Space Details

Key:	XW
Name:	XWork
Description:	
Creator (Creation Date):	plightbo (Apr 18, 2004)
Last Modifier (Mod. Date):	phil (Jan 03, 2007)

Available Pages

- Documentation 🏠
  - Annotations
    - After Annotation
    - AnnotationWorkflowInterceptor
    - Before Annotation
    - BeforeResult Annotation
    - Conversion Annotation
    - ConversionErrorFieldValidator Annotation
    - CreateIfNull Annotation
    - CustomValidator Annotation
      - ValidationParameter annotation
    - DateRangeFieldValidator Annotation
    - DoubleRangeFieldValidator Annotation
    - Element Annotation
    - EmailValidator Annotation
    - ExpressionValidator Annotation
    - FieldExpressionValidator Annotation
    - IntRangeFieldValidator Annotation
    - Key Annotation
    - KeyProperty Annotation
    - RegexFieldValidator Annotation
    - RequiredFieldValidator Annotation
    - RequiredStringValidator Annotation
    - StringLengthFieldValidator Annotation
    - StringRegexValidator Annotation
    - TypeConversion Annotation
    - UrlValidator Annotation
    - Validation Annotation
    - Validations Annotation
    - VisitorFieldValidator Annotation
  - Building XWork
  - Colophon
  - Configuring XWork in xwork.xml

- Action configuration
- Include configuration
- Interceptor Configuration
- Namespace Configuration
- Package Configuration
- Result Configuration
- Creating a distribution
- Documentation Style Guide
- Precise Error Reporting
- Reloading
- XWork Actions
- XWork Architecture
- XWork Articles
- XWork Configurations
- XWork Conversion
- XWork Core Concepts
- XWork FAQs
- XWork Features
- XWork Hibernate Integration
- XWork Installation
- XWork Interceptors
- XWork Localization
- XWork Object Factory
- XWork package
- XWork PreResultListeners
- XWork Profiling
- XWork Requirements And Dependencies
- Xwork Results
- XWork specific OGNL Features
- Xwork Spring Integration
- XWork Tutorial
  - XWork2 Hello World Tutorial
- XWork Validation
- XWork Value Stack

Documentation

This page last changed on Jan 03, 2007 by phil.

> ⚠ XWork2 documentation is out of date due to some stuff added to make it works with Struts2. The XWork1 documentation is merged into WebWork

About XWork

- Features
- Concepts
- Team
- Contributing
- Bugs and Issues
- Articles
- License

Getting Started

- Download
- Installation
- Requirements and Dependencies
- Tutorial

Reference

- API
- Architecture
- xwork.xml
- Configurations
- Interceptors
- Package
- Actions
- Pre Result Listeners
- Results
- Object Factory
- Conversion
- Validation
- Profiling
- Value Stack
- Annotations
- Reloading
- OGNL
- Localization

Integration

- Spring

- [Hibernate](#)

Help

- [FAQs](#)

Contributors Guide

Source

- [Building the Framework from Source](#)
- [How to Write Doc Comments for the Javadoc Tool](#) (Sun)
- [Precise Error Reporting](#)
- [Distribution](#)

Documentation

- [Documentation Colophon](#)
- [Documentation Style Guide](#)

Annotations

This page last changed on Jan 17, 2007 by phil.

In many places, applications can use use Java 5 annotations as an alternative to XML and Java properties configuration. This page serves as a reference for all annotations across the framework.

Interceptor Annotations

To use these annotations, you have to specify the AnnotationWorkflowInterceptor to your interceptor stack.

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

This page last changed on Dec 01, 2006 by rainerh.

After Annotation

Marks a action method that needs to be called after the main action method and the result was executed. Return value is ignored.

Usage

The After annotation can be applied at method level.

Parameters

Parameter	Required	Default	Notes
priority	no	10	Priority order of method execution

Examples

```
public class SampleAction extends ActionSupport {

 @After
 public void isValid() throws ValidationException {
   // validate model object, throw exception if failed
 }

 public String execute() {
   // perform action
   return SUCCESS;
 }
}
```

AnnotationWorkflowInterceptor Interceptor

Invokes any annotated methods on the action. Specifically, it supports the following annotations:

- @Before - will be invoked before the action method. If the returned value is not null, it is returned as the action result code
- @BeforeResult - will be invoked after the action method but before the result execution
- @After - will be invoked after the action method and result execution

There can be multiple methods marked with the same annotations, but the order of their execution is not guaranteed. However, the annotated methods on the superclass chain are guaranteed to be invoked before the annotated method in the current class in the case of a Before annotations and after, if the annotations is After.

Examples

```
public class BaseAnnotatedAction {
      protected String log = "";
<p/>
      @Before
      public String baseBefore() {
            log = log + "baseBefore-";
            return null;
      }
 }
<p/>
 public class AnnotatedAction extends BaseAnnotatedAction {
      @Before
      public String before() {
            log = log + "before";
            return null;
      }
<p/>
      public String execute() {
            log = log + "-execute";
            return Action.SUCCESS;
      }
<p/>
      @BeforeResult
      public void beforeResult() throws Exception {
            log = log +"-beforeResult";
      }
<p/>
      @After
      public void after() {
            log = log + "-after";
      }
 }
```

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

```
    <interceptor-stack name="annotatedStack">
            <interceptor-ref name="staticParams"/>
            <interceptor-ref name="params"/>
            <interceptor-ref name="conversionError"/>
            <interceptor-ref name="annotationWorkflow"/>
    </interceptor-stack>
```

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>
```

With the interceptor applied and the action executed on `AnnotatedAction` the log instance variable will contain `baseBefore-before-execute-beforeResult-after`.

This page last changed on Dec 01, 2006 by rainerh.

Before Annotation

Marks a action method that needs to be executed before the main action method.

Usage

The Before annotation can be applied at method level.

Parameters

Parameter	Required	Default	Notes
priority	no	10	Priority order of method execution

Examples

```
public class SampleAction extends ActionSupport {

 @Before
 public void isAuthorized() throws AuthenticationException {
   // authorize request, throw exception if failed
 }

 public String execute() {
    // perform secure action
    return SUCCESS;
 }
}
```

This page last changed on Dec 01, 2006 by rainerh.

BeforeResult Annotation

Marks a action method that needs to be executed before the result. Return value is ignored.

Usage

The BeforeResult annotation can be applied at method level.

Parameters

Parameter	Required	Default	Notes
priority	no	10	Priority order of method execution

Examples

```
public class SampleAction extends ActionSupport {

 @BeforeResult
 public void isValid() throws ValidationException {
   // validate model object, throw exception if failed
 }

 public String execute() {
    // perform action
    return SUCCESS;
 }
}
```

Conversion Annotation

A marker annotation for type conversions at Type level.

Usage

The Conversion annotation must be applied at Type level.

Parameters

Parameter	Required	Default	Description
conversion	no		used for Type Conversions applied at Type level.

Examples

```
@Conversion()
public class ConversionAction implements Action {
}
```

ConversionErrorFieldValidator Annotation

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.

Usage

The ConversionErrorFieldValidator annotation must be applied at method level.

Parameters

Parameter	Required	Default	Notes
message	yes		field error message
key	no		i18n key from language specific properties file.
fieldName	no		
shortCircuit	no	false	If this validator should be used as shortCircuit.
type	yes	ValidatorType.FIELD	Enum value from ValidatorType. Either FIELD or SIMPLE can be used here.

Examples

```
@ConversionErrorFieldValidator(message = "Default message", key = "i18n.key", shortCircuit =
true)
```

CreateIfNull Annotation

This page last changed on Dec 01, 2006 by rainerh.

Sets the CreateIfNull for type conversion.

Usage

The CreateIfNull annotation must be applied at field or method level.

Parameters

Parameter	Required	Default	Description
value	no	false	The CreateIfNull property value.

Examples

```
@CreateIfNull( value = true )
private List<User> users;
```

CustomValidator Annotation

This annotation can be used for custom validators. Use the ValidationParameter annotation to supply additional params.

Usage

The annotation must be applied at method or type level.

Parameters

Parameter	Required	Default	Notes
message	yes		field error message
key	no		i18n key from language specific properties file.
fieldName	no		
shortCircuit	no	false	If this validator should be used as shortCircuit.
type	yes	ValidatorType.FIELD	Enum value from ValidatorType. Either FIELD or SIMPLE can be used here.

Examples

```
@CustomValidator(type ="customValidatorName", fieldName = "myField")
```

Adding Parameters

Use the ValidationParameter annotation to add custom parameter values.

This page last changed on Dec 01, 2006 by rainerh.

ValidationParameter Annotation

The ValidationParameter annotation is used as a parameter for CustomValidators.

Usage

The annotation must embedded into CustomValidator annotations as a parameter.

Parameters

Parameter	Required	Default	Notes
name	yes		parameter name.
value	yes		parameter value.

Examples

```
@CustomValidator(
  type ="customValidatorName",
  fieldName = "myField",
  parameters = { @ValidationParameter( name = "paramName", value = "paramValue" ) }
)
```

DateRangeFieldValidator Annotation

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

Usage

The annotation must be applied at method level.

Parameters

Parameter	Required	Default	Notes
message	yes		field error message
key	no		i18n key from language specific properties file.
fieldName	no		
shortCircuit	no	false	If this validator should be used as shortCircuit.
type	yes	ValidatorType.FIELD	Enum value from ValidatorType. Either FIELD or SIMPLE can be used here.
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.

Examples

