Property Validation Usage

Intro

The Intro page provides an overview and describes the motivation for the features described below. This page explains the most important APIs and mechanisms of the property validation module provided by ExtVal. Please note that this page doesn't show all possibilities. **If you have any question, please contact the community!**

Dependencies

The page ExtVal Modules provides an overview about ExtVal modules and how to add them to your project.

For using the features described in this page, you have to add the core and the property validation module for the JSF version you are using.

Hello-World examples

Setup

Required steps for using ExtVal:

- Download the required ExtVal jar-files or checkout the Source Code and build it. (details)
- Add these jar-files to your classpath (manually or via maven)
- Use the annotations (details)

That's it!

Simple validation

The sample page

The sample page

```
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<html>
<head>
  <title>Hello World!</title>
</head>
<body>
<f:view>
  <h:form id="mainForm">
    <h:panelGrid columns="3">
      <h:outputLabel for="name" value="Please enter your name" />
      <h:inputText id="name" value="#{helloWorld.name}" />
      <h:message for="name" showSummary="true" showDetail="false" />

      <h:commandButton value="Press me" action="#{helloWorld.send}" />
      <h:panelGroup />
      <h:panelGroup />
    </h:panelGrid>
  </h:form>
</f:view>
</body>
</html>
```

Note that there is no JSF validator in the page.

The sample bean

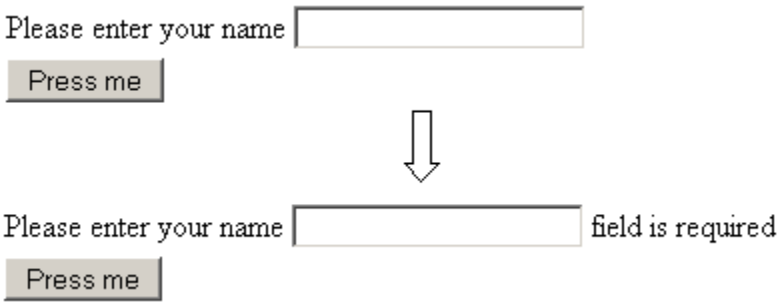
Fragment of the HelloWorldController bean

```
public class HelloWorldController {

    @Required
    private String name;

    // getters and setters omitted for brevity
}
```

The result



As you can see, the field is not valid, as the `@Required` annotation causes the field to be required.

Cross validation

Usage of cross validation

```
public class Person {

    @NotEquals("lastName")
    private String firstName;

    private String lastName;

    // Getters and setters omitted for brevity
}
```

The rest is equivalent to the simple validation (just bind the properties to your input components). Note how we refer to the `lastName` property by simply using the name of the property.

Available annotations

With the property validation module, there are two ways to define validation rules. The first is to (re)use JPA annotations, see #JPA based validation. The second is to use ExtVal's simple validation annotations, see #Simple validation and #Cross-Validation.

JPA based validation

JPA annotations are used automatically for UI validation as soon as you:

- Add myfaces-extval-core-**.jar and myfaces-extval-property-validation-**.jar to the classpath
- Use JPA annotations within the model (e.g.: `@Column(nullable = false)` -> the field is required)

That's it!
The table below gives an overview of the validations that are generated, based on JPA annotations.

JPA Annotation	Generated validation

`@Column`	• Field will be required if `nullable == false`. • If the field is a `String` and the `columns` property is set on the `@Column` annotation, the maximum length of the `String` will be validated accordingly.
`@Basic`	Field will be required if `optional == false`.
`@OneToOne`	Field will be required if `optional == false`.
`@ManyToOne`	Field will be required if `optional == false`.
`@Id`	Field will be required.

