the `ModelDriven` interface to Action above:

```
public class AddContactAction implements Action, ModelDriven {
    private Contact contact = new Contact();

    public Object getModel() { return this.contact ; }

    public void execute() {
        // save the contact information
    }
}
```

Now the `ParametersInterceptor` and the `StaticParametersInterceptor` will be applied directly to our `Contact` so when `execute` gets called, `this.contact` will already be populated with all the information we need. Neat, huh?

Behavior similar to model driven can be achieved just using the parameter interceptor. For example, rather than implementing `ModelDriven`, we could have written:

```
public class AddContactAction implements Action {
    private Contact contact = new Contact();

    public Contact getContact { return this.contact ; }

    public void execute() {
        // save the contact information
    }
}
```

The difference between this `Action` and the previous `ModelDriven` action is twofold:

- Using the `ModelDriven` action, we can reference our parameters `name`, `addr`, and `city`. Also, the `Model` (`Contact`) will be pushed onto the `ValueStack` so we'll have `Contact` and `AddContactAction` on the value stack
- When not using the `ModelDriven` action, we need to reference our parameters as `contact.name`, `contact.addr`, and `contact.city`.

One potential drawback when using `ModelDriven` actions is that if you need to access some parameters in order to load the model for the `ModelDriven` action, you will need to call the `ParametersInterceptor` and/or the `StaticParametersInterceptor` twice (before and after the `ModelDrivenInterceptor`). The first time sets all parameters on the `Action`, the second time sets all parameters on the `model`.

ChainingInterceptor

The **ChainingInterceptor** populates the `Action` it is being applied to with the results of the previous action. When actions are chained together, the action being chained FROM is on the `ValueStack` when the next action is called. This means that when the next `ActionProxy` is executed, the action that is being chained TO will be placed onto the `valuestack`, but the old action will also be there, just down one level. This interceptor works by traversing the `ValueStack` to find the parameters of any objects there and sets them onto the final action.

DefaultWorkflowInterceptor

This page last changed on Jun 26, 2004 by [unkyaku](#).

```
<interceptor name="workflow"
class="com.opensymphony.xwork.interceptor.DefaultWorkflowInterceptor"/>
```

This interceptor implements the workflow that was found in ActionSupport in WebWork 1.x. The following workflow steps are applied before the rest of the Interceptors and the Action are executed, and may short-circuit their execution:

1. If the Action implements **com.opensymphony.xwork.Validateable**, the **validate()** method is called on it, to allow the Action to execute any validation logic coded into it.
2. If the Action implements **com.opensymphony.xwork.ValidationAware** the **hasErrors()** method is called to check if the Action has any registered error messages (either Action-level or field-level). If the ValidationAware Action has any errors, **Action.INPUT** ("input") is returned without executing the rest of the Interceptors or the Action.
3. If the execution did not short-circuit in the step above, the rest of the Interceptors and the Action are executed by calling **invocation.invoke()**

PrepareInterceptor

This page last changed on Jun 27, 2004 by [unkyaku](#).

```
<interceptor name="prepare" class="com.opensymphony.xwork.interceptor.PrepareInterceptor"/>
```

This interceptor watches for Actions that implement **com.opensymphony.xwork.Preparable** and calls the **prepare()** method on it.

ValidationInterceptor

This page last changed on Jun 26, 2004 by [unkyaku](#).

```
<interceptor name="validation" class="com.opensymphony.xwork.validator.ValidationInterceptor"/>
```

This interceptor validates the Action it is applied to. See the documentation on the [Validation Framework](#) for details.

Introduction

This page last changed on Jun 03, 2004 by [plightbo](#).

XWork is a generic command pattern framework.

The Purpose:

- To create a generic, reusable, and extensible command pattern framework not tied to any particular usage.

Features:

- Flexible and customizable configuration based on a simple Configuration interface
- Core command pattern framework which can be customized and extended through the use of interceptors to fit any request / response environment
- Built in type conversion and action property validation using Ognl
- Powerful validation framework based on runtime attributes and a validation interceptor

How does XWork relate to Webwork

Webwork 2.0+ is built on top of XWork and provides web-specific features that allow you to quickly build web applications using XWork's command pattern and interceptor framework.

Localization

This page last changed on Jul 13, 2004 by [unkyaku](#).

Any action can indicate that it supports localization by implementing `com.opensymphony.xwork.TextProvider`. To access a localized message, simply use one of the various `getText()` method calls.

The default implementation for this is `com.opensymphony.xwork.TextProviderSupport`, which in turn relies on `com.opensymphony.xwork.util.LocalizedTextUtil`. Any Action that extends `com.opensymphony.xwork.ActionSupport` will automatically gain localization support via `TextProviderSupport`.

In this implementation, when you attempt to look up a message, it attempts to do the following:

- Look for the message in the Action's class hierarchy.
 - Look for the message in a resource bundle for the class
 - If not found, look for the message in a resource bundle for any interface implemented by the class
 - If not found, get the super-class and repeat from the first sub-step unless the super-class is `Object`
- If not found and the Action is a `ModelDriven` Action, then look for the message in the model's class hierarchy (repeat sub-steps listed above).
- If not found, look for the message in a child property. This is determined by evaluating the message key as an OGNL expression. For example, if the key is `user.address.state`, then it will attempt to see if "user" can be resolved into an object. If so, repeat the entire process from the beginning with the object's class and `address.state` as the message key.
- If not found, look for the message in the Action's package hierarchy.
- If still not found, look for the message in the default resource bundles.

Default Resource Bundles.

It is possible to register default resource bundles with XWork via `LocalizedTextUtil.addDefaultResourceBundle()`.

Message lookup in the default resource bundles is done in reverse order of their registration (i.e. the first resource bundle registered is the last to be searched).

By default, one default resource bundle name is registered with `LocalizedTextUtil` – `"com/opensymphony/xwork/xwork-messages"` – which is bundled with the XWork jar file to provide system-level message texts.

Example

Given a ModelDriven Action called BarnAction where getModel() returns a Horse object, and the Horse object has the following class structure:

```
interface acme.test.Animal;
class acme.test.AnimalImpl implements Animal;
interface acme.test.Quadrapped extends Animal;
class acme.test.QuadrappedImpl extends Animal implements Quadrapped;
class acme.test.Horse extends QuadrappedImpl;
```

Then the localization system will attempt to look up the message in the following resource bundles in this order:

```
acme.test.BarnAction.properties
acme.test.Horse.properties
acme.test.QuadrappedImpl.properties
acme.test.Quadrapped.properties
acme.test.AnimalImpl.properties
acme.test.Animal.properties
acme.test.package.properties
acme.package.properties
```

Message Key Interpolation

When looking for the message, if the key indexes a collection (e.g. user.phone0) and a message for that specific key cannot be found, the general form will also be looked up (i.e. user.phone*).

Message Interpolation

If a message is found, it will also be interpolated. Anything within `${...}` will be treated as an OGNL expression and evaluated as such.

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 the additional language features that are provided on top of the base OGNL EL.

XWork-specific Language Features

The ValueStack

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. XWork, however, 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()
```


Note that the ValueStack is essentially a List. Calling [1] on the stack returns a sub-stack 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 ValueStack that consists of a model and an action ([model, action]). Here's how the following OGNL expressions 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 ValueStack
              // (excluding the CompoundRoot) that supports the toString() method
\[1\].foo    // call getFoo() on the first object in the ValueStack starting from action
              // (excluding the CompoundRoot) that supports a getFoo() method
```

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" (short for ValueStack) prefix:

```
@vs@FOO_PROPERTY
@vs@someMethod()

@vs1@FOO_PROPERTY
@vs1@someMethod()

@vs2@BAR_PROPERTY
@vs2@someOtherMethod()
```

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.

The *top* keyword

XWork also adds a new keyword – **top** – that can be used to access to first object in the ValueStack.

Previous releases

This page last changed on Dec 15, 2004 by [cameronbraid](#).

- Development Version [XWork 1.0.6](#)
- Current release - [XWork 1.0.5](#)
- Previous releases - information on previous releases of XWork
 - [XWork 1.0.4](#)
 - [XWork 1.0.3](#)
 - [XWork 1.0.2](#)
 - [XWork 1.0.1](#)
- Dependencies

Upgrading from 1.0

This page last changed on Dec 15, 2004 by [cameronbraid](#).

Upgrading to XWork 1.0.1 from 1.0 involves very little work. All you need to do is copy over the new xwork-1.0.1.jar in replace of xwork-1.0.jar and make sure that the new [Dependencies](#) are all in place.

Upgrading from 1.0.1

This page last changed on Dec 15, 2004 by [cameronbraid](#).

Upgrading to XWork 1.0.2 from 1.0.1 involves very little work. All you need to do is copy over the new xwork-1.0.2.jar in replace of xwork-1.0.1.jar and make sure that the new [Dependencies](#) are all in place.

Upgrading from 1.0.2

This page last changed on Dec 15, 2004 by [cameronbraid](#).

Upgrading from 1.0.2 is as simple as dropping in the new jar. There are no new dependencies or code changes. This release is simply a re-release of 1.0.2 with the correct packaging.

Upgrading from 1.0.3

This page last changed on Dec 15, 2004 by [cameronbraid](#).

Upgrading to XWork 1.0.4 from 1.0.3 involves very little work. All you need to do is copy over the new xwork-1.0.4.jar in replace of xwork-1.0.3.jar and make sure that the new [Dependencies](#) are all in place.

Upgrading from 1.0.4

This page last changed on Dec 15, 2004 by [cameronbraid](#).

Upgrading to XWork 1.0.5 from 1.0.4 is mostly backwards compatible. However, there is one change you should be aware of:

ParametersInterceptor changes

ParametersInterceptor has been modified to not allow any parameter names to be used that contain any special OGNL characters, such as "#", ",", and "=". This fixes a security hole, but if you were depending on it we recommend that you figure out an alternative way to do what you are doing rather than depend on this vulnerability. Upgrading to 1.0.5 fixes this issue.

XWork 1.0.1















This page last changed on Dec 15, 2004 by [cameronbraid](#).

XWork 1.0.1

Key Changes

- Introduction of an ObjectFactory that provides for easy integration to libraries like Spring and Pico
- Added actionMessages support – just like errorMessages but not counted as an error
- Major performance improvements with Ognl 2.6.5

Changelog

OpenSymphony JIRA (19 issues)		
T	Key	Summary
	XW-188	Correct logging level in DefaultActionInvocation.invokeAction(..)
	XW-183	NPE thrown when trying to set a sub-property of a property that doesnt exist
	XW-182	NPE thrown by LocalizedTextUtil.findText
	XW-167	XWorkBasicConverter conversion from Date to String is not localized
	XW-166	[Patch] Improve support for ModelDriven interface
	XW-165	[PATCH] VisitorFieldValidator not setting fieldError correctly
	XW-164	[PATCH] TypeConverter should check class hierarchy for conversion properties
	XW-163	[PATCH] Added some javadocs to Interceptors
	XW-162	TypeConverter created by ObjectFactory
	XW-161	Replaceable ObjectFactory for creating framework objects to allow easier integration with IoC containers
	XW-160	Add infinite recursion detection to the ChainingInterceptor
	XW-159	XWork build is broken
	XW-158	Email and URL Validators adding error messages for empty fields
	XW-157	Further Optimize Validator Lookup in ActionValidatorManager



[XW-156](#)

[add actionMessages support](#)



[XW-155](#)

[NPE thrown when invalid method is looked up](#)



[XW-117](#)

[Additional methods to determine if errors using JSTL](#)



[XW-56](#)

[Add localization support to XWorkConverter](#)



[XW-23](#)











[Investigate using runtime attributes to configure interceptors](#)

XWork 1.0.2

Key Changes

- Added an **xwork-default.xml** configuration file that can be used as a standard starting place for non-web related XWork deploys (such as in Quartz)
- Localized text for collections can now be referenced in the form **someArray[*].someField**, where * provides a "catch-all" for any index element
- Text retrieved using the default TextProvider now looks attempts to look for messages in the property files of child objects (if applicable) if none can be found.
- **Field-level validation can be short-circuited**, meaning that you can stop the validation chain right away if you wish. See [Validation Framework](#) for details.
- Simple **type conversion, such as int and float, now use the Locale**. This will cause the correct use of "," and "." depending on the locale.
- Much improved handling for type conversion of elements in collections.

Changelog

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



[XW-191](#)

[XW-190](#)

[XW-189](#)

[XW-179](#)

[XW-172](#)

[XWork validation DTD with short circuit](#)

[Type conversion improvement.](#)

[Provide a xwork-default.xml.](#)

[Improve ActionValidationManager's short circuit behaviour](#)

[Optimise OgnlUtil.copy method](#)

[XWorkBasicConverter doesn't care about the current locale](#)

XWork 1.0.3


This page last changed on Dec 15, 2004 by [cameronbraid](#).

XWork 1.0.3

Key Changes

- Fixed critical problem where xwork-validator-1.0.dtd wasn't in the distribution.

Changelog






OpenSymphony JIRA (1 issues)		
T	Key	Summary
	XW-214	xwork-validator-1.0.dtd not in xwork.jar

XWork 1.0.4 Release Notes

Key Changes

This release primarily involves a few internal tweaks and API convenience methods. It is backwards compatible with 1.0.3 and should have no impact when upgrading.

Changelog

OpenSymphony JIRA (5 issues)		
T	Key	Summary
	XW-240	Added convenience methods to ComponentManager
	XW-220	OgnlValueStack.findValue doesnt use custom converter when converting to String.class
	XW-219	xwork.xml should allow no namespaces
	XW-215	upload newest xwork jars to ibiblio servers
	XW-174	ObjectFactory requires same action to return same classname

XWork 1.0.5 Release Notes








Key Changes

- Moderate security vulnerability resolved
- Configuration defaults to ActionSupport for the action if you don't define a class
- Configuration defaults to "success" for the result if you don't define a name
- Minor fixes for i18n-related issues

Migration Notes

Version	Description	Old Code	New Code
1.0.4 and below	Parameters interceptor behavior changed		

Changelog

OpenSymphony JIRA (7 issues)		
T	Key	Summary
	XW-254	Security problem with ParametersInterceptor
	XW-244	Default action class and result name
	XW-241	Allow ObjectFactory impls that support Actions without no-arg constructors
	XW-225	Whitespace in xml configuration
	XW-224	XMLConfigurationProvider should not fail if it cannot load a result type
	XW-222	OgnlException while setting property 'fieldName'
	XW-180	Validation XML file loading does not work with CGLIB proxy class

XWork 1.0.6 Release Notes

Key Changes

- Reworked instantiation of Validators, such that they are now thread safe (no changes visible outside of the framework were made).
- Distribution now includes a source zip which can be used by your IDE to allow you to browse the source.

Migration Notes

Version	Description	Old Code	New Code
---------	-------------	----------	----------

Changelog

OpenSymphony JIRA (0 issues)		
T	Key	Summary

Type Conversion

This page last changed on Jul 21, 2004 by [unkyaku](#).

- Overview
- [Null Property Access](#)
- [Type Conversion Error Handling](#)

Type conversion allows you to easily convert objects from one class to another. Whenever XWork attempts to get or set a property on an Action or object and the Class of the value to be get/set is not what is expected, XWork will try to convert it to the correct class.

A common use case is when submitting forms on the web, where we are usually converting to and from String classes. A good example is date conversion. Let's suppose we have the following class:

```
public class Foo implements Action {
    private Date date;
    public void setDate(Date date) { this.date = date ; }
    public String execute() {
        // do work here
    }
}
```

XWork's type converter would allow us to pass in the String "07/08/2003" and have it be automatically converted to a Date object that's been set to July 8, 2003.

XWork already supports basic type conversion, but you can also use your own type converters. To define a custom type converter, you'll need to perform the following steps:

- Create your custom type converter object by implementing `ognl.TypeConverter` or extending `ognl.DefaultTypeConverter`
- Define your conversion rules in a file named **className-conversion.properties** or add the appropriate entry to **xwork-conversion.properties**

Defining Conversion Rules

Conversion rules can be defined at both the class level as well as the application level.

Class-specific conversion rules

To define a class-specific conversion rules, create a file named **className-conversion.properties**. The syntax for these files are:

```
property = full.class.name.of.converter
Collection_property = full.class.name
Map_property = full.class.name
```

This file defines the type converter to use for properties of an Action or object. If a property is a Collection (List only) or Map, you can also specify the type of elements in the Collection/Map. Please note

that there are some issues to consider when working with [type conversion in collections](#).