```
@DateRangeFieldValidator(message = "Default message", key = "i18n.key", shortCircuit = true,
min = "2005/01/01", max = "2005/12/31")
```

DoubleRangeFieldValidator Annotation

This validator checks that a double field has a value within a specified range. If neither min nor max is set, nothing will be done.

Usage

The annotation must be applied at method level.

Parameters

Parameter	Required	Default	Notes
message	yes		field error message
key	no		i18n key from language specific properties file.
fieldName	no		
shortCircuit	no	false	If this validator should be used as shortCircuit.
type	yes	ValidatorType.FIELD	Enum value from ValidatorType. Either FIELD or SIMPLE can be used here.
minInclusive	no		Double property. The inclusive minimum the number must be.
maxInclusive	no		Double property. The inclusive maximum number can be.
minExclusive	no		Double property. The exclusive minimum the number must be.
maxExclusive	no		Double property. The exclusive maximum number can be.

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

The values for min and max must be inserted as String values so that "0" can be handled as a possible value.

Examples

```
@DoubleRangeFieldValidator(message = "Default message", key = "i18n.key", shortCircuit = true,
minInclusive = "0.123", maxInclusive = "99.987")
```

Element Annotation

Sets the Element for type conversion.

Usage

The Element annotation must be applied at field or method level.

Parameters

Parameter	Required	Default	Description
value	no	java.lang.Object.class	The element property value.

Examples

```
// The key property for User objects within the users collection is the <code>userName</code>
attribute.
@Element( value = com.acme.User )
private Map<Long, User> userMap;

@Element( value = com.acme.User )
public List<User> userList;
```

EmailValidator Annotation

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

Usage

The annotation must be applied at method level.

Parameters

Parameter	Required	Default	Notes
message	yes		field error message
key	no		i18n key from language specific properties file.
fieldName	no		
shortCircuit	no	false	If this validator should be used as shortCircuit.
type	yes	ValidatorType.FIELD	Enum value from ValidatorType. Either FIELD or SIMPLE can be used here.

Examples

```
@EmailValidator(message = "Default message", key = "i18n.key", shortCircuit = true)
```

This page last changed on Dec 01, 2006 by rainerh.

ExpressionValidator Annotation

This non-field level validator validates a supplied regular expression.

Usage

The annotation must be applied at method level.

Parameters

Parameter	Required	Default	Notes
message	yes		field error message
key	no		i18n key from language specific properties file.
shortCircuit	no	false	If this validator should be used as shortCircuit.
expression	yes		An OGNL expression that returns a boolean value.

Examples

```
@ExpressionValidator(message = "Default message", key = "i18n.key", shortCircuit = true,
expression = "an OGNL expression" )
```

FieldExpressionValidator Annotation

This page last changed on Dec 01, 2006 by rainerh.

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

Usage

The annotation must be applied at method level.

Parameters

Parameter	Required	Default	Notes
message	yes		field error message
key	no		i18n key from language specific properties file.
fieldName	no		
shortCircuit	no	false	If this validator should be used as shortCircuit.
expression	yes		An OGNL expression that returns a boolean value.

Examples

```
@FieldExpressionValidator(message = "Default message", key = "i18n.key", shortCircuit = true,
expression = "an OGNL expression")
```

IntRangeFieldValidator Annotation

This page last changed on Dec 01, 2006 by rainerh.

This validator checks that a numeric field has a value within a specified range. If neither min nor max is set, nothing will be done.

Usage

The annotation must be applied at method level.

Parameters

Parameter	Required	Default	Notes
message	yes		field error message
key	no		i18n key from language specific properties file.
fieldName	no		
shortCircuit	no	false	If this validator should be used as shortCircuit.
type	yes	ValidatorType.FIELD	Enum value from ValidatorType. Either FIELD or SIMPLE can be used here.
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.

The values for min and max must be inserted as String values so that "0" can be handled as a possible value.

Examples

```
@IntRangeFieldValidator(message = "Default message", key = "i18n.key", shortCircuit = true, min
= "0", max = "42")
```

Key Annotation

Sets the Key for type conversion.

Usage

The Key annotation must be applied at field or method level.

Parameters

Parameter	Required	Default	Description
value	no	java.lang.Object.class	The key property value.

Examples

```
// The key property for User objects within the users collection is the <code>userName</code>
attribute.
@Key( value = java.lang.Long.class )
private Map<Long, User> userMap;
```

KeyProperty Annotation

Sets the KeyProperty for type conversion.

Usage

The KeyProperty annotation must be applied at field or method level. This annotation should be used with Generic types, if the key property of the key element needs to be specified.

Parameters

Parameter	Required	Default	Description
value	no	id	The key property value.

Examples

```
// The key property for User objects within the users collection is the <code>userName</code>
attribute.
@KeyProperty( value = "userName" )
protected List<User> users = null;
```

RegexFieldValidator Annotation

Validates a string field using a regular expression.

Usage

The annotation must be applied at method level.

Parameters

Parameter	Required	Default	Notes
message	yes		field error message
key	no		i18n key from language specific properties file.
fieldName	no		
shortCircuit	no	false	If this validator should be used as shortCircuit.
type	yes	ValidatorType.FIELD	Enum value from ValidatorType. Either FIELD or SIMPLE can be used here.
expression	yes		The regex to validate the field value against.

Examples

```
@RegexFieldValidator( key = "regex.field", expression = "yourregexp")
```

RequiredFieldValidator Annotation

This validator checks that a field is non-null.

Usage

The annotation must be applied at method level.

Parameters

Parameter	Required	Default	Notes
message	yes		field error message
key	no		i18n key from language specific properties file.
fieldName	no		
shortCircuit	no	false	If this validator should be used as shortCircuit.
type	yes	ValidatorType.FIELD	Enum value from ValidatorType. Either FIELD or SIMPLE can be used here.

Examples

```
@RequiredFieldValidator(message = "Default message", key = "i18n.key", shortCircuit = true)
```

RequiredStringValidator Annotation

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

Usage

The annotation must be applied at method level.

Parameters

Parameter	Required	Default	Notes
message	yes		field error message
key	no		i18n key from language specific properties file.
fieldName	no		
shortCircuit	no	false	If this validator should be used as shortCircuit.
type	yes	ValidatorType.FIELD	Enum value from ValidatorType. Either FIELD or SIMPLE can be used here.
trim	no	true	Boolean property. Determines whether the String is trimmed before performing the length check.

Examples

```
@RequiredStringValidator(message = "Default message", key = "i18n.key", shortCircuit = true,
trim = true)
```

StringLengthFieldValidator Annotation

This validator checks that a String field is of the right length. It assumes that the field is a String. If neither minLength nor maxLength is set, nothing will be done.

Usage

The annotation must be applied at method level.

Parameters

Parameter	Required	Default	Notes
message	yes		field error message
key	no		i18n key from language specific properties file.
fieldName	no		
shortCircuit	no	false	If this validator should be used as shortCircuit.
type	yes	ValidatorType.FIELD	Enum value from ValidatorType. Either FIELD or SIMPLE can be used here.
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.

Examples

```
@StringLengthFieldValidator(message = "Default message", key = "i18n.key", shortCircuit = true,
trim = true, minLength = "5",  maxLength = "12")
```

StringRegexValidator Annotation

An error occurred:
http://svn.opensymphony.com/svn/xwork/trunk/src/java/com/opensymphony/xwork2/validator/annotations/StringReg
The system administrator has been notified.

Usage

An error occurred:
http://svn.opensymphony.com/svn/xwork/trunk/src/java/com/opensymphony/xwork2/validator/annotations/StringReg
The system administrator has been notified.

Parameters

An error occurred:
http://svn.opensymphony.com/svn/xwork/trunk/src/java/com/opensymphony/xwork2/validator/annotations/StringReg
The system administrator has been notified.

Examples

An error occurred:
http://svn.opensymphony.com/svn/xwork/trunk/src/java/com/opensymphony/xwork2/validator/annotations/StringReg
The system administrator has been notified.

TypeConversion Annotation

This annotation is used for class and application wide conversion rules.

Class wide conversion:
The conversion rules will be assembled in a file called `XXXAction-conversion.properties` within the same package as the related action class. Set type to: `type = ConversionType.CLASS`

Allication wide conversion:
The conversion rules will be assembled within the `xwork-conversion.properties` file within the classpath root. Set type to: `type = ConversionType.APPLICATION`

Usage

The TypeConversion annotation can be applied at property and method level.

Parameters

Parameter	Required	Default	Description
key	no	The annotated property/key name	The optional property name mostly used within TYPE level annotations.
type	no	ConversionType.CLASS	Enum value of ConversionType. Determines whether the conversion should be applied at application or class level.
rule	no	ConversionRule.PROPERTY	Enum value of ConversionRule. The ConversionRule can be a property, a Collection or a Map.
converter	either this or value		The class name of the TypeConverter to be used as converter.
value	either converter or this		The value to set for ConversionRule.KEY_PROPERTY.

Examples

```
@Conversion()
public class ConversionAction implements Action {

  private String convertInt;
```

```
    private String convertDouble;
    private List users = null;

    private HashMap keyValues = null;

    @TypeConversion(type = ConversionType.APPLICATION, converter =
"com.opensymphony.xwork2.util.XWorkBasicConverter")
    public void setConvertInt( String convertInt ) {
        this.convertInt = convertInt;
    }

    @TypeConversion(converter = "com.opensymphony.xwork2.util.XWorkBasicConverter")
    public void setConvertDouble( String convertDouble ) {
        this.convertDouble = convertDouble;
    }

    @TypeConversion(rule = ConversionRule.COLLECTION, converter = "java.util.String")
    public void setUsers( List users ) {
        this.users = users;
    }

