JideFX Validation Developer Guide

Table of Contents

Purpose of This Document

Welcome to the **JideFX Validation**, a set of features related to the validation for the JavaFX platform. This document is for developers who want to develop applications using **JideFX Validation**.

What is JideFX Validation

Validation involves adding the validation logic to any nodes, generating the validation result and displaying the result. It is a complete end-to-end solution. A well-designed validation framework should be:

- Won't change the existing code logic. Can be added or removed on fly
- Can be used to validate any node
- The validation result can be passed to anyone who is interested in getting it
- The result can be displayed in a non-intrusive way. That's it won't affect the existing layout.

Packages

The table below lists the package in the JideFX Validation product.

Packages	Description
jidefx.scene.control.validation	**Validation related classes**

Dependency

The **JideFX Validation** product depends on the **JideFX Common** and the **JideFX Decoration**.

Validator

A Validator is a Callback that takes a ValidationObject and returns a ValidationEvent.

```
interface Validator<T> extends Callback<ValidationObject<T>, ValidationEvent<T>>
```

ValidationObject

The ValidationObject is the input to the Validator. It has three fields. See below.

```
public class ValidationObject {
    /**
     * The source. It is usually the node to be validated.
     */
```

```
    private Object _source;

    /**
     * New value.
     */
    private Object _newValue;

    /**
     * Previous value. May be null if not known.
     */
    private Object  oldValue;

 ...
}
```

ValidationObject is an object containing the information needed by a Validator. It has three things - the source, the new value and old value.

The source is the object who has the Validator. It is usually the Node to be validated. For example, in the case of validating a text field, the source will be the text field.

Normally ValidationObject is accompanied by the old and new value. If the value is a primitive type, it must be wrapped as the corresponding java.lang.* Object type (such as Integer or Boolean.

Null values may be provided for the old and the new values if their true values are not known. The new value could be a different data type which were failed to be converted to the expected data type. For example, for an integer field, the new value could be String if the value cannot be converted to an integer. A correctly–written Validator should check for the data type of the new value and generate a proper ValidationEvent when the new value has the wrong data type.

Users can extend this class to create their own ValidationObject to provide additional information that are needed by Validator. For example, TableValidationObject extends ValidationObject to add row and column information in a table.

ValidationEvent

The ValidationEvent is the result from the Validator. The ValidationEvent extends javafx.event.Event so it can be fired as normal just like any other JavaFX events. We use JavaFX event to deliver the validation result so that it can be handled easily just like any other JavaFX events[1]. To get notified of a ValidationEvent, you can addEventFilter or addEventHandler at the validating node or its ancestors. The latter case is very useful because you can listen to all ValidationEvents in a form by add event handler to the form only.

By default, we will add our own EventHandler using addEventFilter. We will use it to display the validation results.

No matter which level of validation it is, all the validator callback will return ValidationEvent. ValidationEvent contains five fields.

[1] Please refer the JavaFX event handling at http://docs.oracle.com/javafx/2/events/processing.htm.

```
private EventType _resultType;

private FailBehavior _failBehavior;

private int _id;

private Object _proposedValue;

private String _message;
```

There are five EventTypes for ValidationEvents.

Event	Ignorable	Description
VALIDATION_OK	Yes	This event occurs on the validation is okay, which means no validation errors were found.
VALIDATION_INFO	Yes	This event occurs on the validation has information
VALIDATION_WARNING	No	This event occurs on the validation has warning
VALIDATION_ERROR	No	This event occurs on the validation has error
VALIDATION_UNKNOWN	Yes	This event occurs on the validation status is unknown. This event can be used to clear the previous validation status. It is better than using OK because OK means there is no validation error. Using UNKNOWN means there might be a validation warning error but we just don't know at the moment, so please clear the validation status for now until further notice.

There are a few more fields in the event. For ignorable events, the only other field could be used is the message. The event handler can choose to display the message to users.

For non-ignorable events, there are failBehavior, proposedValue and message that can be set.

There are four types of failBehaviors.

```
public enum FailBehavior {
```

```
    /**
     * When validation fails, reverts back to the previous valid value. In
     * case of the table cell editing, revert and stop cell editing as
     * normal.
     */
    REVERT,
    /**
     * When validation fails, it will self-correct to the getProposedValue()
     * and stop cell editing as normal in case of table cell.
     */
    SELF_CORRECT,
    /**
     * When validation fails, do not clear the invalid value (and in the
     * case of table, do not stop cell editing) and wait for user to enter a
     * valid value or press ESCAPE to cancel the editing.
     */
    PERSIST,
    /**
     * This is used if you want to display the message but still want to
     * commit the value as normal.
     */
    IGNORE,
}
```

If you don't specify a failBehavior, it would be IGNORE for ignorable events and PERSIST for non-ignorable event. The proposedValue is only used when the failBehavior is SELF_CORRECT thus if you provided a proposedValue, the failBehavior is automatically set to SELF_CORRECT.