When looking for a type converter, XWork will search up the class hierarchy and directly implemented for conversion rules. Given the following class hierarchy:

```
interface Animal;
class AnimalImpl implements Animal;
interface Quadraped extends Animal;
class QuadrapedImpl extends AnimalImpl implements Quadraped;
```

XWork will look for a conversion rule in the following files in the following order:

```
QuadrapedImpl-conversion.properties
Quadraped-conversion.properties
AnimalImpl-conversion.properties
Animal-conversion.properties
```

Application-wide conversion rules

To define application-wide conversion rules, create a file named **xwork-conversion.properties** and place it in the root classpath. The syntax for this file is:

```
full.class.name = full.class.name.of.converter
```

Whenever XWork sees any classes listed in this file, it will use the specified converter to convert values to that class.

When looking up a default converter, XWork will search up the class hierarchy to find the appropriate converter. Given the class hierarchy above, XWork will look for a conversion rule for the class in the following order:

```
QuadrapedImpl
AnimalImpl
```

An Example

To use the Contact example (where Contact is a persistent object), let's say we have the following Action:

```
public class AddContactAction implements Action {
    private Contact contact1;
    private Contact contact2;
    public void setContact1(Contact contact) { this.contact1 = contact; }
    public void setContact2(Contact contact) { this.contact2 = contact; }

    public String execute() { ... }
}
```

What we're expecting from the UI is for contact to be "1", the primary key of the contact. We want the

type converter to convert the string "1" to the Contact with a contactId of 1. Likewise, we'd like the converter to be able to reverse the operation. When displayed, a Contact object should print out its contactId as a String.

The first step is to create our custom TypeConverter:

```
public class ContactConverter extends ognl.DefaultTypeConverter {
    public Object convertValue(Map ognlContext, Object value, Class toType) {
        if( toType == String.class ) {
            Contact contact = (Contact)value;
            return new String(contact.getContactId());
        } else if( toType == Contact.class ) {
            Integer id = new Integer((String)value);
            Session session = ... // get a Hibernate Session
            Contact contact = (Contact)session.load(Contact.class, id);
            return contact;
        }
        return null;
    }
}
```

The next part is to bind our ContactConverter to the previous AddContactAction. I'll bind the ContactConverter to the AddContactAction by creating a file called AddContactAction-conversion.properties that's in the same directory as the AddContactAction class.

I would then populate the properties file as follows:

```
contact1 = com.acme.ContactConverter
contact2 = com.acme.ContactConverter
```

Now, when XWork attempts to populate your object from parameters, you'll be given the actual instances of Contact from your database.

Having said all that, I can't really recommend doing database lookups here as a best practice. In fact, I'd say it's not such a good idea, but it does illustrate type converters well 😊 Any exception thrown here will be handled as described in [Type Conversion Error Handling](#).

Null Property Access

This page last changed on Jun 24, 2004 by [unkyaku](#).

Null property access is a unique feature to XWork that allows object graphs to be created at runtime, saving you the headache of having to pre-initialize them.

Simple object graphs

The feature is quite simple: **only** during the ParametersInterceptor (for WebWork this would be when http parameters are applied to an action), if an expression being set, such as "document.title", results in a NullPointerException, XWork will attempt to create the null object and try again. So in this case, if "document" is null, WebWork will construct a new Document object so that setting the title will succeed.

This is very useful because it reduces the amount of flattening (and unflattening) you are required to do in your action classes when displaying and setting data from web pages. Rather, you can usually represent the complete object graph by just naming your input fields with well thought-out names (like "document.title").

Collections, Lists and Maps

XWork extends this feature even further by offering special support for Collections, Lists and Maps. If you are providing input for one of these interfaces, XWork can be told what type of objects they will hold and automatically populate them accordingly. What this means is that if you refer to "children0.name", XWork will automatically create a new List to hold the children and also add an empty Child object to that list, so that setting a name on the expression "children0.name" will work correctly. The same goes for maps.

For this to work you must tell the converters what the objects in the List or Map will be. You do this by specifying "Collection_**property** = com.acme.CollectionItem" in the conversion properties file. So if you have an action that has a "children" property that should be filled with Child objects, YourAction-conversion.properties should contain:

```
Collection_children = come.acme.Child
```

For the purposes of conversion, WebWork considers Collections and Lists to be the same. If the "children" property was declared to be a Collection, a List would be created and used.

Custom Objects

Advanced users who need to instantiate custom objects will want to extend **com.opensymphony.xwork.ObjectFactory** and override the buildBean(Class) method. The default implementation is rather trivial:


```
public Object buildBean(Class clazz) throws Exception {
    return clazz.newInstance();
}
```

Once you have your own custom `ObjectFactory`, you'll need to let `XWork` know to use it. You do this by calling `ObjectFactory.setObjectFactory(yourObjFactory)`.

Type Conversion Error Handling

This page last changed on Nov 11, 2004 by [plightbo](#).

Type conversion errors are handled by the `XWorkConverter` whenever any `Exception` is thrown by a converter during converting a value. Type conversion errors are added to a `Map` stored in the `ActionContext` which is available via `ActionContext.getContext().getConversionErrors()`. This map is a map of field name to values which will allow you to access the original value which failed conversion.

 Tip: Reporting type conversion errors can be done in two ways:

- On a per-field basis with the conversion validator
- Globally via the conversion interceptor

The default text for conversion errors isn't exactly pretty: *Invalid field value for field "xxx"*. You can change this by defining an `i18n` property named **`invalid.fieldvalue.[propertyName]`**, such as `invalid.fieldvalue.foo`.

There are 2 ways of automatically populating field errors with the type conversion errors into the field errors of the Action. The first will populate all of the field errors from the conversion errors and is implemented as an Interceptor. There are actually 2 Interceptors, one in `XWork` and one in `WebWork` which extends the one in `XWork`. They differ in the implementation of the method

```
protected boolean shouldAddError(String propertyName, Object value)
```

The `XWork` version always returns `true`, whereas the `WebWork` Interceptor returns `false` for values of `null`, `""`, and `{""}`, preventing type conversion exceptions for these common empty values from propagating to the field errors of the Action. The `WebWork` version of this Interceptor has been added to the `webwork-defaults.xml` and to the `defaultStack` defined therein.

If you choose not to use this Interceptor, you can choose to enable this on a per-field basis by using the Validation framework with the new field validator added for this, defined in the `validators.xml` file like this:

```
<validator name="conversion"
class="com.opensymphony.xwork.validator.validators.ConversionErrorFieldValidator"/>
```

This validator will look up the conversion errors and, if it finds a conversion error for the field it is applied to, it will add the appropriate error message to the Action.

Both of these methods use the

```
XWorkConverter.getConversionErrorMessage(propertyName, stack)
```

which looks up the type conversion error message for the property name as was done previously, by looking for a message keyed by **`invalid.fieldvalue.[propertyName]`** and using a default value if it is not found.

Type Conversion In Collections

This page last changed on Apr 15, 2005 by [jsmecham](#).

There is a limit to how much type conversion XWork can do automatically when it works with collections.

Internally, XWork uses XWorkList and XWorkMap. They basically extend ArrayList and HashMap respectively to support type conversion. Anything that gets inserted into them are automatically converted if they're not already the required type.

In situations where the collection is null, XWork will create either an XWorkList or an XWorkMap and set it back on to the target (see [Null Property Access](#) for details), and everything works beautifully. However, if the collection is not null, and the collections returned are neither an XWorkList nor an XWorkMap, there's no way type conversion can be performed. In this case, if you want type conversion to work, you will have to seriously consider using XWorkList or XWorkMap instead of ArrayList or HashMap.

Example

Imagine a User object that has a Collection of e-mail addresses. Now, suppose the input is a String array ["foo@bar.com"], and the target is "emails[1]". What we'd really like XWork to do is convert that String array into a String before inserting it into the collection. The first thing we'd do is create a User-conversion.properties with the line:

```
Collection_emails = java.lang.String
```

If User.getEmails() returns null, then XWork will create a new XWorkList and call User.setEmails() with it. It will then add an empty string at position [0] and insert "foo@bar.com" at position [1] (the array is automatically converted into a String).

If User.getEmails() returns something other than XWorkList, when XWork tries to set emails[1] to ["foo@bar.com"], no type conversion will occur. In addition, if the array is empty, ArrayList will throw an IndexOutOfBoundsException.

Upgrading from 1.1

This page last changed on Apr 11, 2006 by [mrdon](#).

These are issues a developer using XWork 1.1 will need to know when migrating to XWork 1.2:

- Configuration errors fail-fast - When XWork encounters errors loading xwork and validator configuration files, it will throw an exception rather than logging the error and continue processing. The goal is that errors are more apparent to the developer and not buried in a log file somewhere, but it does mean XWork is less tolerant. Please ensure all the appropriate files load correct and report any exceptions thrown that you feel shouldn't interrupt processing.

Validation Framework

This page last changed on Dec 19, 2005 by [mrdon](#).

- [Overview](#)
- [Standard Validators](#) - details on validators bundled with XWork
- [ExpressionValidator Tips](#) - tips for using the ExpressionValidator
- [Building a Validator](#) - how to create your own Validator
- [Generic Object Validation](#) - shows how to use the validators to validate any object
- [Custom Action Validation](#) - details how to specify custom validation code for an Action

Overview

The validation framework in XWork is designed to help you apply simple validation rules to your Actions before they are executed.

At its core, the framework takes just about any object and a String context name for which to validate that object. This allows you to have different validation rules for the same class in different contexts. You can define default validation rules in the class-level validation file (**ClassName-validation.xml**), and then define validation rules which are added on top of these for a specific context (**ClassName-contextName-validation.xml**). The validation rules are applied in the order they are listed in the validation files and error messages are saved into the Object (if it implements ValidationAware).

In the case of Action validation, which is what most XWork users will be doing, the class name is the Action class name and the context is the Action alias.

Registering Validators

Validation rules are handled by validators, which must be registered with the ValidatorFactory. This may either be done programmatically, using the registerValidator(String name, Class clazz) static method of the ValidatorFactory, or by adding a file named **validators.xml** to the root of the classpath that contains this information. The syntax for **validators.xml** is:

```
<validators>
  <validator name="required"
    class="com.opensymphony.xwork.validator.validators.RequiredFieldValidator"/>
  <validator name="requiredstring"
    class="com.opensymphony.xwork.validator.validators.RequiredStringValidator"/>
  <validator name="stringlength"
    class="com.opensymphony.xwork.validator.validators.StringLengthFieldValidator"/>
  <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"/>
    <validator name="regex"
        class="com.opensymphony.xwork.validator.validators.RegexFieldValidator"/>
</validators>

```



Useful Information

validators.xml if being defined should be available in the classpath. However this is not necessary, if no custom validator is needed. Webwork will automatically pick up a predefined sets of validators defined in com/opensymphony/xwork/validator/validators/default.xml packaged together in xwork jar file that comes with webwork distribution. See ValidatorFactory static block for details.



Warning

If custom validator is being defined and a validators.xml is created and place in the classpath, do remember to copy all the other pre-defined validators that is needed into the validators.xml as if not they will not be registered. Once a validators.xml is detected in the classpath, the default one (com/opensymphony/xwork/validator/validators/default.xml) will not be loaded. It is only loaded when a custom validators.xml cannot be found in the classpath. Be careful.

Turning on Validation

The process of applying validation rules to an Action before it is executed is handled by the ValidationInterceptor. As such, all that is required to enable validation for an Action is to declare the ValidationInterceptor in your XWork config file and add an interceptor-ref to it for your Action. See [XW:Configuration](#) for details on how to construct your XWork config file. Here's a simple example:

```

<xwork>
  <package name="example">
    <interceptors>
      <interceptor name="validator"
        class="com.opensymphony.xwork.validator.ValidationInterceptor"/>
    </interceptors>

    <action name="Foo" class="com.opensymphony.xwork.SimpleAction">
      <param name="foo">17</param>
      <param name="bar">23</param>
      <result name="success" type="chain">
        <param name="actionName">Bar</param>
      </result>
      <interceptor-ref name="validator"/>
    </action>
  </package>
</xwork>

```

Bear in mind that the ValidationInterceptor only performs validation. The Action will still be executed even if there are any validation errors. It kicks off the validation by delegating task to ActionValidatorManager.

Understanding Validator flavors

The validators supplied by the Xwork distribution (and any validators you might write yourself) come in two different flavors:

1. Plain Validators
2. FieldValidators

Plain Validators (such as the ExpressionValidator) perform validation checks that are not inherently tied to a single specified field. When you declare a plain Validator in your -validation.xml file you do not associate a *fieldname* attribute with it. (You should avoid using plain Validators within the <field-validator> syntax described below.)

FieldValidators (such as the EmailValidator) are designed to perform validation checks on a single field. They require that you specify a *fieldname* attribute in your -validation.xml file. There are two different (but equivalent) XML syntaxes you can use to declare FieldValidators (see "<validator> vs. <field-Validator> syntax" below).

There are two places where the differences between the two validator flavors are important to keep in mind:

1. when choosing the xml syntax used for declaring a validator (either <validator> or <field-validator>)
2. when using the short-circuit capability

Note that you do not declare what "flavor" of validator you are using in your -validation.xml file, you just declare the name of the validator to use and WebWork will know whether it's a "plain Validator" or a "FieldValidator" by looking at the validation class that the validator's programmer chose to implement.

Defining Validation Rules

To define validation rules for an Action, create a file named **ActionName-validation.xml** in the same package as the Action. You may also create alias-specific validation rules which add to the default validation rules defined in **ActionName-validation.xml** by creating another file in the same directory named **ActionName-aliasName-validation.xml**. In both cases, **ActionName** is the name of the Action class, and **aliasName** is the name of the Action alias defined in the **xwork.xml** configuration for the Action.

The framework will also search up the inheritance tree of the Action to find validation rules for directly implemented interfaces and parent classes of the Action. This is particularly powerful when combined with ModelDriven Actions and the [VisitorFieldValidator](#). Here's an example of how validation rules are discovered. Given the following class structure:

- interface Animal;
- interface Quadraped extends Animal;
- class AnimalImpl implements Animal;
- class QuadrapedImpl extends AnimalImpl implements Quadraped;
- class Dog extends QuadrapedImpl;

The framework method will look for the following config files if Dog is to be validated:

1. Animal
2. Animal-aliasname
3. AnimalImpl
4. AnimalImpl-aliasname
5. Quadraped
6. Quadraped-aliasname
7. QuadrapedImpl
8. QuadrapedImpl-aliasname
9. Dog
10. Dog-aliasname

While this process is similar to what the [XW:Localization](#) framework does when finding messages, there are some subtle differences. The most important difference is that validation rules are discovered from the parent downwards.



Take Note

Child's *-validation.xml will override parent's *-validation.xml according to the class hierarchy defined above.

Syntax for Validation Rules

Here is a sample config file containing validation rules for SimpleAction from the Xwork test cases:

```
<!DOCTYPE validators PUBLIC "-//OpenSymphony Group//XWork Validator 1.0.2//EN"
"http://www.opensymphony.com/xwork/xwork-validator-1.0.2.dtd">
<validators>
  <field name="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>
</validators>
```

```
<validator type="expression">
  <param name="expression">foo &gt; bar</param>
  <message>Foo must be greater than Bar. Foo = ${foo}, Bar = ${bar}</message>
</validator>
</validators>
```

All `<validator>` (and `<field-validator>`) elements must have a *type* attribute, which refers to a name of a Validator registered with the ValidatorFactory as described above. These elements may also have `<param>` elements with name and value attributes to set arbitrary parameters into the Validator instance.


All `<validator>` (and `<field-validator>`) elements must also define one **message** subelement, which defines the message that should be added to the Action if the validator fails. By default, the message will be that contained in the body of the message tag. The **message** element also has one optional attribute, *key*, which specifies a message key to look up in the Action's ResourceBundles 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). When a *key* is specified, any text contained in the body of the message tag becomes the default message if a message cannot be found for the given key.

If the validator fails, the validator is pushed onto the ValueStack and the message – either the default or the locale-specific one if the *key* attribute is defined (and such a message exists) – 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.

A more complete example of the validation rules can be found [here](#).

 Since validation rules are in an XML file, you must make sure you escape special characters. For example, notice that in the expression validator rule above we use ">" instead of ">". Consult a resource on XML for the full list of characters that must be escaped. The most commonly used characters that must be escaped are: & (use &), > (use >), and < (use <).

`<validator>` vs. `<field-Validator>` syntax

There are two ways you can define validators in your `-validation.xml` file:

1. `<validator>`
2. `<field-validator>`

Keep the following in mind when using either syntax:

`<validator>`

The `<validator>` element allows you to declare both types of validators (either a plain Validator or a field-specific FieldValidator).