    @TypeConversion(rule = ConversionRule.MAP, converter = "java.math.BigInteger")
    public void setKeyValues( HashMap keyValues ) {
        this.keyValues = keyValues;
    }

    @TypeConversion(type = ConversionType.APPLICATION, property = "java.util.Date", converter =
"com.opensymphony.xwork2.util.XWorkBasicConverter")
    public String execute() throws Exception {
        return SUCCESS;
    }
}
```

UrlValidator Annotation

This validator checks that a field is a valid URL.

Usage

The annotation must be applied at method level.

Parameters

Parameter	Required	Default	Notes
message	yes		field error message
key	no		i18n key from language specific properties file.
fieldName	no		
shortCircuit	no	false	If this validator should be used as shortCircuit.
type	yes	ValidatorType.FIELD	Enum value from ValidatorType. Either FIELD or SIMPLE can be used here.

Examples

```
@UrlValidator(message = "Default message", key = "i18n.key", shortCircuit = true)
```

Validation Annotation

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

Usage

The Validation annotation must be applied at Type level.

Parameters

Parameter	Required	Default	Notes
validations	yes		

Examples

An Annotated Interface

- Mark the interface with @Validation()
- Apply standard or custom annoations at method level

```
@Validation()
public interface AnnotationDataAware {

    void setBarObj(Bar b);

    Bar getBarObj();

    @RequiredFieldValidator(message = "You must enter a value for data.")
    @RequiredStringValidator(message = "You must enter a value for data.")
    void setData(String data);

    String getData();
}
```

An Annotated Class

```
@Validation()
public class SimpleAnnotationAction extends ActionSupport {

    @RequiredFieldValidator(type = ValidatorType.FIELD, message = "You must enter a value for
bar.")
    @IntRangeFieldValidator(type = ValidatorType.FIELD, min = "6", max = "10", message = "bar
must be between ${min} and ${max}, current value is ${bar}.")
    public void setBar(int bar) {
        this.bar = bar;
    }

    public int getBar() {
        return bar;
```

```
        }

    @Validations(
            requiredFields =
                    {@RequiredFieldValidator(type = ValidatorType.SIMPLE, fieldName =
"customfield", message = "You must enter a value for field.")},
            requiredStrings =
                    {@RequiredStringValidator(type = ValidatorType.SIMPLE, fieldName =
"stringisrequired", message = "You must enter a value for string.")},
            emails =
                    { @EmailValidator(type = ValidatorType.SIMPLE, fieldName = "emailaddress",
message = "You must enter a value for email.")},
            urls =
                    { @UrlValidator(type = ValidatorType.SIMPLE, fieldName = "hreflocation",
message = "You must enter a value for email.")},
            stringLengthFields =
                    {@StringLengthFieldValidator(type = ValidatorType.SIMPLE, trim = true,
minLength="10" , maxLength = "12", fieldName = "needstringlength", message = "You must enter a
stringlength.")},
            intRangeFields =
                    { @IntRangeFieldValidator(type = ValidatorType.SIMPLE, fieldName =
"intfield", min = "6", max = "10", message = "bar must be between ${min} and ${max}, current
value is ${bar}.")},
            dateRangeFields =
                    {@DateRangeFieldValidator(type = ValidatorType.SIMPLE, fieldName =
"datefield", min = "-1", max = "99", message = "bar must be between ${min} and ${max}, current
value is ${bar}.")},
            expressions = {
                @ExpressionValidator(expression = "foo > 1", message = "Foo must be greater
than Bar 1. Foo = ${foo}, Bar = ${bar}."),
                @ExpressionValidator(expression = "foo > 2", message = "Foo must be greater
than Bar 2. Foo = ${foo}, Bar = ${bar}."),
                @ExpressionValidator(expression = "foo > 3", message = "Foo must be greater
than Bar 3. Foo = ${foo}, Bar = ${bar}."),
                @ExpressionValidator(expression = "foo > 4", message = "Foo must be greater
than Bar 4. Foo = ${foo}, Bar = ${bar}."),
                @ExpressionValidator(expression = "foo > 5", message = "Foo must be greater
than Bar 5. Foo = ${foo}, Bar = ${bar}.")
    }
    )
    public String execute() throws Exception {
        return SUCCESS;
    }
}
```

Validations Annotation

This page last changed on Dec 01, 2006 by rainerh.

If you want to use several annotations of the same type, these annotations must be nested within the @Validations() annotation.

Usage

Used at METHOD level.

Parameters

Parameter	Required	Notes
requiredFields	no	Add list of RequiredFieldValidators
customValidators	no	Add list of CustomValidators
conversionErrorFields	no	Add list of ConversionErrorFieldValidators
dateRangeFields	no	Add list of DateRangeFieldValidators
emails	no	Add list of EmailValidators
fieldExpressions	no	Add list of FieldExpressionValidators
intRangeFields	no	Add list of IntRangeFieldValidators
requiredStrings	no	Add list of RequiredStringValidators
stringLengthFields	no	Add list of StringLengthFieldValidators
urls	no	Add list of UrlValidators
visitorFields	no	Add list of VisitorFieldValidators
regexFields	no	Add list of RegexFieldValidator
expressions	no	Add list of ExpressionValidator

Examples

```
@Validations(
        requiredFields =
                {@RequiredFieldValidator(type = ValidatorType.SIMPLE, fieldName =
"customfield", message = "You must enter a value for field.")},
```

```
        requiredStrings =
                {@RequiredStringValidator(type = ValidatorType.SIMPLE, fieldName =
"stringisrequired", message = "You must enter a value for string.")},
        emails =
                { @EmailValidator(type = ValidatorType.SIMPLE, fieldName = "emailaddress",
message = "You must enter a value for email.")},
        urls =
                { @UrlValidator(type = ValidatorType.SIMPLE, fieldName = "hreflocation",
message = "You must enter a value for email.")},
        stringLengthFields =
                {@StringLengthFieldValidator(type = ValidatorType.SIMPLE, trim = true,
minLength="10" , maxLength = "12", fieldName = "needstringlength", message = "You must enter a
stringlength.")},
        intRangeFields =
                { @IntRangeFieldValidator(type = ValidatorType.SIMPLE, fieldName =
"intfield", min = "6", max = "10", message = "bar must be between ${min} and ${max}, current
value is ${bar}.")},
        dateRangeFields =
                {@DateRangeFieldValidator(type = ValidatorType.SIMPLE, fieldName =
"datefield", min = "-1", max = "99", message = "bar must be between ${min} and ${max}, current
value is ${bar}.")},
        expressions = {
                @ExpressionValidator(expression = "foo > 1", message = "Foo must be greater than
Bar 1. Foo = ${foo}, Bar = ${bar}."),
                @ExpressionValidator(expression = "foo > 2", message = "Foo must be greater than
Bar 2. Foo = ${foo}, Bar = ${bar}."),
                @ExpressionValidator(expression = "foo > 3", message = "Foo must be greater than
Bar 3. Foo = ${foo}, Bar = ${bar}."),
                @ExpressionValidator(expression = "foo > 4", message = "Foo must be greater than
Bar 4. Foo = ${foo}, Bar = ${bar}."),
                @ExpressionValidator(expression = "foo > 5", message = "Foo must be greater than
Bar 5. Foo = ${foo}, Bar = ${bar}.")
  }
  )
  public String execute() throws Exception {
      return SUCCESS;
  }
```

VisitorFieldValidator Annotation

This page last changed on Dec 01, 2006 by rainerh.

The validator allows you to forward validator to object properties of your action using the objects own validator 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 validator messages added by the validations for the Object message.

Usage

The annotation must be applied at method level.

Parameters

Parameter	Required	Default	Notes
message	yes		field error message
key	no		i18n key from language specific properties file.
fieldName	no		
shortCircuit	no	false	If this validator should be used as shortCircuit.
context	no	action alias	Determines the context to use for validating the Object property. If not defined, the context of the Action validation is propogated 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 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 *appendPrefix* is true, then the field error will be stored under the field "bean.name". If *appendPrefix* is false, then the field error will be stored under the field "name". ⚠ If you are using the VisitorFieldValidator to validate the model from a ModelDriven Action, you should set *appendPrefix* to false unless you are using "model.name" to reference the properties on your model.

Examples

```
@VisitorFieldValidator(message = "Default message", key = "i18n.key", shortCircuit = true,
context = "action alias", appendPrefix = true)
```

Building XWork

This page last changed on Dec 01, 2006 by rainerh.

Why Build?

Most developers will never need to build the framework from source. The distribution package contains everything a developer needs to get started and become productive with the framework. For more on how to go to work with the distributed binaries right away, see Ready, Set, Go\!. However, there are situations when someone will want to build the framework from scratch. You may want to try out new tweaks and patches, or you might want to try writing your own tweak or patch.

Getting the Sources

The source code for the framework is available as distribution you can download directly or from the source code repository.

Distribution

The distributions of the framework contain all sources, as well as all needed libraries for building JARs and running. Distributions can be downloaded from here.

After downloading simply unzip it using:

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

Respository (SVN)

Use Subversion to checkout the source code.

