Vaadin Bean Validation Add-on 1.0.0 Manual

Henri Sara

Table of Contents

Introduction

Bean Validation for Vaadin Add-on consists of a Vaadin validator and a form that automatically applies that validator. It uses the Java Bean Validation API 1.0 (JSR-303) for validating beans based on annotations on their fields.

Using BeanValidationForm

In most cases where bean validation is used, all the relevant fields of beans should be automatically validated. To achieve that, `BeanValidationForm` can be used. `BeanValidator` is automatically added for all relevant fields, and fields annotated as `@NotNull` are marked as required.

If the item used is a `BeanItem<T>` where `T` is the bean class, properties are validated based on the run-time type of the bean. Otherwise, the validation is only based on the properties present in the bean class given as a parameter to the `BeanValidationForm` constructor, and not the run-time type of the bean.

A sample bean with validation annotations:

```
import javax.validation.constraints.*;

public class BeanToValidate {
    @NotNull
    @Size(min = 3, max = 16)
    private String firstname;

    @NotNull(message = "Last name must not be empty")
    private String lastname;

    @Min(value = 18, message = "Must be 18 or above")
    @Max(150)
    private int age;

    @Digits(integer = 3, fraction = 2)
    private String decimals;
```

```
    public String getFirstname() {
        return firstname;
    }

    public void setFirstname(String firstname) {
        this.firstname = firstname;
    }

    // ... other accessors
}
```

Using `BeanValidationForm` for editing a `BeanToValidate`:

```
BeanToValidate myBean = new BeanToValidate();

BeanValidationForm<BeanToValidate> form =
    new BeanValidationForm<BeanToValidate>(BeanToValidate.class);
// use a BeanItem<BeanToValidate> for full validation
form.setItemDataSource(new BeanItem<BeanToValidate>(myBean));

// perform validation when a field loses focus, but do not
// update myBean before form.commit() is called
form.setImmediate(true);
form.setWriteThrough(false);
```

Using BeanValidationValidator

It is also possible to use `BeanValidationValidator` directly, applying it as a validator for a single field. This can be useful e.g. when the field is not on a `Form`.

To apply `BeanValidationValidator` directly to a field, there are two alternative approaches:

- Use the static method `BeanValidationValidator.addValidator(Field field, Object propertyId, Class<?> beanClass)` to add the validator using bean validation annotations and mark the field with the `@NotNull` annotation as required.
- Create a `BeanValidationValidator` and add it to a `Field` as a normal Vaadin validator.

Using the same `BeanToValidate` as in the previous example, create a stand-alone text field for the first name:

```
Field firstNameField = new TextField();
BeanValidationValidator.addValidator(firstNameField,
    "firstName", BeanToValidate.class);
```

Using Custom Validators

The bean validation API also allows users to define and use custom bean validators. Here is a short example on how to define and use one:

The validation annotation `@SingleDigit`.

```
@Target({METHOD, FIELD})
@Retention(RUNTIME)
@ConstraintValidator(SingleDigitValidator.class)
public @interface SingleDigit {
    String message() default "Not a single digit";
    String[] groups() default {};
```

The validator class `SingleDigitValidator`.

```
public class SingleDigitValidator implements Constraint<SingleDigit> {
    public void initialize(SingleDigit constraintAnnotation) {
        //no initialization needed
    }

    public boolean isValid(Object object) {
        if ( object == null) return true;
        if ( ! (object instanceof String) )
            throw new IllegalArgumentException(
                "@SingleDigit only applies to String");
        String stringValue = (String) object;
        if ( stringValue.length() != 1 ) return false;
        try {
            Integer.parseInt( stringValue );
            return true;
        } catch (NumberFormatException nfe) {
            return false;
        }
    }
}
```

In the bean to validate, the annotation `@SingleDigit` can now be applied to a field:

```
    @SingleDigit
    private String fieldToValidate;
```

For a more detailed treatment of custom bean validators, see the JSR 303 Bean Validation specification and other available resources.

Known issues

When using Vaadin validators (not only `BeanValidationValidator`), the form is also validated when setting the value programmatically, before any user input. Therefore, some fields may show the error indicator before the user interacts with them.