STRUTS 2 - ANNOTATIONS TYPES

Struts 2 applications can use Java 5 annotations as an alternative to XML and Java properties configuration. Here is the list of most important annotations related to different categories:

Namespace Annotation *ActionAnnotation*

The @Namespace annotation allows the definition of an Action's namespace in the **Action** class rather than based on Zero Configuration's conventions.

```
@Namespace("/content")
public class Employee extends ActionSupport{
  ...
}
```

Result Annotation - *ActionAnnotation*

The @Result annotation allows the definition of Action results in the Action class rather than an XML file.

```
@Result(name="success", value="/success.jsp")
public class Employee extends ActionSupport{
 ...
}
```

Results Annotation - *ActionAnnotation*

The @Results annotation defines a set of results for an Action.

```
@Results({
   @Result(name="success", value="/success.jsp"),
   @Result(name="error", value="/error.jsp")
})
public class Employee extends ActionSupport{
 ...
}
```

After Annotation - *InterceptorAnnotation*

The @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.

```
public class Employee extends ActionSupport{
   @After
   public void isValid() throws ValidationException {
      // validate model object, throw exception if failed
   }
   public String execute() {
      // perform secure action
      return SUCCESS;
   }
}
```

Before Annotation - *InterceptorAnnotation*

The @Before annotation marks a action method that needs to be called before the main action method and the result was executed. Return value is ignored.

```
public class Employee extends ActionSupport{
   @Before
   public void isAuthorized() throws AuthenticationException {
```

```
      // authorize request, throw exception if failed
   }
   public String execute() {
      // perform secure action
      return SUCCESS;
   }
}
```

BeforeResult Annotation - *InterceptorAnnotation*

The @BeforeResult annotation marks a action method that needs to be executed before the result. Return value is ignored.

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

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

ConversionErrorFieldValidator Annotation - *ValidationAnnotation*

This validation annotation checks if there are any conversion errors for a field and applies them if they exist.

```
public class Employee extends ActionSupport{
   @ConversionErrorFieldValidator(message = "Default message",
                        key = "i18n.key", shortCircuit = true)
   public String getName() {
       return name;
   }
}
```

DateRangeFieldValidator Annotation - *ValidationAnnotation*

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

```
public class Employee extends ActionSupport{
   @DateRangeFieldValidator(message = "Default message",
   key = "i18n.key", shortCircuit = true,
   min = "2005/01/01", max = "2005/12/31")
   public String getDOB() {
       return dob;
   }
}
```

DoubleRangeFieldValidator Annotation - *ValidationAnnotation*

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

```
public class Employee extends ActionSupport{

   @DoubleRangeFieldValidator(message = "Default message",
   key = "i18n.key", shortCircuit = true,
   minInclusive = "0.123", maxInclusive = "99.987")
   public String getIncome() {
       return income;
   }
}
```

EmailValidator Annotation - *ValidationAnnotation*

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

```
public class Employee extends ActionSupport{

   @EmailValidator(message = "Default message",
   key = "i18n.key", shortCircuit = true)
   public String getEmail() {
       return email;
   }
}
```

ExpressionValidator Annotation - *ValidationAnnotation*

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

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

IntRangeFieldValidator Annotation - *ValidationAnnotation*

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

```
public class Employee extends ActionSupport{

   @IntRangeFieldValidator(message = "Default message",
   key = "i18n.key", shortCircuit = true,
   min = "0", max = "42")
   public String getAge() {
       return age;
   }
}
```

RegexFieldValidator Annotation - *ValidationAnnotation*

This annotation validates a string field using a regular expression.

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

RequiredFieldValidator Annotation - *ValidationAnnotation*

This validation annotation checks that a field is non-null. The annotation must be applied at method level.

```
public class Employee extends ActionSupport{

   @RequiredFieldValidator(message = "Default message",
   key = "i18n.key", shortCircuit = true)
   public String getAge() {
       return age;
   }
}
```

RequiredStringValidator Annotation - *ValidationAnnotation*

This validation annotation checks that a String field is not empty $i.e. non-null with a length > 0$.

```
public class Employee extends ActionSupport{

   @RequiredStringValidator(message = "Default message",
```

```
    key = "i18n.key", shortCircuit = true, trim = true)
    public String getName() {
        return name;
    }
}
```

StringLengthFieldValidator Annotation - *ValidationAnnotation*

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.

```
public class Employee extends ActionSupport{

   @StringLengthFieldValidator(message = "Default message",
   key = "i18n.key", shortCircuit = true,
   trim = true, minLength = "5",  maxLength = "12")
   public String getName() {
       return name;
   }
}
```

UrlValidator Annotation - *ValidationAnnotation*

This validator checks that a field is a valid URL.

```
public class Employee extends ActionSupport{

   @UrlValidator(message = "Default message",
   key = "i18n.key", shortCircuit = true)
   public String getURL() {
       return url;
   }
}
```

Validations Annotation - *ValidationAnnotation*

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

```
public class Employee extends ActionSupport{

  @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.")}
   )
   public String getName() {
       return name;
   }
}
```

CustomValidator Annotation - *ValidationAnnotation*

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

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

Conversion Annotation - *TypeConversionAnnotation*

This is a marker annotation for type conversions at Type level. The Conversion annotation must be applied at Type level.

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

CreateIfNull Annotation - *TypeConversionAnnotation*

This annotation sets the CreateIfNull for type conversion. The CreateIfNull annotation must be applied at field or method level.

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

Element Annotation - *TypeConversionAnnotation*

This annotation sets the Element for type conversion. The Element annotation must be applied at field or method level.

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

Key Annotation - *TypeConversionAnnotation*

This annotation sets the Key for type conversion. The Key annotation must be applied at field or method level.

```
@Key( value = java.lang.Long.class )
private Map<Long, User> userMap;
```

KeyProperty Annotation - *TypeConversionAnnotation*

This annotation sets the KeyProperty for type conversion. The KeyProperty annotation must be applied at field or method level.

```
@KeyProperty( value = "userName" )
protected List<User> users = null;
```

TypeConversion Annotation - *TypeConversionAnnotation*

This annotation annotation is used for class and application wide conversion rules. The TypeConversion annotation can be applied at property and method level.

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