```
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

> ⚠ Building XWork2 requires JDK 1.5 (Tiger)

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
```

Building XWork2 for Java 1.4

To build a Java 1.4 compliant version of the jars, we use retroweaver to translate the classes.

> **ⓘ You need to have a Java 1.4 VM installed**
>
> On Windows/Linux/Solaris point the java14.jar property to tools.jar

```
mvn clean install -Pj4 -Djava14.jar=$JAVA14_HOME/../Classes/classes.jar
```

Colophon

The XWork2 documentation is maintained online and posted to the OpenSymphony web site on a regular basis. A snapshot of the documentation is bundled with each distribution, so that people are able to refer to the documentation for that distribution.

> ⚠️ The online version represents the latest development version ("nightly build") and may document features **not** available in your release. See the documentation bundled with each distribution for the best information about the features available in that distribution.

The HTML version of the documentation is maintained using the AutoExport plugin. (JAR annexed.)

The documentation "single sources" code segments using the Snippet macro. (JAR annexed.) For more about using snippets, see the Documentation Style Guide.

Configuring XWork in xwork.xml

Overview

All action configuration is done from within xwork.xml.
In this section we discuss the various elements that make up the action configuration, such as actions, interceptors, results, and package.

1. Package Configuration
2. Namespace Configuration
3. Include configuration
4. Action configuration
5. Result Configuration
6. Interceptor Configuration

Action configuration

This page last changed on Dec 01, 2006 by rainerh.

Actions are the basic "unit-of-work" in XWork, they define, well, actions. An action will usually be a request, (and usually a button click, or form submit). The main action element has two parts, the friendly name (referenced in the URL, i.e. saveForm.action) and the corresponding "handler" class.

```
<action name="formTest" class="com.opensymphony.xwork.example.SampleAction"
method="processSample">
```

The optional "**method**" parameter tells XWork which method to call based upon this action. If you leave the method parameter blank, XWork will call the method **execute()** by default. If there is no execute() method and no method specified in the xml file, XWork will throw an exception.

Also, you can tell XWork to invoke "**doSomething**" method in your action by using the pattern "**actionName!something**" in your form. For example, "**sampleTest!save.action**" will invoke the method "**save**" in SampleAction class. The method must be public, take no arguments and also returns a String which indicate the name of the result to be executed:

```
public String save() throws Exception
  {
      ...
      return SUCCESS;
  }
```

All the configuration for "**actionName**" will be used for "actionName!something" (interceptors, result types, etc...)

Action Support

Action class attribute could be left out such as following

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

In this case, the class will default to com.opensymphony.xwork.ActionSupport which have an execute() method that returns SUCCESS by default.

Default Action Reference

You are also able to specify a default action to be executed when no other action is matched in the xwork.xml. This feature is provided mainly to eliminate the need to create an action class and/or element for very simple or similar results. The default action name can be set inside the package element like

below:

```
<package name="myPackage" ....>

...

<default-action-ref name="simpleViewResultAction">

<!--
An example of a default action that is just a simple class
that has 3 fields: successUrl, errorUrl, and inputUrl.  This action
parses the request url to set the result values.  In the normal case
it just renders velocity results of the same name as the requested url.
-->
<action name="simpleViewResultAction" class="SimpleViewResultAction">
        <result type="velocity">${successUrl}</result>
        <result name="error" type="velocity">${errorUrl}</result>
        <result name="input" type="velocity">${inputUrl}</result>
</action>

...

</package>
```

⚠ **Caution**

This feature should be configured such that there is only one default action per namespace.
Therefore if you have multiple packages declaring a default action with the same namespace, it is
not guaranteed which action will be the default.

⚠ **Note**

Note that the name attribute is left out for the first result, as XWork will default to "success" if it is
left out.

In this case any request to action not defined in this package will automatically trigger action with alias
"simpleViewResultAction" to be executed.

Include configuration

Description

To make it easy to manage large scale development (lots of actions + configuration), XWork allows you to include other configuration files from xwork.xml :

```
<xwork>
    <include file="xwork-default.xml"/>
    <include file="xwork-extension.xml"/>
    <include file="xwork-mail.xml"/>
    <include file="xwork-xmlrpc.xml"/>
    ....
</xwork>
```

The included files must be the same format as xwork.xml (with the doctype and everything) and be placed on classpath.

This page last changed on Dec 01, 2006 by rainerh.

Description

Interceptors allow you to define code to be executed before and/or after the execution of an action. Interceptors can be a powerful tool when writing web applications. Some of the most common implementations of an Interceptor might be:

- Security Checking (ensuring the user is logged in)
- Trace Logging (logging every action)
- Bottleneck Checking (start a timer before and after every action, to check bottlenecks in your application)

You can also chain Interceptors together to create an interceptor **stack**. If you wanted to do a login check, security check, and logging all before an Action call, this could easily be done with an interceptor package.

Intercepters must first be defined (to give name them) and can be chained together as a stack:

```
<interceptors>
  <interceptor name="security" class="com.mycompany.security.SecurityInterceptor"/>
  <interceptor-stack name="defaultComponentStack">
    <interceptor-ref name="component"/>
    <interceptor-ref name="defaultStack"/>
  </interceptor-stack>
</interceptors>
```

To use them in your actions:

```
<action name="VelocityCounter" class="com.opensymphony.xwork.example.counter.SimpleCounter">
    <result name="success">...</result>
    <interceptor-ref name="defaultComponentStack"/>
</action>
```

NOTE: Reference name can be either the name of the interceptor or the name of a stack

For more details, see Interceptors reference.

Namespace Configuration

This page last changed on Dec 01, 2006 by rainerh.

Namespaces

The namespace attribute allows you to segregate action configurations into namespaces, so that you may use the same action alias in more than one namespace with different classes, parameters, etc.

Default Namespace

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. It is also intended that the namespace functionality can be used for security, thus allowing the use of J2EE declarative security on paths to be easily implemented and maintained.

Root Namesapce

Root namespace, which is "/" is also allowed in WebWork. It will be the namespace when a request directly under the context path is received. As with other namespace, it will fall back to the default namespace if no such action alias is found in it.

Namespace example

```
<package name="default">
    <action name="foo" class="mypackage.simpleAction">
        <result name="success" type="dispatcher">greeting.jsp</result>
    </action>
    <action name="bar" class="mypackage.simpleAction">
        <result name="success" type="dispatcher">bar1.jsp</result>
    </action>
</package>

<package name="mypackage1" namespace="/">
    <action name="moo" class="mypackage.simpleActtion">
        <result name="success" type="dispatcher">moo.jsp</result>
    </action>
</package>

<package name="mypackage2" namespace="/barspace">
    <action name="bar" class="mypackage.simpleAction">
        <result name="success" type="dispatcher">bar2.jsp</result>
    </action>
</package>
```

Explanation

Document generated by Confluence on Apr 01, 2008 04:35   Page 50

If a request for /barspace/bar.action is made, '/barspace' namespace is searched and if it is found the bar action is executed, else it will fall back to the default namespace. In this example bar alias do exists in the '/barspace' namespace, so it will get executed and if success, the request will be forwarded to bar2.jsp.

Note: If a request is made to /barspace/foo.action, the action foo will be searched for in a namespace of /barspace. If the action is not found, the action will then be searched for in the default namespace. Unless specified, the default namespace will be "". In our example above, their is no action foo in the namespace /barspace, therefore the default will be searched and /foo.action will be executed.

If a request for /moo.action is made, the root namespace ('/') is searched for 'moo' action alias, if it is not found it will fall back to trying to find it in the default namespace. In this example, moo action alias does exists and hence will be executed. Upon sucess, the request will get forwarded to bar2.jsp.

Note: If a request is made to '/foo.action', '/' namespace will be searched and if it is found it will be execueted, else an attempt to try to find it in the default namespace will occurred. In this example foo action alias does not exist in the '/' namespace, hence it will falled back to the default namespace and execute the foo alias there.

Note: Namespace is only one level. For example if the url '/barspace/myspace/bar.action' is requested, Webwork will try to search for namespace '/barspace/myspace', which does not exist in this case, and will fall back to the default namespace '' and tried the search for action with 'bar' alias. As a result the bar action in the default will be used.

This page last changed on Dec 01, 2006 by rainerh.

Overview

Packages are a way to group Actions, Results, Result Types, Interceptors and Stacks into a logical unit that shares a common configuration. Packages are similiar to objects in that they can be extended and have individual parts overridden by "sub" packages.

Packages

The package element has one required attribute, "name", which acts as the key for later reference to this package. The "extends" attribute is optional and allows one package to inherit the configuration of one or more previous packages including all interceptor, interceptor-stack, and action configurations. Note that the configuration file is processed sequentially down the document, so the package referenced by an "extends" should be defined above the package which extends it. The "abstract" optional attribute allows you to make a package abstract, which will allow you to extend from it without the action configurations defined in the abstract package actually being available at runtime.

Attribute	Required	Description
name	**yes**	key to for other packages to reference
extends	no	inherits package behavior of the package it extends
namespace	no	see Namespace Configuration
abstract	no	declares package to be abstract (no action configurations required in package)

Result Configuration

Description

Results are string constants that Actions return to indicate the status of an Action execution. A standard set of Results are defined by default: error, input, login, none and success. Developers are, of course, free to create their own Results to indicate more application specific cases. Results are mapped to defined Result Types using a name-value pair structure.

- Global results
- Default results

Result tags

Result tags tell XWork what to do next after the action has been called. There are a standard set of result codes built-in to XWork, (in the Action interface) they include:

```
String SUCCESS = "success";
String NONE    = "none";
String ERROR   = "error";
String INPUT   = "input";
String LOGIN   = "login";
```

You can extend these as you see fit. Most of the time you will have either **SUCCESS** or **ERROR**, with **SUCCESS** moving on to the next page in your application;

```
<result name="success" type="dispatcher">
    <param name="location">/thank_you.jsp</param>
</result>
```

...and **ERROR** moving on to an error page, or the preceding page;