Declaring a plain Validator using the `<validator>` syntax:

```
<validator type="expression">
  <param name="expression">foo gt bar</param>
  <message>foo must be great than bar.</message>
</validator>
```

Declaring a FieldValidator using the `<validator>` syntax:

```
<validator type="required">
  <param name="fieldName">bar</param>
  <message>You must enter a value for bar.</message>
</validator>
```

<field-validator>

The `<field-validator>` elements are basically the same as the `<validator>` elements except that they inherit the *fieldName* attribute from the enclosing `<field>` element. FieldValidators defined within a `<field-validator>` element will have their *fieldName* automatically filled with the value of the parent `<field>` element's *fieldName* attribute. The reason for this structure is to conveniently group the validators for a particular field under one element, otherwise the *fieldName* attribute would have to be repeated, over and over, for each individual `<validator>`.

Declaring a FieldValidator using the `<field-validator>` syntax:

```
<field name="email_address">
  <field-validator type="required">
    <message>You cannot leave the email address field empty.</message>
  </field-validator>
  <field-validator type="email">
    <message>The email address you entered is not valid.</message>
  </field-validator>
</field>
```

The choice is yours. It's perfectly legal to only use `<validator>` elements without the `<field>` elements and set the *fieldName* attribute for each of them. The following are effectively equal:

```
<field name="email_address">
  <field-validator type="required">
    <message>You cannot leave the email address field empty.</message>
  </field-validator>
  <field-validator type="email">
    <message>The email address you entered is not valid.</message>
  </field-validator>
</field>

<validator type="required">
  <param name="fieldName">email_address</param>
  <message>You cannot leave the email address field empty.</message>
</validator>
<validator type="email">
  <param name="fieldName">email_address</param>
  <message>The email address you entered is not valid.</message>
</validator>
```



Hint

It is always better to define field-validator inside a `<field>` tag instead of using a `<validator>` tag and supplying `fieldName` as its param as the xml code itself is clearer (grouping of field is clearer)

Note that you should only use FieldValidators (**not** plain Validators) within a `<field-validator>` block. A plain Validator inside a `<field>` will not be allowed and would generate error when parsing the xml, as it is not allowed in the defined dtd (`xwork-validator-1.0.2.dtd`)

Short-circuiting validation checks

Beginning with XWork 1.0.1 (bundled with WebWork 2.1), it is possible to short-circuit a stack of validators. Here is another sample config file containing validation rules from the Xwork test cases:

```
<!DOCTYPE validators PUBLIC "-//OpenSymphony Group//XWork Validator 1.0.2//EN"
"http://www.opensymphony.com/xwork/xwork-validator-1.0.2.dtd">
<validators>
  <!-- Field Validators for email field -->
  <field name="email">
    <field-validator type="required" short-circuit="true">
      <message>You must enter a value for email.</message>
    </field-validator>
    <field-validator type="email" short-circuit="true">
      <message>Not a valid e-mail.</message>
    </field-validator>
  </field>
  <!-- Field Validators for email2 field -->
  <field name="email2">
    <field-validator type="required">
      <message>You must enter a value for email2.</message>
    </field-validator>
    <field-validator type="email">
      <message>Not a valid e-mail2.</message>
    </field-validator>
  </field>
  <!-- Plain Validator 1 -->
  <validator type="expression">
    <param name="expression">email.equals(email2)</param>
    <message>Email not the same as email2</message>
  </validator>
  <!-- Plain Validator 2 -->
  <validator type="expression" short-circuit="true">
    <param name="expression">email.startsWith('mark')</param>
    <message>Email does not start with mark</message>
  </validator>
</validators>
```

Notice that some of the `<field-validator>` and `<validator>` elements have the *short-circuit* attribute set to true.

short-circuiting and Validator flavors

Plain validator takes precedence over field-validator. They get validated first in the order they are defined and then the field-validator in the order they are defined. Failure of a particular validator marked as short-circuit will prevent the evaluation of subsequent validators and an error (action error or field error depending on the type of validator) will be added to the ValidationContext of the object being validated.

In the example above, the actual execution of validator would be as follows:

1. Plain Validator 1
2. Plain Validator 2
3. Field Validators for email field
4. Field Validators for email2 field

Since Field Validator 2 is short-circuited, if its validation failed, it will causes Field validators for email field and Field validators for email2 field to not be validated as well.



Useful Information

More complecated validation should probably be done in the validate() method on the action itself (assuming the action implements Validatable interface which ActionSupport already does).

A plain Validator (non FieldValidator) that gets short-circuited will completely break out of the validation stack – no other validators will be evaluated and plain validator takes precedence over field validator meaning that they get evaluated in the order they are defined before field validator gets a chance to be evaludated again according to their order defined.

short-circuiting and Field Validator flavors

A FieldValidator that gets short-circuited will only prevent other FieldValidators for the **same field** from being evaluated. Note that this "same field" behavior applies regardless of whether the <validator> or <field-validator> syntax was used to declare the validation rule. By way of example, given this -validation.xml file:

```
<validator type="required" short-circuit="true">
  <param name="fieldName">bar</param>
  <message>You must enter a value for bar.</message>
</validator>

<validator type="expression">
  <param name="expression">foo gt bar</param>
  <message>foo must be great than bar.</message>
</validator>
```

both validators will be run, even if the "required" validator short-circuits. "required" validators are FieldValidator's and will not short-circuit the plain ExpressionValidator because FieldValidators only short-circuit other checks on that same field. Since the plain Validator is not field specific, it is not short-circuited.

How Xwork finds a validator for an action

As mentioned above, the framework will also search up the inheritance tree of the action to find default validations for interfaces and parent classes of the Action. If you are using the *short-circuit* attribute and relying on default validators higher up in the inheritance tree, make sure you don't accidentally short-circuit things higher in the tree that you really want!



Upgrade Alert: The *short-circuit* attribute was added to the DTD in version 1.0.2. Your validation files must use version 1.0.2 of the xwork-validation.dtd to validate properly.

Building a Validator

This page last changed on Dec 15, 2004 by [jhouse](#).

Validators implement the **com.opensymphony.xwork.validator.Validator** interface

```
public interface Validator {
    void setDefaultMessage(String message);

    String getDefaultMessage();

    String getMessage(Object object);

    void setMessageKey(String key);

    String getMessageKey();

    /**
     * This method will be called before validate with a non-null ValidatorContext.
     * @param validatorContext
     */
    void setValidatorContext(ValidatorContext validatorContext);

    ValidatorContext getValidatorContext();

    /**
     * The validation implementation must guarantee that setValidatorContext
     * will be called with a non-null ValidatorContext before validate is called.
     * @param object
     * @throws ValidationException
     */
    void validate(Object object) throws ValidationException;
}
```

FieldValidators implement **com.opensymphony.xwork.validator.FieldValidator**, which extends Validator:

```
public interface FieldValidator extends Validator {

    /**
     * Sets the field name to validate with this FieldValidator
     * @param fieldName
     */
    void setFieldName(String fieldName);

    /**
     * @return the field name to be validated
     */
    String getFieldName();
}
```

If you want to be able to use the "short-circuit" attribute, you should also implement **com.opensymphony.xwork.validator.ShortCircuitableValidator**.

Validators and FieldValidators can extend base classes

com.opensymphony.xwork.validator.validators.ValidatorSupport and **com.opensymphony.xwork.validator.validators.FieldValidatorSupport** to get the base message and short-circuiting behavior, and will only need to implement `validate(Action action)`.

The Support classes provide the functionality to use the message key and default message to get the localized message body and the parsing of the message body to provide for parameterized messages. Implementations of the Validator Interface which do not extend the Support base classes should provide

this functionality as well for consistency.

The **ValidatorContext** set into the Validator is an interface which extends both **ValidationAware** and **LocaleAware** and is used for looking up message texts and setting errors. When validators are called from the ValidationInterceptor, a DelegatingValidatorContext is created which delegates these calls to the Action if it implements these interfaces. If the Action does not implement LocaleAware, a LocaleAwareSupport instance is created which uses the Action's class to look up resource bundle texts, if available. If the action does not implement ValidationAware, an implementation which simply logs the validation errors is created and delegated to. When calling the validation framework from outside the ValidationInterceptor, any ValidatorContext implementation can be passed in.

A new instance of the Validator is created by the framework each time it is asked to do work. This provides for thread safety from within the framework.

Validator classes may define any number of properties using the usual getX() setX() naming convention and have those properties set using <param name="x">foo</param> elements below the <validator> element. The values of these properties may then be used in the validate() method to parameterize the validation. Validators which extend the Support classes may also use the

```
Object getFieldValue(String name, Action action)
```

method to get the field value of a named property from an Action.

Custom Action Validation

This page last changed on Dec 19, 2005 by [mrdon](#).

XWork's validators work great when you need to apply a common validation to your Action, particularly syntactic validations that simply check values for the correct format. In a real world application, you might need to execute custom validation code that runs checks that are specific to an Action. XWork allows you to, by implementing the *Validatable* interface, define a custom *validate()* method on your action, which will be called before the Action will be executed.

For example, say you were developing a To Do list application, which contains a form that allows the user to modify an existing todo entry. You need to create an Action which processes the submitted information and updates the todo information in database. To check that the fields' values were in the correct formation, you might use the following XWork validators on the following fields:

Validator	Field(s)
RequiredFieldValidator	<i>id,name,description,completed</i>
StringLengthFieldValidator	<i>name</i>
IntRangeFieldValidator	<i>id</i>

However, you want to make sure the *id* of the todo form actually exists before you have the business layer make update the database. While the XWork validators checked that the *id* field existed and contained a numeric value, you need to ensure custom validation code will be called during the validation process.

The solution is to have your Action implement the *Validatable* interface, which defines a *validate()* method that you must implement. If your action has the *DefaultWorkflowInterceptor* in its interceptor stack, your *validate()* method will be called. If the *validate()* method, or any validators previously have generated any error messages, the chain will abort processing and return the *Action.INPUT* result (see [DefaultWorkflowInterceptor](#) for more information).

This is what the *validate()* method of our todo Action, which extends *ActionSupport*, might look like:

```
public void validate() {
    if (todoManager.getTodo(id) == null) {
        String error = getText("todo.err.notFound");
        addActionError(error);
    }
}
```

The *todoManager* variable is a instance of the business management object that handles all todo database operations. If the *id* field as submitted by the form doesn't exist, we lookup the text for the error and add it as an action error.

If, however, you allowed todo entries to be modified in an other area of the application, you might consider converting this code into a custom Validator to avoid code duplication. See [Building a Validator](#) for details.

ExpressionValidator Tips

This page last changed on Mar 25, 2005 by [gaffonso](#).

ExpressionValidator Tips

Fieldname references

As long as your Action provides a "setter" and a "getter" for your form fields, you can simply refer to them in your expression by name.

Given this form field:

```
<input type=text name="foo">
```

And this Action fragment

```
private String foo;  
public void setFoo(String Foo) { this.foo = foo; }  
public String getFoo() {return foo; }
```

Your form fields are available to your expression like this:

```
foo == "sometext"
```

Advanced field references

If you expose an object in your Action via a getter, you can directly set (from the form) and access (from the validation expression) properties **within** that object.

Given this form fragment:

```
<input type=text name="address.firstName">  
<input type=text name="address.lastName">
```

And this Action fragment:

```
private Address address;  
public void setAddress(Address address) { this.address = address; }  
public Address getAddress() {return address; }
```

Your validation expression can do this:

```
address.firstName == "Joe" and address.lastName == "Bloe"
```



Some use this feature to expose their "model" or "domain" objects directly to their forms and to their validation code.

This can be incredibly powerful but should be used with caution as it allows anybody monkey-ing with URL's (or custom forms) to inject data directly into your domain objects that you might not want to have there.

Jason said it well: "It's a great tool, but it's a sharp tool, and you can cut yourself with it."

ExpressionValidators use OGNL

OGNL: Pronounced like the "ogonal" part of orthogonal.

The ExpressionValidator expressions are simply OGNL expressions that evaluate to true or false. You have the complete OGNL language and stack available to you when writing ExpressionValidator expressions.

For most of what you probably want to do, you don't need to worry about all that. Just reference your fields by name, use standard comparison operators, etc.

But if you want the full meal deal there is a huge amount of power (and complexity) available if you want to take the time to plumb the depths of OGNL. The OGNL docs are here:

<http://www ognl.org/>

False expressions trigger the validation error

I often find myself thinking of my validation tests exactly opposite of how WebWork thinks of them.

If you want to trigger a validation error if two fields are inequal, you might be inclined to do this:

```
email != emailVerified
```

That will do exactly the opposite of what you expect. It will return "false" when the fields are equal (and trigger a validation error) and "true" when the fields are not equal (not triggering a validation error).

You want to do this:

```
email == emailVerified
```

If your brain works like mine, you'll probably end up negating a lot of your expressions. If you have long

or compound expressions that evaluate to true, just wrap the whole thing in parenthesis to negate it:

```
!  
(  
  ( chosenAddressId.intValue() eq -1 )  
  and  
  ( regionId.intValue() eq -1 )  
  and  
  ( ( countryId.intValue() eq 1 ) or ( countryId.intValue() eq 2 ) )  
)
```

Expression can be multi-line

As you can see above, your expression can span as many lines as you like. This really helps keep complex expressions readable.

Use the validation message for debugging

A common mistake is to incorrectly refer to the field you want to compare. You can verify that the field you **think** you're referring to is the field you're **actually** referring to by creating an ExpressionValidator that always returns false (meaning that it always triggers a validation error), and emitting the field values in its message like so:

```
<field name="user.password_confirm">  
  <field-validator type="fieldexpression">  
    <param name="expression">false</param>  
    <message>${password}    ${password_verified}</message>  
  </field-validator>  
</field>
```

Special handling of null and String equality

nulls

The comparison operators used by the ExpressionValidator will internally handle nulls and, generally, "do the right thing" with regard to comparisons containing fields with null values. Recall that in java you might be inclined to write a test like so...

```
if( field != null && field.intValue() == 2 )
```

...to ensure your field wasn't null so you avoid a potential null-pointer exception in the rest of the test. This is unnecessary in OGNL, simply do this:

```
field.intValue() == 2
```

Strings

There is also no need to do .equals() comparisons on strings. Recall that in Java you usually do not compare two strings with the == operator, you use the .equals() method of your String object instead. But the OGNL expression language will automatically do a .equals() comparison for you when you use the == operator on two Strings.

These two are equivalent:

```
password.equals(passwordVerified)
```

... is equivalent to ...

```
password == passwordVerified
```

From the OGNL docs:

Equality is tested for as follows. If either value is null, they are equal if and only if both are null. If they are the same object or the equals() method says they are equal, they are equal. If they are both Numbers, they are equal if their values as double-precision floating point numbers are equal. Otherwise, they are not equal. These rules make numbers compare equal more readily than they would normally, if just using the equals method.

FieldValidators do not short-circuit ExpressionValidators.

Ever.

See the docs (currently only available on the Wiki docs) for information about short-circuiting:

<http://wiki.opensymphony.com/display/XW/Validation+Framework>

The upshot is that a FieldValidator (regardless of whether you use the <validator> or <field-validator> syntax) will **never** short-circuit an ExpressionValidator. And since most of the built-in validators are FieldValidators, you probably will not get the short-circuit behavior you want by mixing FieldValidators and ExpressionValidators.

Given this example...

```
<validator type="required" short-circuit="true">
  <param name="fieldName">bar</param>
  <message>You must enter a value for bar.</message>
</validator>

<validator type="expression">
  <param name="expression">foo gt bar</param>
  <message>foo must be great than bar.</message>
</validator>
```

...the ExpressionValidator will **always** run.

If you find yourself needing to short-circuit a FieldValidator you can always create an ExpressionValidator that does the equivalent work of the FieldValidator. Once your validation check is implemented as an ExpressionValidator it will be short-circuited by other validators.

ExpressionValidators do short-circuit FieldValidators.

An ExpressionValidator **will** short-circuit FieldValidators.

If we rewrite the above example to place the short-circuit on the ExpressionValidator...

```
<validator type="required">
  <param name="fieldName">bar</param>
  <message>You must enter a value for bar.</message>
</validator>

<validator type="expression" short-circuit="true">
  <param name="expression">foo gt bar</param>
  <message>foo must be great than bar.</message>
</validator>
```

... we get behavior wherein a validation error by the ExpressionValidator will cause the "required" FieldValidator to be skipped.