(A simple demo is available here: demo_000)

Simple validation

Annotation	Description
DoubleRange	delegates to the implementation of `javax.faces.validator.DoubleRangeValidator`
JoinValidation	to reuse (point to) annotations of a different property (see re-use existing annotations)
Length	delegates to the implementation of `javax.faces.validator.LengthValidator`
LongRange	delegates to the implementation of `javax.faces.validator.LongRangeValidator`
Pattern	use a regular expression for validation
Required	alternative to the required attribute
SkipValidation	allows to keep validation optional. (the annotations which are afterwards and support this mechanism)
Validator	generic validator to delegate validation to an existing jsf validator e.g.: `@Validator(EmailValidator.class)`

Cross-Validation

Annotation	Description
DateIs	validates if a date is equal, before or after a second date
Equals	validates if two values are equal
NotEquals	validates if two values are different
RequiredIf	validates if a value is required depending on a second value (if it is empty or not)
EmptyIf	opposite of `@RequiredIf`

Existing/3rd party constraints

ExtVal has no special requirements for annotations. It's the responsibility of the validation strategy to know how to validate the annotation. So you can use annotations within any layer without introducing an ExtVal dependency below the view layer. If you would like to validate 3rd party annotations you can provide a mapping. With the same mechanism you can replace existing (ExtVal) validation strategies. Find detailed information below.

Custom annotations

Simple annotations

ExtVal provides the possibility to validate annotations with so called validation strategies.

The simplest case is to create a custom annotation and to use a name convention for the validator (= validation strategy). The validation strategy has to implement the `ValidationStrategy` interface. Or you can extend a class which implements this interface in-/directly. (A simple demo is available here: demo_002, demo_006)

Hint

If you don't like the default conventions, you can provide a custom name mapper, or you provide a mapping between annotations and the validation strategies (via properties file or ExtVal Java API), or ...

Minimal custom constraint

```
package my.custom.package

@Target({METHOD, FIELD})
@Retention(RUNTIME)
public @interface CustomConstraint {
}
```

Note that this is just a minimalistic annotation definition. There's nothing ExtVal-specific here.

Minimal validation strategy implementation

```
package my.custom.package

public class CustomConstraintValidationStrategy implements ValidationStrategy {

    public void validate(FacesContext facesContext, UIComponent uiComponent,
                         MetaDataEntry metaDataEntry, Object convertedObject) {

        //custom validation logic
    }
}
```

Note that only one method (`validate()`) has to be implemented. The naming convention here is that the validation strategy has the same name as the custom annotation, with `ValidationStrategy` appended. The annotation and the validation strategy must also be in the same package.

That's just the simplest case. You can also use one of the other available name conventions or you can provide a custom convention or a custom name mapper or ...

Hint

It's recommended to subclass `AbstractValidatorAdapter` or `AbstractAnnotationValidationStrategy` rather than implementing the `ValidationStrategy` interface yourself.

In ExtVal r3+ validation strategies don't have to be aware of empty/null values. So it's safe to delegate to legacy JSF validators. If a validation strategy should validate such values, it's possible to annotate the validation strategy with `@NullValueAwareValidationStrategy` and/or `@EmptyValueAwareValidationStrategy`.

Cross validation annotations

The class below implements the validation strategy for the `@Equals` annotation. Note that the class can be relatively simple, since the hard work is done for us in the `AbstractCompareStrategy`.

Custom cross-validation strategies

```
@SkipValidationSupport
public class EqualsStrategy extends AbstractCompareStrategy {

    public boolean useTargetComponentToDisplayErrorMsg(CrossValidationStorageEntry crossValidationStorageEntry)
{
        return true;
    }

    protected String getValidationErrorMsgKey(Annotation annotation, boolean isTargetComponent) {
        return ((Equals) annotation).validationErrorMsgKey();
    }

    public boolean isViolation(Object object1, Object object2, Annotation annotation) {
        return object1 != null && !object1.equals(object2);
    }

    public String[] getValidationTargets(Annotation annotation) {
        return ((Equals) annotation).value();
    }
}
```

Hint

Be aware that the `isViolation()` method should return `true` if the input values are not valid (not equal in this case). Although the name of the method is very clear, you might have expected the opposite.

In case of model aware cross-validation the validation error message is displayed at the source component instead of the target component. Since the message is still meaningful, there's no need to provide a special reverse message. (This can be changed by returning `false` in the `useTargetComponentToDisplayErrorMsg()` method.

Cross validation details

As we've seen before, we can easily reference a property within the same bean by using the property name.