```
<result name="error" type="dispatcher">
    <param name="location">/error.jsp</param>
</result>
```

Results are specified in a xwork xml config file (xwork.xml) nested inside <action>. If the `location` param is the only param being specified in the result tag, you can simplify it as follows:

```
<action name="bar" class="myPackage.barAction">
  <result name="success" type="dispatcher">
    <param name="location">foo.jsp</param>
  </result>
</action>
```

or simplified

```
<action name="bar" class="myPackage.barAction">
  <result name="success" type="dispatcher">foo.jsp</result>
</action>
```

or even simplified further

```
<action name="bar" class="myPackage.barAction">
   <result>foo.jsp</result>
</action>
```

> ⚠ **Default Action Class**
>
> If the class attribute is not specified in the action tag, it will default to XWork's ActionSupport.

> ⚠ **Default Location Parameter**
>
> If no param tag eg. <param name="location"> ,,, </param> is given as child of the <result ..> tag, XWork will assume the text enclosed within the <result> </result> tags to be the location.

> ⚠ **Default Result Type**
>
> If no type attribute is specified in the <result ...> tag, XWork assume the type to be dispatcher. (Analogus to Servlet's Specs. SerlvetDispatcher's forward).

Creating a distribution

This page last changed on Dec 02, 2006 by rainerh.

Update the docs

Export the wiki docs from Confluence http://wiki.opensymphony.com/display/XW

1	Add a press release page for the new release
2	Click on browse Space # Advanced Tab # Export Space
3	Check OFF everything that isn't under XWork2, except for the path to Documentation (ie: XWork)
4	Uncheck include comments
5	Open to disable the refresh buttons for the export of wiki pages http://wiki.opensymphony.com/plugins/snippet/toggle.act
6	Export both HTML and PDF (will take a while...)
7	Open to enable the refresh buttons for the SNIPPET macro http://wiki.opensymphony.com/plugins/snippet/toggle.act
8	Delete everything in docs/wikidocs, remove from SVN and commit
9	Add everything back in, when the HTML stuff is one
10	Add the PDF to docs/wikidocs/docs.pdf
11	Commit changes to SVN

Build the release bundles with maven

1	Update the POMs to remove "-SNAPSHOT" from the version
2	Commit the POM changes
3	Tag the release by making a SVN copy of the head or designated revision

	<pre>svn copy -r ######
	  https://svn.opensymphony.com/svn/xwork/trunk/
	  https://svn.opensymphony.com/svn/xwork/tags/xwork_#_#_#
	  -m "Tag r###### as XWork #.#.#"</pre>

4	Assemble the release

| | ```
mvn clean install site assembly:assembly
-Pj4
-Djava14.jar=/System/Library/Frameworks/JavaVM.framework/
``` |
5	Upload the release bundles to https://xwork.dev.java.net/servlets/ProjectDocumentList
	Create a new folder for the release as Version #.#.#
	Upload the file bundles and add file comments
6	Deploy the artifacts to our Maven repository
	ℹ️ You need shell access for this step!
	⛔ **Warning** Don't forget to upload the pom.xml as well!
	Prune any obsolete snapshots from `opensymphony.com://opt/repository/ibiblio.org/opensymph`
7	Update the POMs to next version number and add the "-SNAPSHOT" suffix
8	Commit the POM changes
9	Deploy the new snapshot
| | ```
mvn -N install
``` |
10	Add the next version to our issue tracker for scheduling new features and fixes
11	Update docs/meta.xml with the newly added download bundles
12	Clear the project cache so that he latest meta.xml information is used `www.opensymphony.com/clearProjectCache.jsp`
13	Verify the download link
14	Announce the release http://blogs.opensymphony.com

Documentation Style Guide

It's well-known that a consistent user interface is easier to use. Consistency helps users focus on the task rather than the user interface. Likewise, a consistent documentation style helps users focus on the information, rather than the formatting.

A related goal is to design the documentation so that it is easy to maintain, so that it tends to remain internally consistent with the framework itself.

Do it now. Do it once. Do it well.

Overall, there are three goals for the documentation system.

- Say it all
- Say it once
- Say it well

First, we want the documentation to be both complete and concise. This is job one! The documentation should also be a quick but practical introduction to the framework, so newcomers can get started as easily as possible. To keep people coming back, the documenation should also be a repository of the tips and tricks we use in our own applications, so that people can find it here instead of asking over and over again on the list.

Second, the documentation should be easy to maintain. Ideally, we should cover the detail of each topic once, and draw as much detail from the source code and examples as possible (using the snippet macro).

Third, the documentation should be text-book quality; if not in the first draft, then in the next. Don't hesitate to hack in a new page. Better that we have the page than we don't. (See Job One!) But, as time allows, we should try to make each page the best that it can be. A great many people access the documentation, and it's worth the effort to make the "documentation experience" productive and enjoyable.

Capitalization of common terms

- Java
- Javadoc
- HTML
- XML

General Punctuation and Grammar

Good online resources for punctuation, grammar, and text style include

- [Punctuation Made Simple](#)
- [Associated Press Style Guide Essentials](#)
- [Wikipedia Manual of Style](#)

In print, two excellent (and inexpensive!) resources are

- [The Elements of Style](#)
- [Associated Press Stylebook](#)

Also excellent, but more expensive:

- [Chicago Manual of Style](#)

Quick Tips

- Use as few words as possible. Instead of "but there are some quirks about it" try "but there are quirks".
- If a list of items includes both a term and an explanation, consider using a table instead of bullets.
- Avoid using "This" by itself. Instead of "This lets us" try "This strategy lets us".
  - Ask yourself: "This what?"
- References to other wiki pages can be unqualified. Instead of "See WW:Documentation Style Guide page", try "See WW:Documentation Style Guide."

Don't be smurfy!

A lot of API members use the term "action". We have

- action extensions on pages,
- action attributes in forms,
- action elements in configuration files, and
- Action Java classes, some of which may implement the
- Action interface.

Here are some terms that can be used to help clarify which action is which.

- Use "the framework" or "XWork" to refer to the codebase as a whole, including any frameworks we use internally, like OGNL.
- Use "Action class" or "action handler" to refer to the Java class incorporated by the action element.
- Use "action mapping" to refer to the object created by the action element.

Page Save Comment

Try to include a brief description of a change when saving a page. The comments are included in the page's history. The comments are also included on the daily change report. In a group environment, it's important to help each other follow along.

Parent Pages

Use the Parent Page feature to create a hierarchy of pages. The parent pages are reflected in the "bread crumb" menu. If propertly used, parent pages can help browsers "visualize" the documenation as an outline.

The root of the documentation is the "Home" page, which is also the "Welcome" page. The documnentation is ordered into three main areas: Tutorials, FAQs, and Guides. Each area has a contents page, whose parent is Home. Other pages within each section can also serve as parents, to help organize the documentation into a coherent outline.

Labels

Pages can be cross-indexed with the Label feature. Labels are not be used much yet, except for internal authoring.

FIXME	A page that mentions a problem in the distribution that we intend to fix. Review these pages before tagging a distribution to see if the issue has been resolved.
TODO	A page that is incomplete. Try to complete these pages before tagging a distribution

Shortcuts Links

The Shortcut Link feature should be used for any external reference that may be used elsewhere. Shortcuts being used include

jira	A ticket in our issue tracker
primer	A bookmark in our Key Technologies Primer
xwork/api	A class in the XWork Javadocs
java/api	A class in the Java Javadocs
wiki/help	A topic the Confluence help system

Please add new shortcuts as needed.

About Headings

This section refers to: [Notation Guide >> Headings|section=headings@wiki].

About h1

Don't use `h1.` at the top of each page. The page title serves as the "top level header". This is not as obvious online, but it is very apparent when the documentation is exported to HTML or PDF.

Try to start each page with some introductory text, to separate the page title from the rest of content.

Likewise, try to have some content between all page headings. Avoid placing headings one after the other.

Document sections

Headings can help you divide your document in sections, subsections, sub-subsections and so forth.

Advantages

Your document becomes more organized.

Disadvantages

Too many headings can fragment the text.

> ⊖ **Here we go again!**
>
> This segment an example of overusing headings. This whole "Headings" section has so few paragraphs that it really should have been written in just one section. The "advantages" and "disadvantages" would be just as easy to render as a table.

Headings capitalization

Try to use intial capitals for `h1` and `h2` headers.

For `h3` and smaller headings, try to capitalize only the first word, and any proper nouns.

By using different capilazation sytles, we emphasize the importance of bigger headings.

Avoid skipping headers

The headers form an outline for the page. When writing term papers, it is not a good practice to skip outline levels. When writing hypertext, it is not a good practice to skip heading levels either. Try not to skip from a `h2` to a `h4`.

> ✔ **Too many headings?**
>
> If you find yourself writting too many `h2` headings in a single page, consider breaking the page into child pages.

More on Text Effects

⭐ This section refers to: [Notation Guide >> Text Effects](#).

Text effects like **strong**, *emphasis*, and <u>inserted</u> can be used in the usual way to denote important parts of a sentence.

`Monospaced` should be used to files, tags, and methods, like `xwork.xml`, `<xmltag />`, and `execute`. Class and Interface names may be left in normal face, like Action and Interceptor.

A panel should be preferred to a block quote.

The color fonts should be avoided or used only with great care. Some people have difficulty seeing some colors, and the colors may not be apparent if the page is printed.

Text Breaks

⭐ This section refers to: [Notation Guide >> Text Breaks](#).

Text breaks should not be used to format blocks on the screen. If there is an issue with the way paragraphs or headings are being rendered, we should customize the stylesheet.

Lists

⭐ This section refers to: [Notation Guide >> Lists](#).

Unordered lists should be created only with the * (star) notation.

Ordered list should be used when numbering the items is important. Otherwise, prefer unordered lists.

- This is an unordered list in star notation;
- Items can have sub-items
  - That can have sub-items
    - That can have sub-items ...
      - What is the limit?
- Mixing ordered and unordered lists is possible:
  1. One;
  2. Two;
  3. Three.

Images

⭐ This section refers to: [Notation Guide >> Images](#) and [Notation Guide >> Misc](#).

Avoid using external images for bullets or icons. Prefer the equivalents provided with Confluence.

Images can be inclued by URL or annexing the binary to the page. Prefer annexing when possible, since URLs are subject to change.

Always observe copyright issues. Do not annex images unless it an orginal or public domain work, or the author has donated the image to the foundation.

Example: Cannot resolve external resource into attachment.

Icons

Use 🛈, ⊗, ⚠, and ✅ to bullet important one-liners. Use 💡 to highlight cross references.

Used carefully, icons can make the content easier to read and understand.

However, if icons are overused, they lose impact (and can make a page look like a ransom note).

Casual icons like 🙂 and 👉 should be used with care or avoided.

Tables

⭐ This section refers to: [Notation Guide >> Tables](#).

Prefer lists for single-value entries. Prefer tables for lists with multiple columns.

Tables are very useful when lists just don't do it. Meaning: don't write a table when a list suffices. Tables are more organized, because you can align the text in columns. Since the markup text for tables in Confluence is not easy to read, complex and big tables can be hard to maintain.

File	Optional	Location (relative to webapp)	Purpose
web.xml	no	/WEB-INF/	Web deployment descriptor to include all necessary WebWork components
xwork.xml	no	/WEB-INF/classes/	Main configuration, contains result/view types, action mappings, interceptors, and so forth

Advanced Formatting

⭐ This section refers to: [Notation Guide >> Advanced Formatting](#).

Panels should be used as needed. Try to select the right panel for the content.

Try to give all panels and {code} blocks meaningful titles. People scan the pages looking for likely tips and examples.

Avoid generic titles like "Warning" or "Example". Style the headings like they were h3. or smaller.

When a panel contains a file or a class, the panel title should refer to the filename or classname.

Try to specify the language for {code} blocks.