Turn on XML reloading

During development/debugging of expressions, it can be immensely helpful to turn on XML reloading. This will allow your changes to the -validation.xml files be noticed by WebWork without a restart.

In webwork.properties set...

```
webwork.configuration.xml.reload = true
```

See the docs about the webwork.properties file and where it should be placed.



Note that if your development environment has you editing your "main" code files in one place and then copying them into your app-server for deployment *xml.reload* only reloads the **deployed** copies.

That's a no-duh, but it's easy to forget in IDE's like NetBeans. NetBeans maintains the "main" code in a separate directory from the code actually used by the deployed app (which it locates in the build directory).

If this is the case with your dev environment, no worries, just edit the **deployed** -validation.xml file while you're developing and debugging and then copy the final changes back to your "main" file when you're done.

Be sure to copy those changes back to your main file before you rebuild!!

Letter-based comparison operators are your friends

the problem

Many of the normal comparison operators cannot be used un-escaped in your XML validation file. For example, you can't use an expression like this...

```
field > 2
```

...because the greater-than sign has special meaning in XML.

the solution

Some would suggest you escape the greater-than sign like this:

```
field &gt; 2
```

Others have suggested that you use a CDATA block like this (note, I have not personally tried this):

```
<![CDATA[ field > 2 ]]>
```

But I find both of those rather ugly. The easier thing to do is to just remember that all of the OGNL comparison operators have letter-based notations. So the above (within an ExpressionValidator) works perfectly well:

```
field gt 2
```

The letter-based notation for each of the OGNL comparison operators can be found in this section of the OGNL docs:

<http://www ognl.org/2.6.7/Documentation/html/LanguageGuide/apa.html#operators>

I've gotten into the habit of using letter-based operator syntax for all my OGNL comparison operators whether they need to be escaped or not. Keeps my brain in the "OGNL" groove (and helps remind this is **not** java and that there are some subtle differences such as the null and String handling noted above).



OGNL's letter-based comparison operators feel comfortably perl-ish. But remember that OGNL's letter-based comparison operators work on **all** datatypes, unlike perl where the letter-based comparison operators work only on strings.

Generic Object Validation

This page last changed on Dec 03, 2005 by [rainerh](#).

XWork's validation framework is not just limited to Actions. It can be used to validate virtually any object. Once you've set up your validator config file (**validators.xml**) and your validation rules (**ClassName-validation.xml**), all it takes are a couple lines of code:

```
String context = null;
ValidatorContext context = new DelegatingValidatorContext(objectToValidate);
ActionValidatorManagerFactory.getInstance().validate(objectToValidate, null, context);
```

This will cause any errors to be logged (where it gets logged to depends on how commons-logging is configured).

Ideally, you would either implement your own ValidatorContext to handle how error messages are logged and evaluated, or have your objects that are to be evaluated implement ValidationAware, TextProvider and/or LocaleProvider.

Sample Validation Rules

This page last changed on Jul 16, 2004 by [unkyaku](#).

In this example, we have an Action that attempts to fill the values of a User object. The first validation file would be for a simple Action that had a "user" property. The second validation file is for the exact same Action if it had implemented the ModelDriven interface and exposed the User object via **getModel()** instead of **getUser()**. Both of them uses the VisitorFieldValidator to use the User object's own validation rules to validate the User object, which can be found in the third validation file.

Notice that the *short-circuit* attribute of the **field-validator** element is used in several places to prevent subsequent validators from running. This is useful in preventing multiple error messages when one will do. For example, if the value provided for the User's startDate cannot be parsed as a date, display the appropriate message and don't even bother checking that it is in the correct range.

SampleAction-validation.xml:

```
<validators>
  <field name="user">
    <field-validator type="required" short-circuit="true">
      <message>You must provide a user.</message>
    </field-validator>
    <field-validator type="visitor">
      <param name="context">anotherContext</param>
      <message>user: </message>
    </field-validator>
  </field>
</validators>
```

SampleModelDrivenAction-validation.xml:

```
<validators>
  <field name="model">
    <field-validator type="visitor">
      <param name="appendPrefix">false</param>
      <message />
    </field-validator>
  </field>
</validators>
```

User-validation.xml:

```
<validators>
  <field name="userName">
    <field-validator type="requiredstring" short-circuit="true">
      <message>You must enter an username.</message>
    </field-validator>
    <field-validator type="stringlength">
      <param name="minLength">3</param>
      <param name="maxLength">10</param>
      <message>Username must be between ${minLength} and ${maxLength} characters
long.</message>
    </field-validator>
  </field>
  <field name="email">
    <field-validator type="email">
      <message>You must enter a valid email address.</message>
    </field-validator>
  </field>
  <field name="homepage">
```

```
<field-validator type="url" short-circuit="true">
  <message>You must enter a valid URL.</message>
</field-validator>
<field-validator type="fieldexpression">
  <param name="expression">homepage.indexOf('opensymphony.com') == -1</param>
  <message>Please provide an OpenSymphony website</message>
</field-validator>
</field>
<field name="age">
  <field-validator type="int">
    <param name="min">0</param>
    <param name="max">100</param>
    <message>Not a valid age!</message>
  </field-validator>
</field>
<field name="startDate">
  <field-validator type="conversion" short-circuit="true">
    <validator type="startDate">
      <message>Could not convert input to a valid date.</message>
    </validator>
  </field-validator>
  <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>
```

Standard Validators

This page last changed on Sep 21, 2006 by [perfnorm](#).

These are the standard validators that come with XWork:

- RequiredFieldValidator
- RequiredStringValidator
- StringLengthFieldValidator
- StringRegexValidator
- EmailValidator
- URLValidator
- IntRangeFieldValidator
- DateRangeFieldValidator
- ConversionErrorFieldValidator
- ExpressionValidator
- FieldExpressionValidator
- VisitorFieldValidator

All the example XML validation rule snippets below have been put together into one big example [here](#).

RequiredFieldValidator

This validator checks that a field is non-null.

Example:

```
<validators>
  <field name="user">
    <field-validator type="required">
      <message>You must enter a value for user.</message>
    </field-validator>
  </field>
</validators>
```

RequiredStringValidator

This validator checks that a String field is not empty (i.e. non-null with a length > 0).

Parameter	Required	Default	Notes
trim	no	true	Boolean property. Determines whether the String is trimmed before performing the length

			check.
--	--	--	--------

Example:

```
<validators>
  <field name="userName">
    <field-validator type="requiredstring">
      <message>You must enter an username.</message>
    </field-validator>
  </field>
</validators>
```

StringLengthFieldValidator

This validator checks that a String field is of the right length. It assumes that the field is a String.

Parameter	Required	Default	Notes
trim	no	true	Boolean property. Determines whether the String is trimmed before performing the length check.
minLength	no		Integer property. The minimum length the String must be.
maxLength	no		Integer property. The maximum length the String can be.

If neither *minLength* nor *maxLength* is set, nothing will be done.

Example:

```
<validators>
  <field name="userName">
    <field-validator type="stringlength">
      <param name="minLength">3</param>
      <param name="maxLength">10</param>
      <message>Username must be between ${minLength} and ${maxLength} characters
    long.</message>
    </field-validator>
  </field>
</validators>
```

StringRegexValidator

This validator checks that a String field matches a configured Regular Expression, if it is not an empty String.

Parameter	Required	Default	Notes
expression	yes		String property. The Regular Expression for which to check a match.
caseSensitive	no	true	Whether the matching of alpha characters in the expression should be done case-sensitively.

Example:

```
<validators>
  <field name="coords">
    <field-validator type="regex">
      <param name="expression">[0-9],[0-9]</param>
      <message>The value of coords must be in the format "x, y", where x and y are
between 0 and 9</message>
    </field-validator>
  </field>
</validators>
```

EmailValidator

This validator checks that a field is a valid e-mail address if it contains a non-empty String.

Example:

```
<validators>
  <field name="email">
    <field-validator type="email">
      <message>You must enter a valid email address.</message>
    </field-validator>
  </field>
</validators>
```

URLValidator

This validator checks that a field is a valid URL.

Example:

```
<validators>
  <field name="homepage">
    <field-validator type="url">
      <message>You must enter a valid URL.</message>
    </field-validator>
  </field>
</validators>
```

IntRangeFieldValidator

This validator checks that a numeric field has a value within a specified range.

Parameter	Required	Default	Notes
min	no		Integer property. The minimum the number must be.
max	no		Integer property. The maximum number can be.

If neither *min* nor *max* is set, nothing will be done.

Example:

```
<validators>
  <field name="age">
    <field-validator type="int">
      <param name="min">0</param>
      <param name="max">100</param>
      <message>Not a valid age!</message>
    </field-validator>
  </field>
</validators>
```

DateRangeFieldValidator

This validator checks that a date field has a value within a specified range.

Parameter	Required	Default	Notes
min	no		Date property. The minimum the date must be.
max	no		Date property. The maximum date can be.

If neither *min* nor *max* is set, nothing will be done.

Example:

```
<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>
```

ConversionErrorFieldValidator

This validator checks if there are any conversion errors for a field and applies them if they exist. See [Type Conversion Error Handling](#) for details.

Example:

```
<validators>
  <field name="startDate">
    <field-validator type="conversion">
      <message>Could not convert input to a valid date.</message>
    </field-validator>
  </field>
</validators>
```

ExpressionValidator

This validator uses an [OGNL](#) expression to perform its validation. The error message will be added to the **action** if the expression returns false when it is evaluated against the value stack.

Parameter	Required	Default	Notes
expression	yes		An OGNL expression that returns a boolean value.

Example:

```
<validator type="expression">
  <param name="expression">foo > bar</param>
  <message default="Foo must be greater than Bar. Foo = ${foo}, Bar = ${bar}."/>
</validator>
```

FieldExpressionValidator

This validator uses an [OGNL](#) expression to perform its validation. The error message will be added to the **field** if the expression returns false when it is evaluated against the value stack.

Parameter	Required	Default	Notes
expression	yes		An OGNL expression that returns a boolean value.

Example:

```
<validators>
  <field name="homepage">
    <field-validator type="fieldexpression">
      <param name="expression">homepage.indexOf('opensymphony.com') == -1</param>
      <message>Please provide an OpenSymphony website</message>
    </field-validator>
  </field>
</validators>
```

VisitorFieldValidator

The validator 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.

The error message for the VisitorFieldValidator will be appended in front of validation messages added by the validations for the Object message.

Parameter	Required	Default	Notes
context	no	action alias	Determines the context to use for validating the Object property. If not defined, the context of the Action validation is propagated to the Object property validation. In the case of Action validation, this context is the Action alias.
appendPrefix	no	true	Determines whether the field name of this field validator should be prepended to the field

			<p>name of the visited field to determine the full field name when an error occurs. For example, suppose that the bean being validated has a "name" property. If <i>appendPrefix</i> is true, then the field error will be stored under the field "bean.name". If <i>appendPrefix</i> is false, then the field error will be stored under the field "name".</p> <p>⚠️ If you are using the <code>VisitorFieldValidator</code> to validate the model from a <code>ModelDriven</code> Action, you should set <i>appendPrefix</i> to false unless you are using "model.name" to reference the properties on your model.</p>
--	--	--	--

Example:

```
<validators>
  <field name="user">
    <field-validator type="visitor">
      <param name="context">anotherContext</param>
      <message>user: </message>
    </field-validator>
  </field>
</validators>
```

Here we see the *context* being overridden in the validator mapping, so the action alias context will not be propagated.

ModelDriven example:

```
<validators>
  <field name="model">
    <field-validator type="visitor">
      <param name="appendPrefix">false</param>
      <message />
    </field-validator>
  </field>
</validators>
```

This will use the model's validation rules and any errors messages will be applied directly (nothing is prefixed because of the empty message).

Overview

Xwork is a command pattern framework centralized around an Action interface. You define action classes by implementing an Action interface, then XWork will setup and execute your actions. XWork is most widely known from the web MVC framework called Webwork. However, XWork can be used by itself, so its important to understand the XWork layers and how actions are set up and executed. This section describes the core layers within Xwork and provides a simple example of how to execute actions.

- Action Interface
- ActionProxy interface
- ActionInvocation interface
- ActionContext
- A simple example

Actions

Actions are the basic unit of execution...

The Action Interface

The basic interface which all XWork actions must implement. It provides several standard return values like SUCCESS and INPUT, and only contains one method:

```
public interface Action {
    public static final String SUCCESS = "success";
    public static final String NONE = "none";
    public static final String ERROR = "error";
    public static final String INPUT = "input";
    public static final String LOGIN = "login";

    public String execute() throws Exception;
```

In general, Actions can simply extend the `com.opensymphony.xwork.ActionSupport` class, which implements the Action interface and provides default behavior for the most common actions.

ActionProxy

Action lifecycles are maintained thru the ActionProxy interface. ActionProxy is the top layer in the Xwork API and should be the starting point to setup and execute actions. XWork provides a factory as an entry point to instantiate action proxies. Most of the implementations of each xwork layer are hidden behind interfaces making it very easy to override the default implementations for complete customization.

Example how to obtain the default impl of ActionProxy (DefaultActionProxy.java)

```
ActionProxyFactory.getFactory().createActionProxy("", "viewBook", objectMap);
```

If I need to register my own implementation of ActionProxy, then I may do so within the factory

```
class CustomizedActionProxyFactory extends DefaultActionProxyFactory{
    createActionProxy(...){ return new CustomizedActionProxy(...); }
}
```

```
ActionProxyFactory.setFactory(new CustomizedActionProxyFactory());
ActionProxy proxy = ActionProxyFactory.getFactory().createActionProxy(...);
```

ActionInvocation

Underneath the ActionProxy layer, exists the ActionInvocation interface. ActionInvocation represents the execution state of an action holding the action instance along with any interceptors that wrap before/after processing of the action.

ActionContext

ActionContext provides access to the execution environment in the form of named objects during an Action invocation. A new ActionContext is created for each invocation allowing developers to access/modify these properties in a thread safe manner. The ActionContext makes a number of properties available that are typically set to appropriate values by the framework. In WebWork 2 for example, the ActionContext session map wraps an underlying HttpSession object. This allows access to environment specific properties without tying the core framework to a specific execution environment.

The ActionContext is acquired through the static ActionContext.getContext() method. The ActionContext is a thread local variable and thus the properties of the ActionContext will be relative to the current request thread. The ActionContext has several methods for commonly used properties as well as get() and set() methods which can be used for application specific properties.

A simple example

Lets look at a simple example starting with a simple javabean.

```
public class Book {
    String id;
    String title;
    Set authors;
    public void setId(id){ this.id = id; }
    public void setTitle(String title){ this.title = title; }
    public void setAuthors(Set authors){ this.authors = authors; }
    public String getId(){ }
    public String getTitle(){ }
    public Set getAuthors(){ }
```

```
}
```

Lets say that we need to retrieve a book object from a data source and pass it back to the caller. We can write an action to handle this. An action in xwork is typically a very simple class. The only requirement is that it implements the Action interface. These days you'll see actions as simple as javabeans with an execute method (Validation, Type conversion, and so forth can all be separated out to provide a good separation of concerns). The purpose of action execution is typically to provide access and manipulation to your data. The result of the action execution is a simple string representation that should define delegation of the action after invocation. So a result could be a success string, a failure string, a forward string, or what ever. In our current example, a book object can be populated in the action if found with a result of "success" or if the book is not found then a "notFound" can be returned. From this, you can easily have a controlling object setup to return the book or possible forward the request off to a different inventory source if the book isn't found.

```
public class ViewBookAction implements Action{
    Book book;
    String id;

    public String execute() throws Exception{

        // lets pretend we have a data access object that will return a book from
storage        book = bookDAO.findById(id, Book.class);
                if(book != null) return "success";
                return "notFound";
    }
    public Book getBook(){ return this.book; }
    public setId(String id){this.id = id; }
}
```

Now that we have an action defined with a simple model, lets setup an action proxy and execute the action.

Setting up XWork to execute the action:

```
// obtain inputs from the caller. For this example, we can just define some dummy params.
Map paramMap = new HashMap();
paramMap.put("id", "0123456789");

// set the ActionContext parameters
Map context = new HashMap();
context.put(ActionContext.PARAMETERS, paramMap);

// create an action proxy with no namespace, action alias (defined in xwork.xml), and a map of
the context info
ActionProxy proxy = ActionProxyFactory.getFactory().createActionProxy("", "viewBook", context);

// we have the action proxy instance, lets execute it and retrieve the action
String result = proxy.execute();
if ("success".equals(result)){
    ViewBookAction action = (ViewBookAction) proxy.getAction();

    // return info back to caller or just print to screen for this example
    System.out.println(action.getBook().getTitle());
} else if ("notFound".equals(result)){
    // forward to another inventory source
} else {
    throw new RuntimeException("Im lazy");
}
```

Not quite done yet, we need to define some configuration in xwork.xml so XWork can find the appropriate class to execute based on the action alias we provided within the createActionProxy(...) method.

```
<xwork>
  <include file="xwork-default.xml"/>
  <package name="default" extends="xwork-default">
    <action name="viewBook" class="com.opensymphony.xwork.example.ViewBookAction"/>
  </package>
</xwork>
```


Press Releases

This page last changed on Aug 08, 2006 by [rainerh](#).