Referencing a local property

```
public class Person
{
    @NotEquals("lastName")
    private String firstName;

    private String lastName;
...
}
```

We can also refer to properties of related beans, as is shown in the example below:

Referencing a property of a local reference

```
public class RegistrationPage
{
    private Person person = new Person();

    @Equals("person.password")
    private String oldPassword;
...
}
```

It is even possible to use EL-like expressions to refer to properties of beans that will be available at runtime.

Referencing a property of a bean

```
public class RegistrationPage
{
    @Equals("#{person.password}")
    private String oldPassword;
...
}
```


Be Careful

You should realize that by referencing bean properties, you may be introducing a dependency of the View-layer.

Details of the examples

Within the page you just have to bind a property which has a cross-validation constraint.
If both, the source property as well as the referenced target, are bound to input components of the same form, the converted objects are used for validation. In case of a validation error, the validation error message(s) is/are displayed.

Model aware cross-validation (since 1.x.2)

If the target of the cross-validation isn't bound to an input component of the same form, the model value of the target is used for validation. The property which triggers the cross-validation always "uses" the converted object (not the model value).

That means:

- Normal cross-validation: converted object (constraint source) + converted object (constraint target)
- Model aware cross-validation: converted object (constraint source) + model object (constraint target)

Reverse validation error messages

In case of model aware cross-validation the following issues are possible:

- Missing "target component" to display validation error message
- Inconsistent validation messages (the message is automatically displayed at the "source component", but the default message makes no sense for the user)

Solution: The validation strategy optionally provides a meaningful validation error message. It's called a reverse validation error message.

Example

```
...
    @Override
    protected String getReverseErrorMessageSummary(Annotation annotation)
    {
        return "meaningful validation error message summary";
    }
...
    @Override
    protected String getReverseErrorMessageDetail(Annotation annotation)
    {
        return "meaningful validation error message details";
    }
...
```

... override these methods to display a meaningful reverse validation message at the source component.

Conditional validation via cross-validation

Conditional validation is possible e.g. via `@SkipValidation` - that works if the condition is already available before the current request.
It might be interesting to validate constraints based on a condition which is available with the same request.

The group validation concept available via BV as well as via a possible add-on for ExtVal-Constraints (a simple example is available at demo of group validation light) might already solve your requirements.

If you prefer a style which directly uses JSF mechanisms, you can also use cross-validation for it. The validation target can be used as condition.

It's possible since the first version of ExtVal. Due to new features introduced in the 3rd release of ExtVal you will see additional things in the example you might not have seen so far. However, they aren't required since the approach just uses the mechanism of cross-validation.:

Custom cross-validation annotation

```
@Target({METHOD, FIELD})
@Retention(RUNTIME)
@Documented
@UsageInformation(UsageCategory.API)
public @interface ValidateLengthIf
{
    String[] validateIf();

    int minimum() default 0;

    int maximum() default Integer.MAX_VALUE;

    Class<? extends ValidationParameter>[] parameters() default ViolationSeverity.Error.class;
}
```

Custom cross-validation strategy

```
@SkipValidationSupport
public class ValidateLengthIfValidationStrategy extends AbstractCompareStrategy<ValidateLengthIf>
{
    private UIComponent component;
    private FacesMessage facesMessage;

    @Override
    protected void initCrossValidation(CrossValidationStorageEntry crossValidationStorageEntry)
    {
        this.component = crossValidationStorageEntry.getComponent();
    }

    public boolean isViolation(Object source, Object target, ValidateLengthIf annotation)
    {
        if (Boolean.TRUE.equals(target))
        {
            try
            {
                LengthValidator lengthValidator = resolveLengthValidator();
                lengthValidator.setMinimum(annotation.minimum());
                lengthValidator.setMaximum(annotation.maximum());
                lengthValidator.validate(FacesContext.getCurrentInstance(), this.component, source);
            }
            catch (ValidatorException e)
            {
                this.facesMessage = e.getFacesMessage();
                return true;
            }
        }
        return false;
    }

    private LengthValidator resolveLengthValidator()
    {
        return (LengthValidator)FacesContext.getCurrentInstance()
                .getApplication().createValidator("javax.faces.Length");
    }

    public String[] getValidationTargets(ValidateLengthIf annotation)
    {
        return annotation.validateIf();
    }

    @Override
    public boolean useTargetComponentToDisplayErrorMsg(CrossValidationStorageEntry crossValidationStorageEntry)
    {
        return false;
    }

    @Override
    protected String getErrorMessageSummary(ValidateLengthIf annotation, boolean isTargetComponent)
    {
        return this.facesMessage.getSummary();
    }

    @Override
    protected String getErrorMessageDetail(ValidateLengthIf annotation, boolean isTargetComponent)
    {
        return this.facesMessage.getDetail();
    }

    protected String getValidationErrorMsgKey(ValidateLengthIf annotation, boolean isTargetComponent)
    {
        //for using the message of the std. validator instead of a cross-validation-key for extval message
resolving
        return null;
    }
}
```