```
/** Hello World class. */
public class HelloWorld {
  /** Main method. */
  public static void main(String[] args) {
    System.out.println("Hello, World!");
  }
}
```

✅ Try to use [snippets](#) for code blocks whenever possible!

Avoid tabs in code blocks, use two spaces instead. Long lines should be formatted to fit in a 800x600 resolution screen, without resorting to horizontal scrolling.

A typical example of `noformat` would be the command line statements to compile and run the code above.

Either the code or noformat block can be used to represent command line windows. The terminal notation ({$}}) should be used to represent a system prompt.

```
$ javac HelloWorld.java

$ java HelloWorld
Hello, World!
```

Change Happens

Anyone who has worked with databases knows the value of normalizing the schema. Ideally, we want to store each fact exactly once, and then use query system to retrieve that fact whereever it is needed. If we store a fact once, we only need to update it once, and we avoid inconsistencies in our data set.

To the extent possible, we want to "normalize" our technical documentation. Like a database, all technical documentation is subject to change. When change happens, we want the documentation to be as easy to update as possible. One way to do that is to try and minimize redudency (without sacrificing ease of use).

Single sourcing with snippets

The "holy grail" of technical documentation is [single sourcing](). One way we try to single-source documentation is to pull content directly from the [Javadocs]() and source code into the documentation.

Using a **[snippet macro]**, we are able to tag portions of the Javadocs and source code for reuse. The macro fetches those snippets from the repository and merges the content into the documentation.

> ✅ **Use the Source!**
>
> Before writing any new content, ask yourself if we could place the content in the repository in either one of the example applications or the Javadocs. Rather than contrive an example, can you pull a snippet from one of the applications? Rather than reiterate Javadoc, could we update the Javadoc and make it a snippet?

- Javadoc
  - {snippet:id=javadoc|javadoc=true|url=com.opensymphony.xwork2.Result}

- Coding Examples
  - {snippet:id=example|lang=xml|javadoc=true|url=src/java/com/opensymphony/xwork2/Result.java}

	Snippet Attributes
id	The *name* of the snippet (**required**).
url	The URL where the snippet can be found (**required**).
lang	The language that the code block. If this snippet is simply text, don't include this parameter and the content will be printed outside of a code block.
javadoc	If true, the content is within a Javadoc block. If this is set to true, then the preceeding "* " (asterisk-space) characters will be stripped before merging the content. Also, the content is assumed to be already HTML escaped and won't be escaped again.

All snippets are marked off by the pattern `START SNIPPET: XXX` and `END SNIPPET: XXX` where `XXX` is the `name` of the snippet that is assigned in the `id` attribute of the macro. The URL is typically a location that points to the project's source control contents. |

About URLs

As you probably noticed in the examples, there are several formats for URL patterns. A fully-qualified URL is always allowed, though that is often not practical. We've customized the macro to be a bit more intelligent with the URL attribute.

- If the URL appears to be a class, we assume it lives in `src/java`, convert all the dots to slashes, and then append `.java` to it.
- If the URL doesn't start with "http", then it is assumed to start with [https://opensymphony.dev.java.net/source/browse/*checkout*/](), as you saw in the third

example.

About snippet markers

When possible, all snippet markers should be in comment blocks. How they are commented depends on where the snippet is being embedded.

```
<!-- START SNIPPET: xxx -->
...
<!-- END SNIPPET: xxx -->
```

```
if (true != false) {
    // START SNIPPET: xxx
    System.out.println("This is some silly code!");
    // END SNIPPET: xxx
}
```

If the snippet is embedded in Javadocs, use HTML comments as they won't render in the Javadocs. For XML examples in Javadocs can be tricky. (See Timer Interceptor for an example.). Javadocs want to use the `<pre>` tag, but you don't want that tag in the snipped content.

One technique is to embed the snippet markers *inside* the `<pre>` tag.

```
* <pre>
* <!-- START SNIPPET: example -->
* &lt;!-- records only the action's execution time --&gt;
* &lt;action name="someAction" class="com.examples.SomeAction"&gt;
*     &lt;interceptor-ref name="completeStack"/&gt;
*     &lt;interceptor-ref name="timer"/&gt;
*     &lt;result name="success"&gt;good_result.ftl&lt;/result&gt;
* &lt;/action&gt;
* <!-- END SNIPPET: example -->
```

Precise Error Reporting

With the multiple levels of configuration, constant overrides, and bean selection, it can be confusing as to how the framework is configured and how it got there. To assist debugging and provide the ability to continue to provide more line-precise error reporting, the configuration loader remembers the location of not only XML elements, but also Java Properties file properties.

How it works is instead of storing configuration properties in a plain java.util.Properties object, we have a special LocatableProperties class in XWork. This class leverages the location classes in XWork to store location information for the whole properties file but also individual properties. We use this to gather configuration properties during configuration loading. This will allow us to display at any given point the location of each property setting. This capability should be very useful when trying to determine what XML or Properties file overwrote what and when.

Interestingly, in addition to remembering the URI and line number of Properties properties, the parser we "borrowed" from Commons Configuration even gathers preceding comments, which are also stored in the Location object.

There are some very interesting possibilities here to increase the transparency of the framework in error and debugging conditions.

Reloading

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".

> Obviously, this feature does not come for free. It will cause your application to run **considerably slower**.

XWork Actions

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 exists, 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. Parmameter 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 = ActionContext.getContext();
Map contextMap = context.getContextMap();
contextMap.get("someParameter");
```

This page last changed on Nov 13, 2006 by jmitchell.

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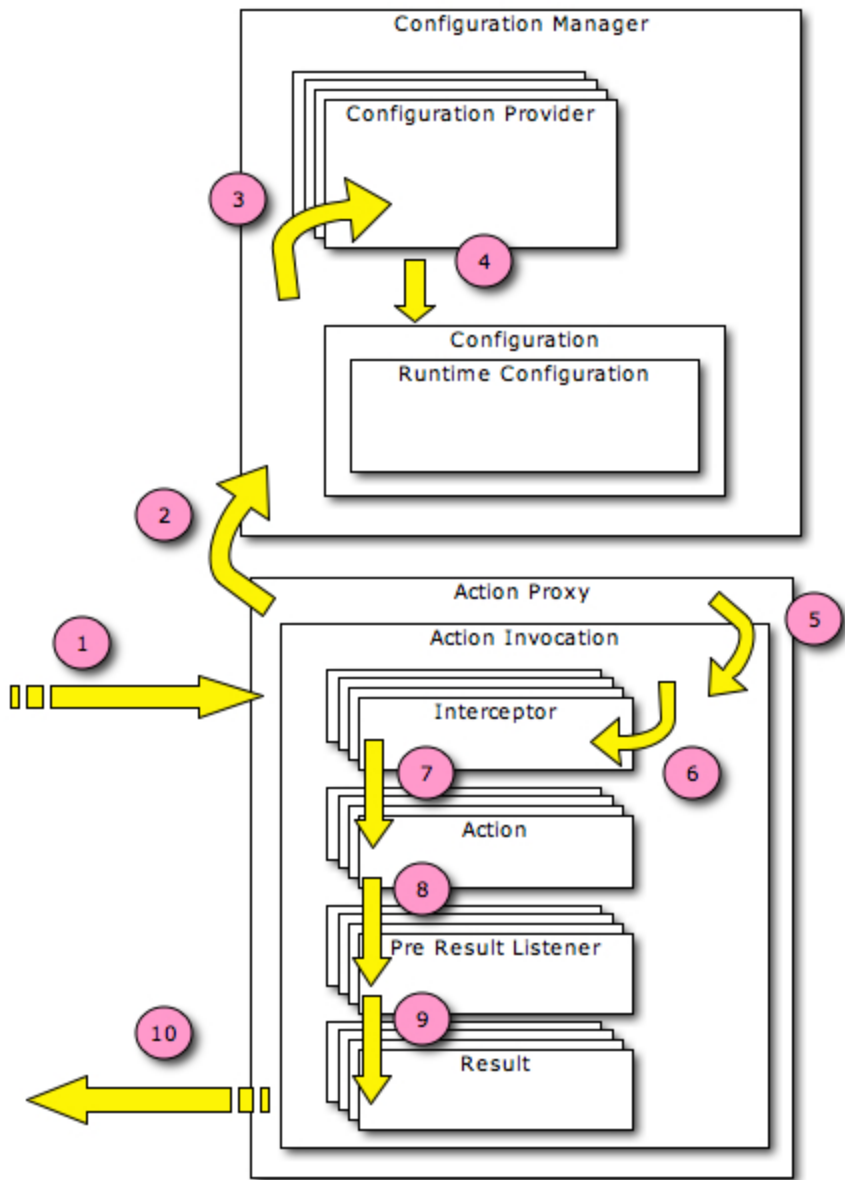
Command Design Pattern