Here you will find all the press releases for XWork. Each press release should include the [About](#) page at the top of it, which you don't need to worry about as long as you use the standard Press Release template.

- [1.2.0 Press Release](#)
- [1.1.2 Press Release](#)
- [1.0.5 Press Release](#)
- [1.0.4 Press Release](#)
- [1.0.3 Press Release](#)
- [1.0.2 Press Release](#)
- [1.0.1 Press Release](#)

1.0.1 Press Release

This page last changed on Jun 23, 2004 by [plightbo](#).

XWork 1.0.1 Released

The [OpenSymphony](#) group is proud to announce the release of XWork 1.0.1. This new release paves the way for future versions of WebWork as well as new products such as "SwingWork" and other concepts that are thought up every day by OpenSymphony's active community.

- XWork: <http://www.opensymphony.com/xwork>
- Release notes and changes:
<http://www.opensymphony.com/xwork/wikidocs/Release%20Notes.html>
- Documentation: <http://www.opensymphony.com/xwork/wikidocs/Documentation.html>

New Features

The biggest addition to XWork is the new ObjectFactory which allows for tight integration to other projects such as Spring and Pico. Not only that, the ObjectFactory is very useful to integrate XWork in to your existing project infrastructure. Documentation (both reference manuals and JavaDocs) have been greatly improved as well.

Bug Fixes

There have been several important bug fixes, the most important is a performance-related bug fix that makes the expression language magnitudes faster. This also solved an extremely rare problem where some applications would lock up.

Special Thanks

Special thanks go out to Bill Lynch of [Jive Software](#) for helping with all the added JavaDocs. Mark Woon has also done a superb job assisting with bug fixes, new features, and documentation.

About XWork

XWork is a generic command pattern implementation and inversion of control (IoC) container. It is designed to be small, fast, and fit specifically for workflow-related usages, such as web application frameworks and workflow engines. WebWork provides the foundation for [WebWork](#), a Java-based web framework.

Besides WebWork, you can find integration support in [OSWorkflow](#) and [JPublish](#).

Core features include:

- Command pattern implementation

- Interceptor support allowing you to provide tight application integration
- Inversion of Control framework, helping you manage complex component dependencies
- Decoupled validation framework, letting you focus on disconnecting input validation from core business logic
- Extremely fast and flexible expression language based off of [OGNL](#)
- Advanced type conversion support, allowing you to never have to write data conversion code inside of your core business objects

1.0.2 Press Release

This page last changed on Aug 26, 2004 by [plightbo](#).

XWork 1.0.2 Released

The [OpenSymphony](#) group is proud to announce the release of XWork 1.0.2. This release adds several new features and bug fixes in the areas of validation and type conversion.

- XWork: <http://www.opensymphony.com/xwork>
- Release notes and changes:
<http://www.opensymphony.com/xwork/wikidocs/Release%20Notes%20-%201.0.2.html>
- Documentation: <http://www.opensymphony.com/xwork/wikidocs/Documentation.html>

Special Thanks

Special thanks to Mark Woon for providing much of the code and documentation in this release. Mark has become one of the core developers for XWork and WebWork in the past couple of months and we expect to see much more come from him in the future.

About XWork

XWork is a generic command pattern implementation and inversion of control (IoC) container. It is designed to be small, fast, and fit specifically for workflow-related usages, such as web application frameworks and workflow engines. WebWork provides the foundation for [WebWork](#), a Java-based web framework.

Besides WebWork, you can find integration support in [OSWorkflow](#) and [JPublish](#).

Core features include:

- Command pattern implementation
- Interceptor support allowing you to provide tight application integration
- Inversion of Control framework, helping you manage complex component dependencies
- Decoupled validation framework, letting you focus on disconnecting input validation from core business logic
- Extremely fast and flexible expression language based off of [OGNL](#)
- Advanced type conversion support, allowing you to never have to write data conversion code inside of your core business objects

1.0.3 Press Release

This page last changed on Sep 07, 2004 by [plightbo](#).

XWork 1.0.3 Released

The [OpenSymphony](#) group is proud to announce the release of XWork 1.0.3. This release is a minor repackaging of 1.0.2, which was missing xwork-validator-1.0.dtd. No code changes are in this release, so upgrading is 100% safe.

- XWork: <http://www.opensymphony.com/xwork>
- Release notes and changes: <http://www.opensymphony.com/xwork/wikidocs/Release%20Notes%20-%201.0.3.html>
- Documentation: <http://www.opensymphony.com/xwork/wikidocs/Documentation.html>

About XWork

XWork is a generic command pattern implementation and inversion of control (IoC) container. It is designed to be small, fast, and fit specifically for workflow-related usages, such as web application frameworks and workflow engines. WebWork provides the foundation for [WebWork](#), a Java-based web framework.

Besides WebWork, you can find integration support in [OSWorkflow](#) and [JPublish](#).

Core features include:

- Command pattern implementation
- Interceptor support allowing you to provide tight application integration
- Inversion of Control framework, helping you manage complex component dependencies
- Decoupled validation framework, letting you focus on disconnecting input validation from core business logic
- Extremely fast and flexible expression language based off of [OGNL](#)
- Advanced type conversion support, allowing you to never have to write data conversion code inside of your core business objects

1.0.4 Press Release

This page last changed on Nov 14, 2004 by [plightbo](#).

XWork 1.0.4 Released

The [OpenSymphony](#) group is proud to announce the release of XWork 1.0.4. This release contains minor internal bug fixes and tweaks to the API. It is completely backwards compatible with version 1.0.3.

- XWork: <http://www.opensymphony.com/xwork/>
- Release notes and changes:
<http://www.opensymphony.com/xwork/wikidocs/Release%20Notes%20-%201.0.4.html>
- Documentation: <http://www.opensymphony.com/xwork/wikidocs/Documentation.html>
- Download: <https://xwork.dev.java.net/files/documents/709/8836/xwork-1.0.4.zip>

About XWork

XWork is a generic command pattern implementation and inversion of control (IoC) container. It is designed to be small, fast, and fit specifically for workflow-related usages, such as web application frameworks and workflow engines. WebWork provides the foundation for [WebWork](#), a Java-based web framework.

Besides WebWork, you can find integration support in [OSWorkflow](#) and [JPublish](#).

Core features include:

- Command pattern implementation
- Interceptor support allowing you to provide tight application integration
- Inversion of Control framework, helping you manage complex component dependencies
- Decoupled validation framework, letting you focus on disconnecting input validation from core business logic
- Extremely fast and flexible expression language based off of [OGNL](#)
- Advanced type conversion support, allowing you to never have to write data conversion code inside of your core business objects

1.0.5 Press Release

This page last changed on Dec 12, 2004 by [plightbo](#).

XWork 1.0.5 Released

The [OpenSymphony](#) group is proud to announce the release of XWork 1.0.5.

This release fixes a moderate security vulnerability, as well as a few minor configuration improvements.

- XWork: <http://www.opensymphony.com/xwork/>
- Release notes and changes:
<http://www.opensymphony.com/xwork/wikidocs/Release%20Notes%20-%201.0.5.html>
- Documentation: <http://www.opensymphony.com/xwork/wikidocs/Documentation.html>
- Download: <https://xwork.dev.java.net/files/documents/709/9668/xwork-1.0.5.zip>

Special Thanks

A very special thanks to Aleksandr Shneyderman for discovering the security vulnerability fixed in this version.

About XWork

XWork is a generic command pattern implementation and inversion of control (IoC) container. It is designed to be small, fast, and fit specifically for workflow-related usages, such as web application frameworks and workflow engines. WebWork provides the foundation for [WebWork](#), a Java-based web framework.

Besides WebWork, you can find integration support in [OSWorkflow](#) and [JPublish](#).

Core features include:

- Command pattern implementation
- Interceptor support allowing you to provide tight application integration
- Inversion of Control framework, helping you manage complex component dependencies
- Decoupled validation framework, letting you focus on disconnecting input validation from core business logic
- Extremely fast and flexible expression language based off of [OGNL](#)
- Advanced type conversion support, allowing you to never have to write data conversion code inside of your core business objects

1.1.2 Press Release













This page last changed on Mar 22, 2006 by [rainerh](#).


XWork 1.1.2 Released





















The [OpenSymphony](#) group is proud to announce the release of XWork 1.1.2.
This release is a bugfix release with some new improvements.

















- Parameterized interceptors
- Java5 support improvements
- Javadoc improvements
- Localization and I18n improvements
- New DoubleRangeFieldValidator
- Default action support for packages
- Improved type conversion














For a complete list of all the changes, please refer to the [complete changelog](#)

OpenSymphony JIRA (41 issues)										
T	Key	Summary	Assignee	Reporter	Pr	Status	Res	Created	Updated	Due
	XW-366	Improve log message in LocalizedTextUtil	Rainer Hermanns	Schava Eugene		 Closed	FIXED	Mar 21, 2006	Mar 21, 2006	
	XW-363	DefaultTextRenderer does not parse default message with a MessageFormat	Rainer Hermanns	Rainer Hermanns		 Closed	FIXED	Mar 17, 2006	Mar 17, 2006	
	XW-360	Setting allowed and blocked parameters for ParameterFilterInterceptor is not working as expected	Rainer Hermanns	Schava Eugene		 Closed	FIXED	Mar 16, 2006	Mar 21, 2006	
	XW-359	ConfigurationManager.clearConfigurationProviders & destroyConfiguration should	Rainer Hermanns	jeff		 Resolved	FIXED	Mar 15, 2006	Mar 15, 2006	

	call destroy appropriately with symbols ' and \'	Rainer Hermann	Schava Eugene		 Closed	FIXED	Mar 07, 2006	Mar 15, 2006
	Default action support is not existent	Andres March	Konstantin Pribluda		 Closed	IMPLEMENTED	Mar 06, 2006	Mar 18, 2006
	Another new unit test	Claus Ibsen	Claus Ibsen		 Closed	FIXED	Mar 05, 2006	Mar 07, 2006
	Some more unit tests and fixed a potential bug in TextProviderSupport ± polished javadocs	Rainer Hermann	Claus Ibsen		 Closed	FIXED	Mar 04, 2006	Mar 04, 2006
	Again some improved unit tests	Rainer Hermann	Claus Ibsen		 Closed	FIXED	Mar 04, 2006	Mar 04, 2006
	Class javadoc improved for util package	Rainer Hermann	Claus Ibsen		 Closed	FIXED	Mar 04, 2006	Mar 04, 2006
	Unit test of LoggingInterceptor is not executed	Rainer Hermann	Claus Ibsen		 Closed	FIXED	Mar 04, 2006	Mar 04, 2006
	Update clover.jar to newest version v1.3.12	Rainer Hermann	Claus Ibsen		 Closed	FIXED	Mar 04, 2006	Mar 07, 2006
	Class javadoc improved	Rainer Hermann	Claus Ibsen		 Closed	FIXED	Mar 03, 2006	Mar 04, 2006
	Clover report without log debug	Rainer Hermann	Claus Ibsen		 Closed	FIXED	Mar 03, 2006	Mar 04, 2006

	XW-348 Some improved unit tests of interceptors	Rainer Hermanns	Claus Ibsen	↓	 Closed	FIXED	Mar 01, 2006	Mar 02, 2006
	XW-347 I18nInterceptor is not unit tested	Rainer Hermanns	Claus Ibsen	↓	 Closed	FIXED	Mar 01, 2006	Mar 02, 2006
	XW-346 Missing copyright in some recent submitted javafiles	Rene Gielen	Claus Ibsen	↓	 Closed	FIXED	Mar 01, 2006	Mar 01, 2006
	XW-345 PrepareInterceptor is not unit tested	Rene Gielen	Claus Ibsen	↓	 Resolved	FIXED	Feb 27, 2006	Mar 01, 2006
	XW-344 ValidatorFactory - potential loss of stacktrace	Rene Gielen	Claus Ibsen	↓	 Closed	FIXED	Feb 27, 2006	Mar 01, 2006
	XW-343 RegexFieldValidator is not unit tested	Rene Gielen	Claus Ibsen	↓	 Closed	FIXED	Feb 27, 2006	Mar 01, 2006
	XW-342 DoubleRangeValidator configuration should not be locale aware for the given double parameters	Rene Gielen	Rene Gielen	↑	 Resolved	FIXED	Feb 26, 2006	Feb 26, 2006
	XW-339 typeconversion annotation key defaults to empty string	Rainer Hermanns	Tuomas Karkkkainen	↓	 Closed	FIXED	Feb 13, 2006	Mar 02, 2006
	XW-338 Incorrect handling of localized numbers while converting to primitive int	Alexandru Popescu	Michal Karwanski	↑	 Resolved	FIXED	Feb 13, 2006	Mar 10, 2006
	XW-337 Add	Rainer	Rainer	↑		FIXED	Feb 09,	Feb 15,

		@Key,	HermannHermanns		Closed	2006	2006
		@Element					
		and					
		@CreateIfNull					
		annotations					
	XW-336	XWork	Rainer katsumi ↑		FIXED	Feb 08, 2006	Feb 08, 2006
		type	Hermannstakahashi	Resolved			
		conversion					
		of					
		Milisecond					
		support					
	XW-335	@StringLiteral	Rainer Peter ↓		FIXED	Feb 07, 2006	Feb 08, 2006
		creates	HermannsWestlin	Resolved			
		log-warnings					
		when not					
		setting					
		property					
		maxLength					
	XW-332	Chain	AlexandruAlberto ↑		FIXED	Feb 02, 2006	Mar 03, 2006
		Result	Popescu Vilches	Resolved			
		throws					
		Exception					
		when you					
		use a					
		actionName					
		with a					
		method					
	XW-314	Create	UnassignedGabriel ↑		FIXED	Dec 22, 2005	Feb 09, 2006
		ParameterFilter	Intez Ziemer	Closed			
	XW-294	Downgrade	Rainer David ↑		FIXED	Jul 25, 2005	Mar 02, 2006
		error	HermannsCroft	Closed			
		logging					
		of					
		missing					
		result					
		classes					
	XW-291	configuration	Rainer Luigi R. ↓		FIXED	Jun 30, 2005	Mar 04, 2006
		improvement	HermannsViggiano	Closed			
		to detect					
		misconfigurations					
		(different					
		package					
		with					
		same					
		name)					
	XW-287	Cached	Rainer Tom ↓		FIXED	May 25, 2005	Mar 04, 2006
		validators	HermannsDavies	Closed			
		hang on					
		to their					
		old					
		contexts					
	XW-286	ognl	Rainer Mike ↑		FIXED	May 19, 2005	Feb 08, 2006
		parsing	HermannsMosiewicz	Closed			
		of values					

	XW-281	Query String containing large numbers causes exception	Alexandru Mike Popescu Mosiewicz			Resolved	FIXED	Apr 29, 2005	Mar 10, 2006
	XW-280	Configuration should distinguish an empty value from a missing value	Rainer John Hermanns Patterson			Closed	FIXED	Apr 23, 2005	Mar 04, 2006
	XW-277	Shouldn't LocalizedFormat log level be raised?	Rainer Grégory Hermanns Joseph			Closed	FIXED	Apr 19, 2005	Feb 20, 2006
	XW-269	Conversion bug when propertyName contains single quotes	Rainer Erik Plogi Hermanns			Resolved	NOT A PROBLEM	Feb 28, 2005	Feb 08, 2006
	XW-252	TextProvider should have more methods on its interface	Rainer Alex Hermanns Shneyderman			Closed	FIXED	Dec 10, 2004	Feb 15, 2006
	XW-248	OgnlUtil.class corrupts cglib proxied actions	Rainer sutter2k Hermanns			Closed	FIXED	Dec 04, 2004	Mar 02, 2006
	XW-223	Improve "No object in the CompoundRoot has a property named 'bar'" exception	Rainer Jeroen van Vianen Hermanns			Closed	FIXED	Sep 16, 2004	Mar 04, 2006
	XW-216	Create a DoubleRangeException	Rainer Ricardo Hermanns Echeta			Closed	FIXED	Sep 09, 2004	Feb 08, 2006



[XW-212](#) [serializable configuration](#) Rainer Stepha
Hermanns Chauvin



Resolved

FIXED

Aug 24, Feb 09,
2004 2006

About XWork

XWork is a generic command pattern implementation and inversion of control (IoC) container. It is designed to be small, fast, and fit specifically for workflow-related usages, such as web application frameworks and workflow engines. WebWork provides the foundation for [WebWork](#), a Java-based web framework.