Provide/replace validation strategy mappings

If you don't like to use the name mapping concept, you can provide static mappings between annotations and validation strategies.
Use the convention for a mapping file (`org.apache.myfaces.extensions.validator.custom.strategy_mappings.properties`) (it's customizable)

or use the ExtVal Java API:

Register a resource-bundle file which contains an annotation/validation strategy mapping:

Alternative config approach via a property file

```
StaticConfiguration<String, String> staticConfig = new StaticResourceBundleConfiguration();
staticConfig.setSourceOfMapping("[custom package + name of the properties file.]");
ExtValContext.getContext().addStaticConfiguration(StaticConfigurationNames.
META_DATA_TO_VALIDATION_STRATEGY_CONFIG, staticConfig);
```

It's also used internally to provide the JPA based validation support. So you can find an example at the `PropertyValidationModuleStartupListener` .
A similar approach is used internally by the annotation based config extension

This approach is more typesafe - a simple example:

Alternative config approach which manual mapping

```
StaticInMemoryConfiguration staticConfig = new StaticInMemoryConfiguration();
staticConfig.addMapping(CustomConstraint.class.getName(), CustomValidator.class.getName());
ExtValContext.getContext().addStaticConfiguration(StaticConfigurationNames.
META_DATA_TO_VALIDATION_STRATEGY_CONFIG, staticConfig);
```

Hint

If you also don't like the approach above, you can implement your own `ValidationStrategyFactory` to introduce your own concept.

I18N support

see Internationalization

Spring support

It's possible to provide a validation strategy as Spring bean.

Use-cases:

- Inject a Spring bean into the validation strategy
- AOP exception handling
- ...

Spring based dependency injection support for validation strategies

```
<!-- The name of annotation: @CustomRequired -->

<!-- Part of the Spring configuration: -->

<bean id="customRequiredValidationStrategy" class="..." lazy-init="true">
    <property name="messageResolver" ref="customMsgResolver"/>
    <property name="requiredValidationService" ref="demoRequiredValidationService"/>
</bean>

<bean id="customMsgResolver"
      class="org.apache.myfaces.extensions.validator.core.validation.message.resolver.
DefaultValidationErrorMessageResolver"
      lazy-init="true">

    <!-- With JSF 1.2 you can use the var name of resource-bundle see faces-config.xml -->
    <property name="messageBundleVarName" value="messages"/>
</bean>

<bean id="demoRequiredValidationService" class="..."/>
```

The bean name follows the available name conventions.
(Also custom name conventions are supported.)

A simple demo is available here: demo_106

Furthermore, it's possible to provide a Meta-Data Transformer as Spring bean.

Constraint Aspects

Since r3 ExtVal uses the concept of constraint aspects for generic validation parameters (in case of bean-validation it's a subset which is called validation **payload**).

Available constraint aspects

- ViolationSeverity
- DisableClientSideValidation
- CaseInsensitive

Mapped Constraint Source

The concept is similar to `@JoinValidation`. However, this version is more type-save and it is supported by the property- as well as the bean-validation module.

Since both modules support the same syntax you will find further information here.