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

⚠️ 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

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 pluged 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 stoted 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 in the xwork.xml configuration file:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE xwork PUBLIC
        "-//OpenSymphony Group//XWork 1.1.1//EN"
        "http://www.opensymphony.com/xwork/xwork-1.1.1.dtd">


<xwork>
        <include file="xwork-default.xml" />
        <package name="default-hello-world" extends="xwork-default" namespace="/helloWorld">
                <result-types>
                        <result-type name="printToConsole"
class="com.opensymphony.xwork2.showcase.PrintToConsoleResult" />
                </result-types>

                <action name="helloWorld"
class="com.opensymphony.xwork2.showcase.helloworld.HelloWorldAction">
                        <result type="printToConsole">
                                <param name="param">${message}</param>
                        </result>
                </action>
        </package>
</xwork>
```

This page last changed on Nov 26, 2006 by husted.

- 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 specifi PropertyAccessor, MethodAccessor, NullHandler and TypeConverter etc. into Ognl runtime.

Global Level

To create a global level conversion, create an 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 hypotenical '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 convertion 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 an example of a hypotenical 'MyAction-conversion.properties'.

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

Custom Converter

The following is an example of a converter. It extends 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.
      ....
   }
  }
```

This page last changed on Jan 02, 2007 by rainerh.

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 encapulate 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 mutiple 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 similiar 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.

Next: **XWork Articles**

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 Jan 02, 2007 by rainerh.

About XWork

XWork 2 is a generic command pattern framework. It forms the core of [Struts 2](). 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

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 seperated 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>
```

Next: XWork Core Concepts

XWork Hibernate Integration

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 imlpementation 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 Jan 03, 2007 by mrdon.

> ⚠️ XWork2 works best with at least JDK 5 (Tiger), but it also ships with a Java 1.4-compatible jar built using RetroTranslator.

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/
    + xwork2-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
   + xwork2-xxx.jar
   + ... jar
 + classes
   + myPackage
     + MyMain.class
 + resources
   + xwork.xml
```

```
<procject ... >
  ...
  <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>
```

Next: **XWork Requirements And Dependencies**

XWork Interceptors

This page last changed on Jan 06, 2007 by rainerh.

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

Default interceptors that comes with XWork2

```
<interceptor name="timer" class="com.opensymphony.xwork2.interceptor.TimerInterceptor"/>
<interceptor name="logger" class="com.opensymphony.xwork2.interceptor.LoggingInterceptor"/>
<interceptor name="chain" class="com.opensymphony.xwork2.interceptor.ChainingInterceptor"/>
<interceptor name="staticParams"
class="com.opensymphony.xwork2.interceptor.StaticParametersInterceptor"/>
<interceptor name="params" class="com.opensymphony.xwork2.interceptor.ParametersInterceptor"/>
<interceptor name="filterParams"
class="com.opensymphony.xwork2.interceptor.ParameterFilterInterceptor"/>
<interceptor name="removeParams"
class="com.opensymphony.xwork2.interceptor.ParameterRemoverInterceptor"/>
<interceptor name="modelDriven"
class="com.opensymphony.xwork2.interceptor.ModelDrivenInterceptor"/>
<interceptor name="scopedModelDriven"
          class="com.opensymphony.xwork2.interceptor.ScopedModelDrivenInterceptor"/>
<interceptor name="validation"
class="com.opensymphony.xwork2.validator.ValidationInterceptor"/>
<interceptor name="workflow"
class="com.opensymphony.xwork2.interceptor.DefaultWorkflowInterceptor"/>
<interceptor name="prepare" class="com.opensymphony.xwork2.interceptor.PrepareInterceptor"/>
<interceptor name="conversionError"
class="com.opensymphony.xwork2.interceptor.ConversionErrorInterceptor"/>
<interceptor name="alias" class="com.opensymphony.xwork2.interceptor.AliasInterceptor"/>
<interceptor name="exception"
class="com.opensymphony.xwork2.interceptor.ExceptionMappingInterceptor"/>
<interceptor name="i18n" class="com.opensymphony.xwork2.interceptor.I18nInterceptor"/>
```

XWork Localization

This page last changed on Dec 01, 2006 by rainerh.

Overview

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 fromthe 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.Quadraped extends Animal;
class acme.test.QuadrapedImpl extends Animal implements Quadraped;
class acme.test.Horse extends QuadrapedImpl;
```

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.QuadrapedImpl.properties
acme.test.Quadraped.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.

XWork Object Factory

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, you will need to create a subclass of ObjectFactory, then tell XWork to use it in the configuration. To configure your ObjectFactory in `xwork.xml`, declare your custom ObjectFactory implementation:

```
<bean name="default" type="com.opensymphony.xwork2.ObjectFactory"
class="com.mycompany.MyObjectFactory" />
```

Now, XWork will use your ObjectFactory instead of the default implementation.

XWork package

This page last changed on Oct 14, 2006 by tm_jee.

XWork2 introduce the notion of package to group related action under the same namepsace. 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

PreResultListener serves as a hook to influence the currect 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

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 extend 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.ProfilingBlock<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

⚠️ 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	2.6.9	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
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

Next: **XWork Tutorial**

Xwork Results

This page last changed on Jan 03, 2007 by mrdon.

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 paremeter 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' paremeter.
    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

```
<result-types>
    <result-type name="chain" class="com.opensymphony.xwork2.ActionChainResult"/>
```

```
</result-types>
```

XWork specific OGNL Features

This page last changed on Dec 01, 2006 by rainerh.

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 explicetly 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.

Xwork Spring Integration

This page last changed on Nov 11, 2006 by tm_jee.

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.setAuowiringStrategy(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 intepreted 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" />
```

```
        ...
        <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>
```

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

> ℹ 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.

> ℹ 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

The following tutorials are available for XWork 2:

- XWork2 Hello World Tutorial — The objective of this simple 'Hello World' tutorial is to be a gentle introduction of the components in xwork2.

XWork2 Hello World Tutorial

The objective of this simple 'Hello World' tutorial is to be a gentle introduction of the components in xwork2.
It shows the required libraries and how to execute a command (an action) that simply prints out 'Hello World' the XWork 2 way.

Step 1: Prepare Project Libraries

You need the following two libraries:

- ognl-2.6.11.jar (or ognl.jar)
- xwork-2.1.0.jar (or xwork.jar)

Step 2: Write up xwork.xml

The `xwork.xml` configuration file, in this case named `xwork-hello-world.xml`, defines a simple package that contains one action and a simple result that prints the data to the console.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE xwork PUBLIC
        "-//OpenSymphony Group//XWork 1.1.1//EN"
        "http://www.opensymphony.com/xwork/xwork-1.1.1.dtd">


<xwork>
        <include file="xwork-default.xml" />
        <package name="default-hello-world" extends="xwork-default" namespace="/helloWorld">
                <result-types>
                        <result-type name="printToConsole"
class="com.opensymphony.xwork2.showcase.PrintToConsoleResult" />
                </result-types>

                <action name="helloWorld"
class="com.opensymphony.xwork2.showcase.helloworld.HelloWorldAction">
                        <result type="printToConsole">
                                <param name="param">${message}</param>
                        </result>
                </action>
        </package>
</xwork>
```

Step 3: Code it up

Next, we need to write the Java code that will initialize XWork and execute our action. The entry point into our application will be a class titled
`com.opensymphony.xwork2.showcase.helloworld.HelloWorldTutorial`.