Besides WebWork, you can find integration support in [OSWorkflow](#) and [JPublish](#).

Core features include:

- Command pattern implementation
- Interceptor support allowing you to provide tight application integration
- Inversion of Control framework, helping you manage complex component dependencies
- Decoupled validation framework, letting you focus on disconnecting input validation from core business logic
- Extremely fast and flexible expression language based off of [OGNL](#)
- Advanced type conversion support, allowing you to never have to write data conversion code inside of your core business objects

1.2.0 Press Release
















This page last changed on Aug 08, 2006 by [rainerh](#).































XWork 1.2.0 Released






















The [OpenSymphony](#) group is proud to announce the release of XWork 1.2.0. This release is a bugfix release with some new improvements.




























- Type converters on nested properties
- Java5 support improvements
- Line-precise error reporting



















For a complete list of all the changes, please refer to the [complete changelog](#)

OpenSymphony JIRA (37 issues)										
T	Key	Summary	Assignee	Reporter	Pr	Status	Res	Created	Updated	Due
	XW-400	i18n and ww:text tag can throw an exception under certain conditions	Rainer Hermanns	Nick Hill		 Resolved	FIXED	Jul 12, 2006	Jul 17, 2006	
	XW-396	Alter prepareInterceptor to call prepare{MethodName} when doing action!methodName	tm_je	tm_je		 Resolved	FIXED	Jul 03, 2006	Jul 04, 2006	
	XW-386	Xwork + SpringIOC. Xwork don't creates the result calling appContext.getBean("myResult").	tm_je	Fabio Falci		 Resolved	FIXED	Apr 17, 2006	Jul 22, 2006	
	XW-385	type conversion of array doesn't work	Unassigned	tm_je		 Resolved	WON'T FIX	Apr 16, 2006	May 21, 2006	
	XW-384	DefaultAction calls Rife every time	Andres March	Andres March		 Resolved	FIXED	Apr 13, 2006	Apr 13, 2006	

	XW-383 Validation configuration errors should fail-fast	Don Brown	Don Brown		 Resolved	FIXED	Apr 10, 2006	Apr 10, 2006
	XW-380 Debug String is output to System.out.println()	Rainer Hermanns	Corby Page		 Resolved	FIXED	Apr 02, 2006	May 10, 2006
	XW-379 Line-precision error reporting	Don Brown	Don Brown		 Resolved	FIXED	Apr 02, 2006	Apr 04, 2006
	XW-378 Make stack lookup failure better visible in devMode	Rene Gielen	Rene Gielen		 Closed	FIXED	Mar 30, 2006	Mar 30, 2006
	XW-377 com.opensymphony.cafebabel.LocalizedTextUtil generate warning when message is the same as key	Claus Ibsen	cafebabel LocalizedTextUtil		 Resolved	FIXED	Mar 30, 2006	Apr 09, 2006
	XW-376 ValidationClassSupport - synchronized - missing some methods	Claus Ibsen	Claus Ibsen		 Closed	FIXED	Mar 30, 2006	Apr 04, 2006
	XW-375 Conflicting validators	Claus Ibsen	Patrick Lightbody		 Resolved	FIXED	Mar 28, 2006	Jun 11, 2006
	XW-374 TimerInterceptor - support setting log category	Claus Ibsen	Claus Ibsen		 Resolved	IMPLEMENTED	Mar 27, 2006	Apr 04, 2006
	XW-373 xwork conversion validator should allow previous value to be displayed upon error in conversion	tm_je	tm_je		 Resolved	FIXED	Mar 27, 2006	Apr 02, 2006
	XW-372 Date Conversion	Rainer Hermanns	tm_je			FIXED	Mar 27, 2006	May 10, 2006

									Resolved	
		should be null instead of conversion error when http request is an empty string								
	XW-371	invalid mail address validation failed	tm_je	katsumi takahashi			Resolved	FIXED	Mar 23, 2006	Apr 27, 2006
	XW-365	Exception Closing Interceptor should allow logging the exception	Claudio Ibsen				Closed	IMPLEMENTED	Mar 21, 2006	Apr 04, 2006
	XW-340	Parameter for interceptors does not evaluated using ognl stack	Alexandru Popescu	Schava Eugene			Closed	NOT A PROBLEM	Feb 21, 2006	Jun 22, 2006
	XW-320	Add tiger and tiger-test targets to AntHill setup	Patrick Lightbody	Rainer Hermanns			Resolved	FIXED	Jan 13, 2006	Jul 10, 2006
	XW-297	Type Converter Can't Be Set On Nested Properties	Patrick Lightbody	Doug Seifert			Resolved	FIXED	Feb 10, 2004	Jul 24, 2006
	XW-296	Multiple components with same <class> but different <enabler> throws exception	Patrick Lightbody	Matthew Denner			Resolved	WON'T FIX	Apr 22, 2004	Jun 23, 2006
	XW-288	ONGL	Alexandru	John			Resolved	WON'T	Jun 09,	Jun 26,

		expression cache to use IdentityHashMap	Popescu Patterson		Closed	FIX	2005	2006
	XW-282	NPE thrown when trying to set a sub-property of a property that exists	AlexandruGanesh Popescu Bhattachan		 Closed	CANNOT REPRODUCE	Apr 29, 2005	Jun 22, 2006
	XW-279	OgnlValue too intolerent of NPEs	AlexandruKenny Popescu MacLeod		 Closed	WON'T FIX	Apr 22, 2005	Jun 22, 2006
	XW-278	Expression validation on a non-alias file.get validated on alias file as well, thus giving 2 error messages	Rainer Steve HermannsLoh		 Resolved	CANNOT REPRODUCE	Apr 21, 2005	May 21, 2006
	XW-271	Child property determination for localization	AlexandruAndrei Popescu Ivanov		 Resolved	FIXED	Mar 27, 2005	Jul 29, 2006
	XW-270	Interceptors supports IoC	Rainer Gang HermannsChan		 Closed	NOT A PROBLEM	Mar 01, 2005	Jun 20, 2006
	XW-260	xWork does not report syntax errors in OGNL expression	AlexandruAndriy Popescu Palamarchuk		 Closed	IMPLEMENTED	Apr 12, 2005	Jun 26, 2006
	XW-249	Enable external-refs elements on interceptors	Rainer Alex HermannsShneyderman		 Closed	WON'T FIX	Dec 07, 2004	Apr 11, 2006
	XW-233	Interceptors initialized twice	AlexandruJeroen Popescu van Vianen		 Closed	CANNOT REPRODUCE	Oct 15, 2004	Jun 22, 2006
	XW-208	short-circuiting	AlexandruFrancisco		 Closed	IMPLEMENTED	Apr 11, 2005	Jun 22, 2006

		of	Popescu	Hernandez		Closed	2004	2006	
		action-level							
		validators							
		before							
		field-level							
		validators							
		doesn't							
		halt							
		execution							
		of the							
		rest of							
		the							
		field-level							
		validators							
	XW-201	visitor	Jason	Francisco		 Resolved	WON'T FIX	Jun 18, 2004	May 21, 2006
		validation	Carreira	Hernandez					
		doesn't							
		work if							
		the							
		object to							
		be							
		validated							
		is null							
	XW-184	ability to	Alexandru	Francisco		 Closed	WON'T FIX	May 27, 2004	Jun 22, 2006
		pass	Popescu	Hernandez					
		parameters							
		to the							
		message							
		key in							
		the							
		validation							
		system							
	XW-176	Create	Alexandru	Mike		 Closed	WON'T FIX	May 06, 2003	Jun 22, 2006
		an	Popescu	Cannon-Brookes					
		InterceptorManager							
		to cache							
		interceptors							
	XW-169	Program	Jason	Simon		 Resolved	FIXED	Mar 08, 2004	Jun 23, 2006
		Configuration	Carreira	Stewart					
		of XWork							
	XW-108	XWorkBase	Alexandru	Mike		 Closed	FIXED	Oct 15, 2003	Jun 26, 2006
		improvement	Popescu	Mosiewicz					
	XW-102	RuntimeException	Rainer	Mike		 Closed	FIXED	Oct 06, 2003	Jul 13, 2006
		should be	Hermanns	Mosiewicz					
		replaced							
		with							
		proper							
		exception							
		handling							

About XWork

XWork is a generic command pattern implementation and inversion of control (IoC) container. It is

designed to be small, fast, and fit specifically for workflow-related usages, such as web application frameworks and workflow engines. WebWork provides the foundation for [WebWork](#), a Java-based web framework.

Besides WebWork, you can find integration support in [OSWorkflow](#) and [JPublish](#).

Core features include:

- Command pattern implementation
- Interceptor support allowing you to provide tight application integration
- Inversion of Control framework, helping you manage complex component dependencies
- Decoupled validation framework, letting you focus on disconnecting input validation from core business logic
- Extremely fast and flexible expression language based off of [OGNL](#)
- Advanced type conversion support, allowing you to never have to write data conversion code inside of your core business objects

About

This page last changed on Jun 22, 2004 by [plightbo](#).

About XWork

XWork is a generic command pattern implementation and inversion of control (IoC) container. It is designed to be small, fast, and fit specifically for workflow-related usages, such as web application frameworks and workflow engines. WebWork provides the foundation for [WebWork](#), a Java-based web framework.

Besides WebWork, you can find integration support in [OSWorkflow](#) and [JPublish](#).

Core features include:

- Command pattern implementation
- Interceptor support allowing you to provide tight application integration
- Inversion of Control framework, helping you manage complex component dependencies
- Decoupled validation framework, letting you focus on disconnecting input validation from core business logic
- Extremely fast and flexible expression language based off of [OGNL](#)
- Advanced type conversion support, allowing you to never have to write data conversion code inside of your core business objects

Sandbox

This page last changed on Feb 18, 2006 by [emolitor](#).

Sandbox

XML Invocation

This page last changed on Feb 18, 2006 by [emolitor](#).

Will document XML Invocation of actions here shortly.

XWork-1_2

This page last changed on Nov 22, 2006 by [tm_je](#).



XWork 1.2 documentation. Its now a copy of XWork2 documentation, since most parts are relevant except DI portion introduced. We are keeping a copy of this so that XWork 1.2 documentation could evolve separately with XWork2 documentations. Please feel free to post back any feedbacks. Feedbacks of any kind are very much welcomed. Cheers.

About XWork

- [Features](#)
- [Team](#)
- [Contributing](#)
- [Bugs and Issues](#)
- [Articles](#)
- [License](#)

Getting Started

- [Download](#)
- [Installation](#)
- [Requirements and Dependencies](#)
- [Tutorial](#)
- [Building XWork](#)

Reference

- [API](#)
- [Architecture](#)
- [Configurations](#)
- [Interceptors](#)
- [Package](#)
- [Actions](#)
- [Pre Result Listeners](#)
- [Results](#)
- [Object Factory](#)
- [Conversion](#)
- [Validation](#)
- [Profiling](#)
- [Value Stack](#)
- [Annotations](#)
- [Reloading](#)

Integration

- [Spring](#)

- [Hibernate](#)

Help

- [FAQs](#)

Building XWork1_2

This page last changed on Nov 22, 2006 by [tm_je](#).

XWork1_2 Actions

This page last changed on Nov 22, 2006 by [tm_jee](#).

XWork1_2 Annotations

This page last changed on Nov 22, 2006 by [tm_je](#).

XWork1_2 Architecture

This page last changed on Nov 22, 2006 by [tm_je](#).

XWork1_2 Articles

This page last changed on Nov 22, 2006 by [tm_jee](#).

XWork1_2 Configurations

This page last changed on Nov 22, 2006 by [tm_je](#).

XWork1_2 Conversion

This page last changed on Nov 22, 2006 by [tm_je](#).

XWork1_2 FAQs

This page last changed on Nov 22, 2006 by [tm_je](#).

XWork1_2 Features

This page last changed on Nov 22, 2006 by [tm_je](#).

XWork1_2 Hibernate Integration

This page last changed on Nov 22, 2006 by [tm_je](#).

XWork1_2 Installation

This page last changed on Nov 22, 2006 by [tm_je](#).

XWork1_2 Interceptors

This page last changed on Nov 22, 2006 by [tm_je](#).

XWork1_2 Object Factory

This page last changed on Nov 22, 2006 by [tm_je](#).

XWork1_2 package

This page last changed on Nov 22, 2006 by [tm_je](#).

XWork1_2 PreResultListeners

This page last changed on Nov 22, 2006 by [tm_jee](#).

XWork1_2 Profiling

This page last changed on Nov 22, 2006 by [tm_je](#).

XWork1_2 Reloading

This page last changed on Nov 22, 2006 by [tm_je](#).

XWork1_2 Requirements And Dependencies

This page last changed on Nov 22, 2006 by [tm_je](#).

Xwork1_2 Results

This page last changed on Nov 22, 2006 by [tm_jee](#).

Xwork1_2 Spring Integration

This page last changed on Nov 22, 2006 by [tm_jee](#).

XWork1_2 Tutorial

This page last changed on Nov 22, 2006 by [tm_je](#).

XWork1_2 Validation

This page last changed on Nov 22, 2006 by [tm_je](#).

XWork2

This page last changed on Nov 22, 2006 by [tm_je](#).



XWork2 documentation is out of date due to some stuff added to make it works with Struts2. Please feel free to post back any feedbacks. Feedbacks of any kind are very much welcomed. Cheers.

About XWork

- [Features](#)
- [Team](#)
- [Contributing](#)
- [Bugs and Issues](#)
- [Articles](#)
- [License](#)

Getting Started

- [Download](#)
- [Installation](#)
- [Requirements and Dependencies](#)
- [Tutorial](#)
- [Building XWork](#)

Reference

- [API](#)
- [Architecture](#)
- [Configurations](#)
- [Interceptors](#)
- [Package](#)
- [Actions](#)
- [Pre Result Listeners](#)
- [Results](#)
- [Object Factory](#)
- [Conversion](#)
- [Validation](#)
- [Profiling](#)
- [Value Stack](#)
- [Annotations](#)
- [Reloading](#)

Integration

- [Spring](#)
- [Hibernate](#)

Help

- [FAQs](#)

Interceptor Annotations

To use these annotations, you have to specify the [AnnotationWorkflowInterceptor](#) to your interceptor stack.

Annotation	Description
After Annotation	Marks a action method that needs to be executed before the result.
Before Annotation	Marks a action method that needs to be executed before the main action method.
BeforeResult Annotation	Marks a action method that needs to be executed before the result.

Validation Annotations

If you want to use annotation based validation, you have to annotate the class or interface with Validation Annotation.

These are the standard validator annotations that come with XWork-tiger:

Annotation	Description
ConversionErrorFieldValidator Annotation	Checks if there are any conversion errors for a field.
DateRangeFieldValidator Annotation	Checks that a date field has a value within a specified range.
DoubleRangeFieldValidator Annotation	Checks that a double field has a value within a specified range.
EmailValidator Annotation	Checks that a field is a valid e-mail address.
ExpressionValidator Annotation	Validates an expression.
FieldExpressionValidator Annotation	Uses an OGNL expression to perform its validator.
IntRangeFieldValidator Annotation	Checks that a numeric field has a value within a specified range.
RegexFieldValidator Annotation	Validates a regular expression for a field.
RequiredFieldValidator Annotation	Checks that a field is non-null.
RequiredStringValidator Annotation	Checks that a String field is not empty.
StringLengthFieldValidator Annotation	Checks that a String field is of the right length.
StringRegexValidator Annotation	

UrlValidator Annotation	Checks that a field is a valid URL.
Validation Annotation	Marker annotation for validation at Type level.
Validations Annotation	Used to group validation annotations.
VisitorFieldValidator Annotation	
CustomValidator Annotation	Use this annotation for your custom validator types.

Type Conversion Annotations

If the xwork-tigerjar is added to the classpath, you will directly have type conversion support for Maps and Collections 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.

If you want to use annotation based type conversion, you have to annotate the class or interface with the Conversion Annotation.