```
01. /*
02.  * Copyright (c) 2002-2006 by OpenSymphony
```

```
03.  * All rights reserved.
04.  */
05. package com.opensymphony.xwork2.showcase.helloworld;
06.
07. import java.util.LinkedHashMap;
08.
09. import com.opensymphony.xwork2.ActionProxy;
10. import com.opensymphony.xwork2.ActionProxyFactory;
11. import com.opensymphony.xwork2.config.Configuration;
12. import com.opensymphony.xwork2.config.ConfigurationManager;
13. import com.opensymphony.xwork2.config.providers.XmlConfigurationProvider;
14.
15. /**
16.  *
17.  * @author tm_jee
18.  * @version $Date$ $Id$
19.  */
20. public class HelloWorldTutorial {
21.
22.     public static void main(String[] args) throws Exception {
23.
24.             ConfigurationManager confManager = new ConfigurationManager();
25.             confManager.addConfigurationProvider(
26.                         new XmlConfigurationProvider(
27.
"com/opensymphony/xwork2/showcase/helloworld/xwork-hello-world.xml",
28.                                             true));
29.
30.          Configuration conf = confManager.getConfiguration();
31.             ActionProxyFactory actionProxyFactory =
conf.getContainer().getInstance(ActionProxyFactory.class);
32.             ActionProxy actionProxy = actionProxyFactory.createActionProxy(
33.                         "/helloWorld", "helloWorld", new LinkedHashMap());
34.
35.
36.             actionProxy.execute();
37.     }
38. }
```

Lines 24-28 show the XWork framework being initialized with our configuration file. Lines 30-33 show how we obtain an `ActionProxy` object that handles the Action, Interceptors, and Result execution. Finally, in line 36, we execute XWork.

NOTE: Between line 24 and 25 should come the following :
confManager.addConfigurationProvider(new XWorkConfigurationProvider());

Step 4: Create the Result and Action

The action:

```
01. /*
02.  * Copyright (c) 2002-2006 by OpenSymphony
03.  * All rights reserved.
04.  */
05. package com.opensymphony.xwork2.showcase.helloworld;
06.
07. import org.apache.commons.logging.Log;
08. import org.apache.commons.logging.LogFactory;
09.
10. import com.opensymphony.xwork2.ActionSupport;
11.
12. /**
13.  *
14.  * @author tm_jee
15.  * @version $Date$ $Id$
```

```
16.  */
17. public class HelloWorldAction extends ActionSupport {
18.
19.      private static final long serialVersionUID = 6874543345469426109L;
20.
21.      private static final Log _log = LogFactory.getLog(HelloWorldAction.class);
22.
23.      private String message;
24.
25.      public String getMessage() { return message; }
26.      public void setMessage(String message) { this.message = message; }
27.
28.      @Override
29.      public String execute() throws Exception {
30.
31.              _log.debug("execute ...");
32.
33.              message = "Hello World";
34.
35.              return SUCCESS;
36.      }
37. }
```

The result:

```
01. /*
02.  * Copyright (c) 2002-2006 by OpenSymphony
03.  * All rights reserved.
04.  */
05. package com.opensymphony.xwork2.showcase;
06.
07. import org.apache.commons.logging.Log;
08. import org.apache.commons.logging.LogFactory;
09.
10. import com.opensymphony.xwork2.ActionInvocation;
11. import com.opensymphony.xwork2.Result;
12. import com.opensymphony.xwork2.util.TextParseUtil;
13.
14.
15. /**
16.  *
17.  * @author tm_jee
18.  * @version $Date$ $Id$
19.  */
20. public class PrintToConsoleResult implements Result {
21.
22.      private static final Log _log = LogFactory.getLog(PrintToConsoleResult.class);
23.
24.      private static final long serialVersionUID = -6173326554804520601L;
25.
26.      private String param = "whatsoever";
27.
28.      public void setParam(String param) { this.param = param; }
29.      public String getParam() { return this.param; }
30.
31.      public void execute(ActionInvocation invocation) throws Exception {
32.
33.              _log.debug("execute ...");
34.
35.              String result = TextParseUtil.translateVariables(param, invocation.getStack());
36.
37.              System.out.println(result);
38.      }
39. }
```

XWork Validation

This page last changed on Jan 03, 2007 by mrdon.

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

```
public class ExpressionValidator extends ValidatorSupport {

    private String expression;
```

```
    public void setExpression(String expression) {
        this.expression = expression;
    }

    public String getExpression() {
        return expression;
    }

    public void validate(Object object) throws ValidationException {
        Boolean answer = Boolean.FALSE;
        Object obj = null;

        try {
            obj = getFieldValue(expression, object);
        } catch (ValidationException e) {
            throw e;
        } catch (Exception e) {
            // let this pass, but it will be logged right below
        }

        if ((obj != null) && (obj instanceof Boolean)) {
            answer = (Boolean) obj;
        } else {
            log.warn("Got result of " + obj + " when trying to get Boolean.");
        }

        if (!answer.booleanValue()) {
            if (log.isDebugEnabled()) log.debug("Validation failed on expression " + expression
+ " with validated object "+ object);
            addActionError(object);
        }
    }
}
```

Field validator

```
public class DoubleRangeFieldValidator extends FieldValidatorSupport {

    String maxInclusive = null;
    String minInclusive = null;
    String minExclusive = null;
    String maxExclusive = null;

    Double maxInclusiveValue = null;
    Double minInclusiveValue = null;
    Double minExclusiveValue = null;
    Double maxExclusiveValue = null;

    public void validate(Object object) throws ValidationException {
        String fieldName = getFieldName();
        Double value;
        try {
            Object obj = this.getFieldValue(fieldName, object);
            if (obj == null) {
                return;
            }
            value = Double.valueOf(obj.toString());
        } catch (NumberFormatException e) {
            return;
        }

        parseParameterValues();
        if ((maxInclusiveValue != null && value.compareTo(maxInclusiveValue) > 0) ||
                (minInclusiveValue != null && value.compareTo(minInclusiveValue) < 0) ||
                (maxExclusiveValue != null && value.compareTo(maxExclusiveValue) >= 0) ||
                (minExclusiveValue != null && value.compareTo(minExclusiveValue) <= 0)) {
            addFieldError(fieldName, object);
        }
    }

    private void parseParameterValues() {
```

```
            this.minInclusiveValue = parseDouble(minInclusive);
            this.maxInclusiveValue = parseDouble(maxInclusive);
            this.minExclusiveValue = parseDouble(minExclusive);
            this.maxExclusiveValue = parseDouble(maxExclusive);
    }

    private Double parseDouble (String value) {
        if (value != null) {
            try {
                return Double.valueOf(value);
            } catch (NumberFormatException e) {
                if (log.isWarnEnabled()) {
                    log.warn("DoubleRangeFieldValidator - [parseDouble]: Unable to parse given
double parameter " + value);
                }
            }
        }
        return null;
    }

    public void setMaxInclusive(String maxInclusive) {
        this.maxInclusive = maxInclusive;
    }

    public String getMaxInclusive() {
        return maxInclusive;
    }

    public void setMinInclusive(String minInclusive) {
        this.minInclusive = minInclusive;
    }

    public String getMinInclusive() {
        return minInclusive;
    }

    public String getMinExclusive() {
        return minExclusive;
    }

    public void setMinExclusive(String minExclusive) {
        this.minExclusive = minExclusive;
    }

    public String getMaxExclusive() {
        return maxExclusive;
    }

    public void setMaxExclusive(String maxExclusive) {
        this.maxExclusive = maxExclusive;
    }
}
```

Default Validators

```
<validators>
    <validator name="required"
class="com.opensymphony.xwork2.validator.validators.RequiredFieldValidator"/>
    <validator name="requiredstring"
class="com.opensymphony.xwork2.validator.validators.RequiredStringValidator"/>
    <validator name="int"
class="com.opensymphony.xwork2.validator.validators.IntRangeFieldValidator"/>
    <validator name="long"
class="com.opensymphony.xwork2.validator.validators.LongRangeFieldValidator"/>
    <validator name="short"
class="com.opensymphony.xwork2.validator.validators.ShortRangeFieldValidator"/>
    <validator name="double"
class="com.opensymphony.xwork2.validator.validators.DoubleRangeFieldValidator"/>
    <validator name="date"
class="com.opensymphony.xwork2.validator.validators.DateRangeFieldValidator"/>
    <validator name="expression"
class="com.opensymphony.xwork2.validator.validators.ExpressionValidator"/>
    <validator name="fieldexpression"
```

```
"com.opensymphony.xwork2.validator.validators.FieldExpressionValidator"/>
    <validator name="email"
class="com.opensymphony.xwork2.validator.validators.EmailValidator"/>
    <validator name="url" class="com.opensymphony.xwork2.validator.validators.URLValidator"/>
    <validator name="visitor"
class="com.opensymphony.xwork2.validator.validators.VisitorFieldValidator"/>
    <validator name="conversion"
class="com.opensymphony.xwork2.validator.validators.ConversionErrorFieldValidator"/>
    <validator name="stringlength"
class="com.opensymphony.xwork2.validator.validators.StringLengthFieldValidator"/>
    <validator name="regex"
class="com.opensymphony.xwork2.validator.validators.RegexFieldValidator"/>
    <validator name="conditionalvisitor"
class="com.opensymphony.xwork2.validator.validators.ConditionalVisitorFieldValidator"/>
</validators>
```

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

```xml
<validators>
        <validator type="expression">
                <param name="expression"><![CDATA[name != null && age != null]]></param>
                <message>Both fields are required</message>
        </validator>
        <field name="name">
                <field-validator type="requiredstring">
                        <message>Name is mandatory</message>
                </field-validator>
        </field>
        <field name="age">
                <field-validator type="requiredstring">
                        <message>Age is mandatory</message>
                </field-validator>
                <field-validator type="int">
                        <param name="min">20</param>
                        <param name="max">50</param>
                        <message>Age must be between 20 and 50</message>
                </field-validator>
        </field>
</validators>
```

XWork Value Stack

This page last changed on Nov 11, 2006 by tm_jee.

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 followng 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) • 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 = fincValue("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");
```