Annotation	Description
Conversion Annotation	Marker annotation for type conversions at Type level.
CreateIfNull Annotation	For Collection and Map types: Create the types within the Collection or Map, if null.
Element Annotation	For Generic types: Specify the element type for Collection types and Map values.
Key Annotation	For Generic types: Specify the key type for Map keys.
KeyProperty Annotation	For Generic types: Specify the key property name value.
TypeConversion Annotation	Used for class and application wide conversion rules.

AnnotationWorkflowInterceptor Interceptor

An error occurred:

https://opensymphony.dev.java.net/source/browse/*checkout*/xwork/tiger/src/java/com/opensymphony/xwork/inter

The system administrator has been notified.

Examples

An error occurred:

https://opensymphony.dev.java.net/source/browse/*checkout*/xwork/tiger/src/java/com/opensymphony/xwork/inter

The system administrator has been notified.

Configure a stack in xwork.xml that replaces the PrepareInterceptor with the AnnotationWorkflowInterceptor:

An error occurred:

https://opensymphony.dev.java.net/source/browse/*checkout*/xwork/tiger/src/java/com/opensymphony/xwork/inter

The system administrator has been notified.

Given an Action, AnnotatedAction, add a reference to the AnnotationWorkflowInterceptor interceptor.

```
<action name="AnnotatedAction" class="com.examples.AnnotatedAction">
  <interceptor-ref name="annotationInterceptor"/>
  <result name="success" type="freemarker">good_result.ftl</result>
</action>
```

An error occurred:

https://opensymphony.dev.java.net/source/browse/*checkout*/xwork/tiger/src/java/com/opensymphony/xwork/inter

The system administrator has been notified.

Building XWork

This page last changed on Oct 13, 2006 by [tm_jee](#).



Building XWork2 requires JDK 1.5 (Tiger)

Fresh check out from SVN

Check out from OpenSymphony's SVN using the following command

```
svn --username xxx --password xxxx checkout https://svn.opensymphony.com/svn/xwork/trunk
./xwork
```

or without username and password

```
svn checkout https://svn.opensymphony.com/svn/xwork/trunk ./xwork
```

Building XWork2 using Maven2

To build XWork2 one will need Maven2 which could be obtained [here](#). More information about Maven2 could be obtained [here](#).

To generate a packaged jar file, run

```
mvn clean package
```

This will do some clean up, compile (both the core and tests) and the core as a jar file under /target directory

To install it in one's local repository simply run,

```
mvn clean install
```

Release distribution

The release distribution could be downloaded [here](#)

After downloading it simply unzip it using

```
unzip ./xwork-xxx.zip -d ./xwork
```


Reloading

This page last changed on Nov 11, 2006 by [tm_je](#).

It is possible to configure XWork2 to keeps reloading its configuration files (eg. xwork.xml, *conversion.xml, *-validation.xml). This is done as follows:

```
FileManager.setReloadingConfigs(true)
```

Configuration reloading by default is "off".



Configuration reloading could cause performance issue. It might most probably causes the application to run considerably slow.

XWork Actions

This page last changed on Oct 15, 2006 by [tm_je](#).

XWork2 Action serves as to capture a particular execution logic. It is distinguished by its unique name and namespace combination.



XWork2 parses its xml configuration according to order. If an action extends off another parent action, the parent action MUST be defined before the action that planned to extend it

Name and namespace

An action is identified by its name and namespace combination. By default, if an action with a specified name and namespace does not exist, it will fall back to an action with the same name but a default namespace (empty namespace) if such a combination exists.

```
<xwork>
  <package name="default" namespace="/myNamespace">
    ...
    <action name="myAction" ...>
      ....
    </action>
    ...
  </package>
</xwork>
```

Action class

An action class could be :-

- plain pojo (extends java.lang.Object which by default every java class does)
- implements com.opensymphony.xwork2.Action interface
- extends com.opensymphony.xwork2.ActionSupport

Action Context

Action could access its context using ActionContext. Parameter passing could be done using this action context. Following is an example

```
ActionProxyFactory factory = ActionProxyFactory.getFactory();
factory.createActionProxy(configuration, "actionAlias", "/namespace",
    new LinkedHashMap() {
        {
            // pass in parameter
            put("someParameter", "some parameter value");
        }
    }
);

// latter on, we could do this
```



```
ActionContext context = RequestContext.getContext();  
Map contextMap = context.getContextMap();  
contextMap.get("someParameter");
```

XWork2 is designed around the concept of GOF's Command Design Pattern.

Command Design Pattern

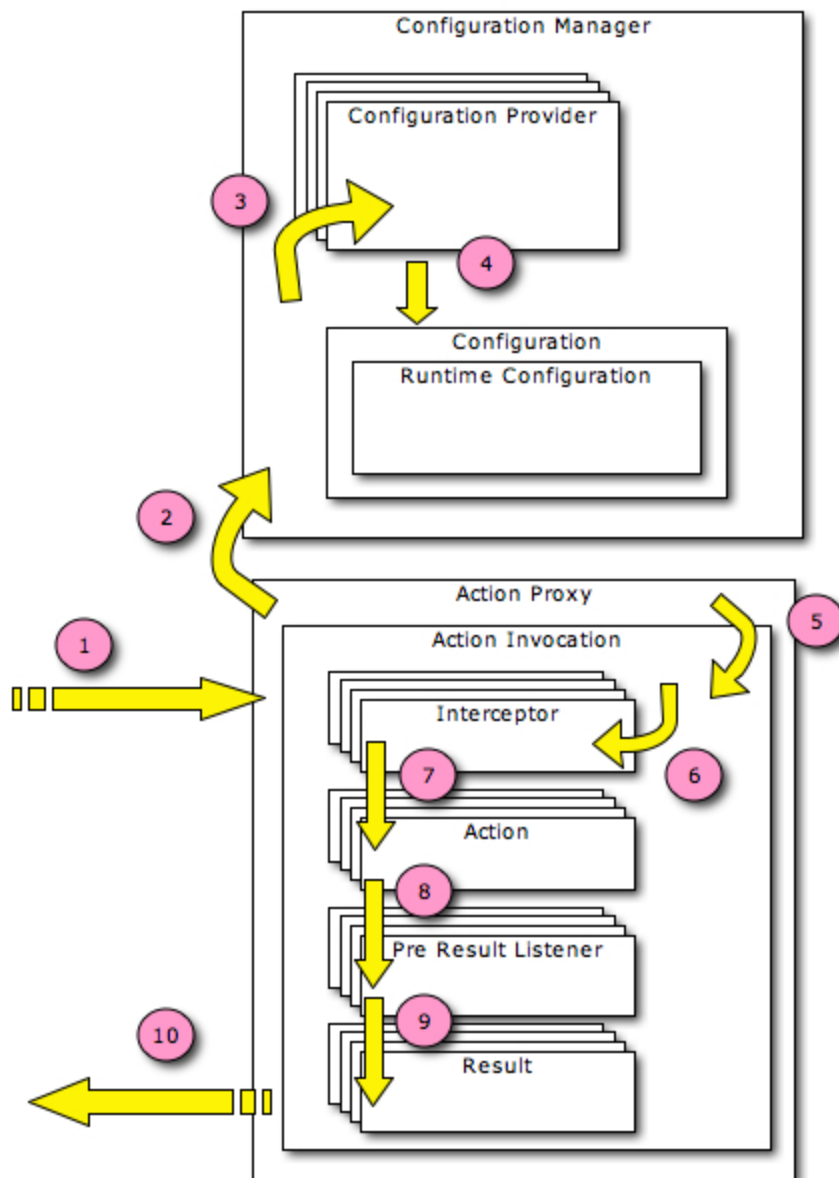
General

A Command design pattern, abstract the logic of a particular execution into an object, the command object. If an originator is to interact with a receiver using a command pattern, direct reference between them could be abstracted away. Originator would probably create a command object, that when executed will know how to get its receiver and perform the necessary logics. There could be a variety of command that being able to communicate with a variety of receivers, most probably created through some sort of Factory pattern.

Relation with XWork2

Command in XWork2 sense would be XWork2 Action, while the originator would most possibly translate to the client that grab hold of a copy of XWork2 Action. A receiver will most likely be XWork2 Result. XWork2 enhance the concept a bit with the introduction of ActionProxy, a thin layer representing the action such that action could be loaded differently and ActionInvocation which details the actual execution of the action itself.

Architecture



XWork Articles

This page last changed on Oct 13, 2006 by [tm_jee](#).



Anybody who have an article about XWork2, please feel free to drop me a line and I'll have it up in here. Cheers.

// TODO: Articles related to XWork2

XWork Configurations

This page last changed on Oct 14, 2006 by [tm_je](#).

Configuring XWork2 centers around the following classes:-

- ConfigurationManager
- ConfigurationProvider
- Configuration

where by both ConfigurationProvider and Configuration are java interface.

ConfigurationManager

This is the center for configuring XWork2. It allows ConfigurationProvider to be plugged in and setting a custom Configuration to be used. Typically one would like to just create one instance of ConfigurationManager for a XWork2 usage.

Code

To create a ConfigurationManager :-

```
ConfigurationManager confManager = new ConfigurationManager();
```

ConfigurationProvider

ConfigurationProvider helps configuring a Configuration, populating it with information regarding what actions it has, how does the results get mapped and what interceptors are there and how they are related to each action etc. A default ConfigurationProvider that comes with XWork2 would be XmlConfigurationProvider and as the name suggest populate information into a Configuration object according the the xml provided.

Code

To create a plug in a custom configuration provider

```
ConfigurationManager confManager = new ConfigurationManager();
confManager.addConfigurationProvider(
    new MyCustomConfigurationProvider(...));
```

To create an XmlConfigurationProvider that points to a particular xml file in the classpath:-

```
ConfigurationManager confManager = new ConfigurationManager();
confManager.addConfigurationProvider(
```

```
new XmlConfigurationProvider("foo/bar/myConf.xml");
```

Configuration

Configuration is a typical value object that contains configuration information. There's only one instance of it for each ConfigurationManager, where it is passed to different ConfigurationProvider in order for information to get stored in it. The default implementation would be DefaultConfiguration.

Code

To plug in a custom Configuration

```
ConfigurationManager confManager = new ConfigurationManager();  
confManager.setConfiguration(new MyCustomConfiguration(...));
```

The following illustrates typically how XWork2 is configured.

```
// this is in SVN Head, we need to get the snippet stuff fix, so we could just grab it from the  
svn
```

XWork Conversion

This page last changed on Oct 14, 2006 by [tm_je](#).

- how conversion works (plugin in through Ognl)
- how to have custom converters
- examples

XWork2 supports validation at a global and action level

How it works

XWork2 make use of Ognl to do its conversion, by plugging in XWork2 specific PropertyAccessor, MethodAccessor, NullHandler and TypeConverter etc. into Ognl runtime.

Global Level

To create a global level conversion, create a file called 'xwork-conversion.properties' at the root level of classpath. The entry in 'xwork-conversion.properties' should be a key-value pair just like any Java property file, where the key represents the FQN (Fully-Qualified-Name) of a particular object where conversion is to be done, whereas the value would be the FQN of the XWork2 converter itself.

The following is an example of a hypothetical 'xwork-conversion.properties'

```
....
java.lang.Boolean=foo.bar.MyBooleanConverter
foo.bar.MyObject=foo.bar.MyObjectConverter
...
```

Action Level

To create an action-level conversion, create a file called 'ActionClassName-conversion.properties' in the same location at the classpath where the Action class itself resides. Eg. if the action class name is MyAction, the action-level conversion properties file should be named 'MyAction-conversion.properties'.

Assuming that 'MyAction' action class looks as follows. Note the properties it has.

```
public class MyAction extends ActionSupport {
    private Boolean myBool;
    private Double myDouble;

    public Boolean getMyBool() { return this.myBool; }
    public void setMyBool(Boolean myBool) { this.myBool = myBool; }

    public Double getMyDouble() { return this.myDouble; }
    public void setMyDouble(Double myDouble) { this.myDouble = myDouble; }

    .....
}
```

The following would be an example of a hypothetical 'MyAction-conversion.properties'.

```
myBool=foo.bar.MyBooleanConverter
myDouble=foo.bar.MyDoubleConverter
```

Custom Converter

The following is an example of a converter. It should extend DefaultTypeConverter (part of Ognl).

```
public class MyConverter extends DefaultTypeConverter {
    public Object convertValue(Map context, Object target, Member member, String propertyName,
                               Object value, Class toType) {
        // type conversion goes here.
        ....
    }
}
```


XWork FAQs

This page last changed on Oct 14, 2006 by [tm_jee](#).



Please feel free to add any FAQs that's appropriate

General

Interceptor

Action

Result

PreResultListener

Conversion

Validation

Configuration

XWork Features

This page last changed on Oct 10, 2006 by [tm_jee](#).

// TODO: Mention Features of XWork2 here

XWork Hibernate Integration

This page last changed on Nov 11, 2006 by [tm_je](#).

XWork2 could have Hibernate integration as well. However such an integration is not build in by default. In other words, it doesn't come bundled with XWork2 itself.

Senario

Typical senario would be to have Hibernate session prepared before the action is actually executed and clear after the action and result are done executing. This could be handled by a XWork2 interceptor.

Code

```
public class HibernateInterceptor implements Interceptor {
    ...
    public String intercept(ActionInvocation invocation) throws Exception {
        Session session = null;
        try {
            // please refer to Hibernate site for implementation of HibernateUtil
            (www.hibernate.org)
            session = HibernateUtil.getSessionFactory().getCurrentSession();
            session.beginTransaction();
            Object action = invocation.getAction();
            if (action instanceof HibernateSessionAware) {
                ((HibernateSessionAware) action).setHibernateSession(session);
            }
            invocation.invoke();
            session.getTransaction().commit();
        }
        catch(Exception e) {
            exceptionHandler(e);
            session.getTransaction().rollback();
        }
    }
    ...
    /**
     * Hook for subclass to handle exception.
     */
    protected void handleException(e) {
        // maybe we could do better than this.
        LOG.error(e.toString(), e);
    }
    ...
}
```

```
public interface HibernateSessionAware {
    void setHibernateSession(Session session);
}
```

The above is just a simple code to show one possibility of how Hibernate / XWork2 integration could be done.

XWork Installation

This page last changed on Oct 13, 2006 by [tm_jee](#).



XWork2 requires at least JDK 5 (Tiger) to work.

To start using XWork2, it is required to have the folloing in the classpath :-

- XWork2 jar
- xwork.xml
- [Other jar dependencies](#)

Using command line

Assuing the following structure.

```
+ classes/  
  + xwork-xxx.jar  
  + xwork.xml  
  + otherJars.jar  
  + myPackage/  
    + MyMain.class
```

```
// Linux / Unix like environment  
java -cp xwork-xxx.jar:xwork.xml:otherJars.jar myPackage.MyMain  
  
// Windows  
java -cp xwork-xxx.jar;xwork.xml;ohterJars.jar myPackage.MyMain
```

Using ant

Assuming the following structure

```
+ src  
  + myPackage  
    + MyMain.java  
+ lib  
  + xwork-xxx.jar  
  + ... jar  
+ classes  
  + myPackage  
    + MyMain.class  
+ resources  
  + xwork.xml
```

```
<project ... >  
  ...  
  <path id="classpath">  
    <fileset dir="lib">  
      <include name="**/*.jar"/>  
    </fileset>  
    <pathelement location="resources"/>
```

```
</path>
....
<target ...>
  <!-- your target refer to path with id as 'classpath' when necessary -->
</target>
```

Using maven2

When using maven2, one might want to have the following dependencies

```
<pom ...>
....
  <dependency>
    <groupId>commons-logging</groupId>
    <artifactId>commons-logging</artifactId>
    <version>1.0.4</version>
  </dependency>
  <dependency>
    .... other dependencies
  </dependency>
....
</pom>
```

XWork Interceptors

This page last changed on Oct 14, 2006 by [tm_je](#).

Interceptor is a core component with XWork2. XWork2 has been designed with action interception in mind from ground up. Basically an interceptor wraps around XWork2 Action. It is capable of influencing the overall result, and hence affecting which result eventually gets executed. In the process it has access to XWork2 Action, its proxy (ActionProxy), ActionInvocation etc. such that the interceptor could inspect the internals of XWork2 before coming up with an overall conclusion of what it should do.

Default interceptors that come with XWork2

```
// TODO: pull from svn
```

Configuring interceptor in xml

Referring each interceptor separately

```
// code is in svn head, we need to get the snippet fixed, and pulled it from the svn
```

Referring to interceptors using interceptor-stack

```
// code is in svn head, we need to get the snippet fixed, and pulled it from the svn
```

Overriding parameter of individual interceptor

```
// code is in svn head, we need to get the snippet fixed, and pulled it from the svn
```

Overriding parameter of interceptor in interceptor-stack

```
// code is in svn head, we need to get the snippet fixed, and pulled it from the svn
```

Default interceptors

```
// todo: pull from snippet  
// snippet name 'xwork2-default-interceptors'  
// from xwork-default.xml
```

XWork Object Factory

This page last changed on Oct 14, 2006 by [tm_jee](#).

ObjectFactory is an abstraction from XWork2 responsible for all the creation of XWork2 related objects. Its how XWork2 integrate with IoC containers like Spring, Pico, Plexus etc.

Customization

To customize an ObjectFactory, simply have a static block that set a custom instance of ObjectFactory as follows :-



Custom ObjectFactory must extends ObjectFactory or any of its child.

```
static {  
    ObjectFactory.setObjectFactory(new CustomObjectFactory(...));  
}
```

XWork package

This page last changed on Oct 14, 2006 by [tm_je](#).

XWork2 introduce the notion of package to group related action under the same namespace. Its much like how java package being used to group related class. Just that XWork2 package supports multiple inheritance.

Namespace

XWork2 package defines the namespace its containing action is under. Without this attribute specified, the action will be automatically considered to be under the default (empty) namespace.

```
<xwork>
...
<package name="default" namespace="/myNamespace">
...
</package>
...
</xwork>
```

Abstraction

An action can be abstract, meaning that it will not be recognized by XWork2 but it does exists. This mean to identify action whose sole purpose tends to be a convenience for other action to subclass.

```
<xwork>
...
<package name="default-abstract" abstract="true">
...
</package>

<package name="default" extends="default-abstract">
...
<!-- more specific action here. -->
</package>
...
</xwork>
```

Extendability

XWork2 action support inheritance such that an action could extend form anohter parent action. XWork2 action could extends off multiple action. The extending action will inherit all its parent package s action, interceptor, result-types etc.

```
<xwork>
...
<package name="default" namespace="/myNamespace" extends="parent1-default, parent2-default">
...
</package>
...
</xwork>
```


XWork PreResultListeners

This page last changed on Oct 14, 2006 by [tm_je](#).

PreResultListener serves as a hook to influence the current action invocation before the result is executed, pretty much clear from the name itself.

Example

Through Action class

```
public class MyAction extends ActionSupport {
    ...
    public String execute() throws Exception {
        ActionContext context = ActionContext.getContext();
        ActionInvocation invocation = context.getActionInvocation();
        invocation.addPreResultListener(
            new PreResultListener() {
                public void beforeResult(ActionInvocation invocation, String resultCode) {
                    // do my magic here
                }
            });
    }
    ...
}
```

Through Interceptor class

```
public MyInterceptor implements Interceptor {
    ....
    public String intercept(ActionInvocation invocation) throws Exception {
        invocation.addPreResultListener(
            new PreResultListener() {
                public void beforeResult(ActionInvocation invocation, String resultCode) {
                    // do my magic here
                }
            });
    }
    ....
}
```

XWork Profiling

This page last changed on Oct 16, 2006 by [tm_je](#).

XWork2 supports build-in profiling.

Profiling aspects

XWork2 profiling aspects involves the following :-

- creation of DefaultActionProxy
 - creation of DefaultActionInvocation
 - creation of Action
- execution of DefaultActionProxy
 - invocation of DefaultActionInvocation
 - invocation of Interceptors
 - invocation of Action
 - invocation of PreResultListener
 - invocation of Result

Activating / Deactivating Profiling

Activating / Deactivating of the profiling feature could be done through:-

Through System property

```
-Dxwork.profile.activate=true
```

This could be done in the container startup script eg. CATALINA_OPTS in catalina.sh (tomcat) or using "java -Dxwork.profile.activate=true -jar start.jar" (jetty)

Through code

```
UtilTimerStack.setActivate(true);  
  
// or  
  
System.setProperty("xwork.profile.activate", "true");  
  
// or  
  
System.setProperty(UtilTimerStack.ACTIVATE_PROPERTY, "true");
```

This could be done in a static block, in a Spring bean with lazy-init="false", in a Servlet with init-on-startup as some numeric value, in a Filter or Listener's init method etc.

Filtering profile information

One could filter out the profile logging by having a System property as follows. With this 'xwork.profile.mintime' property, one could only log profile information when its execution time exceed those specified in 'xwork.profile.mintime' system property. If no such property is specified, it will be assumed to be 0, hence all profile information will be logged.

```
-Dxwork.profile.mintime=10000
```

Write profiling code

One could extends the profiling feature provided by Struts2 in their web application as well.

Using UtilTimerStack's push and pop

```
String logMessage = "Log message";
UtilTimerStack.push(logMessage);
try {
    // do some code
}
finally {
    UtilTimerStack.pop(logMessage); //this needs to be the same text as above
}
```

Using a UtilTimerStack's ProfileBlock template

```
String result = UtilTimerStack.profile("purchaseItem: ",
    new UtilTimerStack.ProfileBlock<String>() {
        public String doProfiling() {
            // do some code
            return "Ok";
        }
    });
```

Profiling Log files

Profiled result is logged using commons-logging under the logger named 'com.opensymphony.xwork2.util.profiling.UtilTimerStack'. Depending on the underlying logging implementation say if it is Log4j, one could direct the log to appear in a different file, being emailed to someone or have it stored in the db.

XWork Requirements And Dependencies

This page last changed on Oct 13, 2006 by [tm_jee](#).



The version number specified are being used to build and test XWork2, later version might most probably work, but there is no guarantee.

Mandatory dependencies

Libraries	Version	Scope
commons-logging	1.0.3	Runtime
ognl	2.6.7	Runtime
spring-mock	1.2.6	Test
junit	3.8.1	Test
mockobjects-core	0.09	Test
easymock	2.0	Test

Optional dependencies

Libraries	Version
rife-Continuations	0.0.2
spring-core	1.2.6
spring-aop	1.2.6
spring-beans	1.2.6
spring-context	1.2.6
spring-web	1.2.6
cglib	2.1

Xwork Results

This page last changed on Oct 14, 2006 by [tm_je](#).

Result represents the 'receiver' in a command pattern. It might be used to generate a response for a webapp, render ui for a Swing base app, execute a batch update etc.

Code

XWork2 result should extends off Result interface. The custom result could have parameter injected into it from xwork.xml file as well, just be sure to have getter / setters for those parameters.



The parameter needs to be String, however one could write up a Abstract result which allows syntax like `${...}` to be evaluated against XWork2 Ognl Value Stack. `StrutsResultSupport` does that actually. In order to help out with that, XWork2 have an utility class called `TextParseUtil` that does that parsing functionality.

```
public class MyResult implements Result {
    // let's have getter/setter for our 'injectable' parameter.
    private String myParam;
    public String getMyParam() { return this.myParam; }
    public void setMyParam(String myParam) { this.myParam = myParam; }

    ....
    public void execute(ActionInvocation invocation) throws Exception {
        ....
    }
    ....
}
```

and registered it in xwork.xml before using it

```
<xwork>
  <package name="myPackage" ...>
    <!-- registre our custom result -->
    <result-types>
      <result-type name="myResult" class="foo.bar.MyResult">
        <param name="myParam">some param value</param>
      </result>
    </result-types>

    <action ....>
      <!-- Now we could use it -->
      <result type="myResult" ...>
        ....
      </result>
    </action>
  </package>
</xwork>
```

Default result types

```
// TODO: pull from svn using snippet
// snippet id: 'xwork2-default-result-types'
// location xwork-default.xml
```


Xwork Spring Integration

This page last changed on Nov 11, 2006 by [tm_je](#).

XWork2 allows its "components" like XWork2's action, interceptor, results etc. to be configured using Spring.

Configuration

To configured XWork2 to integrate with Spring, one need to use `SpringObjectFactory`. XWork2 "components" are created through `ObjectFactory`. `SpringObjectFactory` and its subclass are extension of `ObjectFactory` coded for Spring integration.

```
static {
    // setup Spring's ApplicationContext, probably an ClassPathXmlApplicationContext or others.
    ApplicationContext appContext = ....

    // create XWork2's ObjectFactory for Spring integration
    SpringObjectFactory springObjectFactory = new SpringObjectFactory();

    // hook up Spring's ApplicationContext
    springObjectFactory.setApplicationContext(xmlConfigurationApplicationContext);

    // set our auto-wiring strategy
    springObjectFactory.setAutowiringStrategy(AutowireCapableBeanFactory.AUTOWIRE_BYNAME);

    // do we want to cache classes loaded through Spring?
    springObjectFactory.setUseClassCache(true);

    ObjectFactory.setObjectFactory(springObjectFactory);
}
```

Hereafter, whenever XWork2 needs an object it will go through `ObjectFactory` which will allows Spring beans to be returned.

Integration

XWork's Action, Interceptor, Result class name are actually interpreted as Spring beans name. For example,

```
<xwork ...>
  <package ...>
    <result-types>
      ...
      <result name="myResult" class="myResultBean" />
      ...
    </result-types>

    ...
    <interceptors>
      ...
      <interceptor name="myInterceptor" class="myInterceptorBean" />
      ...
    </interceptors>
    ...

    <action name="myAction" class="myActionBean">
      ...
      <interceptor-ref name="myInterceptor" />
      ...
    </action>
  </package>
</xwork>
```



```
        <result name="success" type="myResult" />
        ...
    </action>
</package>
</xwork>
```

```
<beans>
    <!-- XWork2 Result instantiated, wired using Spring -->
    <bean name="myResultBean" class="..." singleton="false">
        ....
    </bean>

    <!-- XWork2 Interceptor instantiated, wired using Spring -->
    <bean name="myInterceptorBean" class="..." singleton="false">
        .....
    </bean>

    <!-- XWork2 Action instantiated, wired using Spring -->
    <bean name="myActionBean" class="..." singleton="false">
        ....
    </bean>
</beans>
```



XWork2 action needs to be configured with `singleton="false"`, cause XWork2 expect Action a new instance for each request / invocation



XWork2 result is preferably configured with `singleton="false"`, unless some information needs to be kept between invocation. By default, without using `SpringObjectFactory`, a new instance of Result would be created per invocation.



XWork2 interceptors are created once and being repeatably used unless XWork2 is configured to be reloadable. It is preferable that XWork interceptors being configured with `singleton="false"` as well.

With this configuration, XWork2's "components" could be instantiated and possibly have its dependencies wired-up through Spring.

XWork Tutorial

This page last changed on Oct 13, 2006 by [tm_jee](#).

// TODO: list down tutorials. Each tutorial should be a link to its own page i guess.

- [Hello World](#)

XWork2 Hello World Tutorial

This page last changed on Oct 14, 2006 by [tm_jee](#).



The code is in XWork2 SVN head under showcase directory, we need to get the snippet point to there so we could have the code displayed

The objective of this simple 'Hello World' tutorial is to be a gentle introduction of the components in xwork2. Its basically just to show how to execute a command (an action) that simply prints out 'Hello World' the XWork2 way.

Step 1: Write up xwork.xml

Step 2: Code it up

Step 2a: Let XWork2 read xwork.xml

Step 2b: Obtain a ready to be executed Action

Step 2c: Execute the action

XWork Validation

This page last changed on Oct 14, 2006 by [tm_je](#).

XWork2 have build-in validation. They are done through the following major classes:-

- ActionVaidatorManagerFactory
- ActionValidatorManager
- Validator

ActionValidatorManagerFactory

ActionValidatorManagerFactory serves as a static factory for ActionValidatorManager.

ActionValidatorManager

This class is the main entry point to trigger XWork2 validation. Its default implementation (DefaultActionValidatorManager also takes care of loading the validators defined through xml file (default.xml) located in XWork2 jar file)

To trigger validation through code use:-

```
ActionValidatorManager actionValidatorManager = ....
actionValidatorManager.validate(myAction, "actionAlias");
```

Validator

A validator needs to implement the Validator interface. XWork2 provides supporting base class that makes creating a custom validator easier. Typically one would like to extends from either :-

- ValidatorSupport
- FieldValidatorSupport

Extending ValidatorSupport is ideal for if its a global level validator while extending from FieldValidatorSupport is for validator that are meant to validate at a field level.

How to Write a Custom Validator

Global validator

```
// pick up from svn
// id: 'global-level-validator'
// location: ExpressionValidator
```

Field validator

```
// pick up from svn
// id: 'field-level-validator'
// location: DoubleRangeFieldValidator
```

Default Validators

```
// pick up from svn
// id: validators-default
// location: default.xml
```

Integrate validation into Action

Action-level validation

To create an action level validation, create a file 'ActionClass-validation.xml' at the same location where the Action class itself lies. Example, if the action class is `MyAction`, the xml file would be named 'MyAction-validation.xml'

```
<action name="myAlias" class="foo.bar.MyAction">
  ....
</action>
<action name="myAnotherAlias" class="foo.bar.MyAction" method="create">
  ....
</action>
```

An action-level validation allows the validation to be applied to all 'MyAction' action class. In the example above, both action 'myAlias' and 'myAnotherAlias' will have the validation applied.

Action Alias-level validation

To create an action alias level validation, create a file 'ActionClass-actionAlias-validation.xml' at the same location where the Action class itself lies. Example if the action class is `MyAction` with an alias 'myAlias', the xml file would be named 'MyAction-myAlias-validation.xml'.

```
<action name="myAlias" class="foo.bar.MyAction">
  ...
</action>
<action name="myAnotherAlias" class="foo.bar.MyAction" method="create">
  ....
</action>
```

An action-alias-level validation allows the validation to be applied to all 'MyAction' action class with alias 'myAlias' only, hence limiting the scope where the validation is applied compared to global-level validation.

In the example above, action 'myAlias' will have the validation applied whereas action 'myAnotherAlias'

will not.

Validation configuration

Following is an example how a validation xml configuration file looks like

```
// pick up from svn
// id=xwork-validation
// location: showcase -> ValidationExampleAction-validation.xml
```

XWork Value Stack

This page last changed on Nov 11, 2006 by [tm_je](#).

Value stack is a fundamental part of XWork2. As XWork2 processed a command request, objects of interest could be pushed into the value stack or set into its context. This could be done by XWork2 itself, Action, Interceptors, Results etc.



XWork2 Action itself is actually pushed into the stack during the invocation process.

Working Concept

There are two ways objects could be store in XWork2's value stack.

Top of the value stack

Objects get push into XWork2 value stack in a first-in-last-out fashion just like any ordinary stack would.

To push an Object into XWork's value stack, simply do

```
valueStack.push(anObject);
```



Actually XWork2 uses Ognl underneath. By having a CompoundRoot that allows objects to be stack up on as Ognl root, the effect of a stack is achieved.

When value the stack is queried for object using the following method signature eg.

```
valueStack.findString(String);  
valueStack.findValue(String);  
valueStack.findValue(String, Class);
```

the objects residing the XWork2 value stack will be search accordingly (with those in the top of the stack having higher precedence).

For example, with the following value stack,

Value Stack
objectA (foo.bar.ObjectA) <ul style="list-style-type: none">• method0ne• method1• method2
objectB (foo.bar.ObjectB)

- methodTwo
- method3
- method4

objectC (foo.bar.ObjectC)

- methodOne
- methodTwo
- method5
- method6

Case 1

```
Object o = findValue("method2");
```

In this case, the "methodA" of instance foo.bar.ObjectA will be invoked with its returning object returned.

Case 2

```
Object o = findValue("method6");
```

In this case, the "method6" of instance foo.bar.ObjectC will be invoked with its returning object returned.

Case 3

```
Object o = findValue("methodOne");
```

In this case, the "methodOne" of instance foo.bar.ObjectA will be invoked with its returning object returned. This is due to instance of foo.bar.ObjectA being on the top of the stack compared to instance of foo.bar.ObjectC, XWork2 searches down the stack and hence will find instance of foo.bar.ObjectA first.

Case 4

```
Object o = findValue("methodTwo");
```

In this case, the "methodTwo" of instance foo.bar.ObjectB will be invoked with its returning object returned. This is due to instance of foo.bar.ObjectB being on the top of the stack compared to instance of foo.bar.ObjectC, XWork2 searches down the the stack and hence will find instance of foo.bar.ObjectB first.

Value stack's context

To store an object in XWork2's value stack's context, one could use


```
Map context = valueStack.getContext();  
context.put("key", someObject);
```

To query Object from XWork2's value stack's context, one could use

```
valueStack.findValue("#key");
```

or

```
valueStack.getContext().get("key");
```