All events have an id field. We didn't use it anywhere in our code. Many applications have a database for error codes. You can use the id to consistently error coding all the validation errors. You can predefine all the ValidationEvents as constants and use them when needed.

For example, Windows has system error codes up to 15999. The first three error code are the following. The right column shows what it would look like if you predefined all of them using ValidationEvents.

Windows Error Codes	**Predefined ValidationEvents if were defined in JideFX**
ERROR_SUCCESS **0 (0x0)** **The operation completed successfully.**	ERROR_SUCCESS = new ValidationEvent(VALIDATION_OK, 0, "The operation completed successfully.");
ERROR_INVALID_FUNCTION **1 (0x1)** **Incorrect function.**	ERROR_INVALID_FUNCTION = new ValidationEvent(VALIDATION_ERROR, 1, "Incorrect function.");
ERROR_FILE_NOT_FOUND **2 (0x2)** **The system cannot find the file specified.**	ERROR_FILE_NOT_FOUND = new ValidationEvent(VALIDATION_ERROR, 2, "The system cannot find the file specified.");

and so on ……

Install Validator

Now we have the Validator. Generally speaking, in order to validate something, the Validator must be called when the value is changed. That means we need to install the Validator to a node and listen to the value change of the node. When a change occurred, we will call the Validator to generate a ValidationEvent and fire it. Here is the example that validates an email address field.

```
/* EXAMPLE ONLY. DO NOT USE THIS CODE IN YOUR REAL APPLICATION */
emailField.textProperty().addListener(new ChangeListener<String>() {
    @Override
    public void changed(ObservableValue<? extends String> observable, String
oldValue, String newValue) {
        Validator validator = new SimpleValidator(EmailValidator.getInstance());
        emailField.fireEvent(validator.call(new ValidationObject(emailField,
oldValue, newValue)));
    }
});
```

The code above uses a class called SimpleValidator. SimpleValidator uses a class called EmailValidator under Routines package in the Apache commons validation project[2] to do the validation. Please refer to SimpleValidator.java source code in the demo to see how it works.

Obviously, the code above is still too complex. To simplify it, we introduced ValidationUtils. The code can be simplified to one line as below.

```
ValidationUtils.install(field, new SimpleValidator(EmailValidator.getInstance()));
```

There are different situations that you want the validation to be triggered. We categorized those situations and captured them in the enum ValidationMode. There are three modes. See below.

```
/**
 * <code>ValidationMode</code> defines when the validation will be triggered.
 */
public enum ValidationMode {
    /**
     * Validation will be triggered when user types.
     */
    ON_FLY,

    /**
     * Validation will be triggered when the field loses focus.
     */
    ON_FOCUS_LOST,
```

[2] The project is at http://commons.apache.org/proper/commons-validator/. We included its jar in the demo for demoing purpose but the JideFX Validation product doesn't depend on it. Feel free to include it in your own project if you want to use it.

```
    /**
     * Validation will be triggered when called explicitly.
     */
    ON_DEMAND
}
```

The ON_FLY mode is basically the same as the example code above where we listen to the textProperty() of the TextField. The textProperty() could be some other ObservableValues, such as the selectedItemProperty() of a SelectionModel in the ComboBox and ChoiceBox.

The ON_FOCUS_LOST is obviously to listen to the focusProperty() of any given node. When the focus loses, the validator will be triggered.

The ON_DEMAND mode is basically the manual mode. This mode can be used to validate a form. For example, you want the form to be validated before submitting it. If the ON_DEMAND mode is used, the only way to trigger these kind of validators is to call ValidationUtils. validateOnDemand(Node targetRegionOrNode) method. The targetRegionOrNode could be the node to be validated, or the region which contains the nodes to be validated. When this method is called, all children nodes that have an ON_DEMAND Validator installed will be called to validate. It will not trigger any other ON_FLY or ON_FOCUS_LOST validators

You can install different validators in different modes. They won't be conflicting with each other. For the same mode, only one validator is allowed. Installing another validator on the same mode will remove the previous validator.

Once installed, you can uninstall the validator using one of the uninstall methods on ValidationUtils.

By default, we will create an EventHandler to display the validation results. If you want to add more in addition to what we displayed, you can do it by adding your own EventHandler for the ValidationEvent. If you are using ValidationUtils.install method which takes an EventHandler, you can also install your own EventHandler to replace our default one. In this case, the validation result will not be displayed except what you did in your own EventHandler.

Examples

See below for a form that is created using MigPane.

```
MigPane pane = new MigPane(new LC().minWidth("450px").minHeight("280px").insets("20
10 10 10"), new AC().index(0).align("right").gap("10px").index(1).grow(), new
AC().gap("5px"));

pane.add(new Label("Your Email"));
pane.add(new TextField(), new CC().width("250px").wrap());
pane.add(new Label("Confirm Email"));
pane.add(new TextField(), new CC().width("250px").wrap());
pane.add(new Label("Country"));
pane.add(new ChoiceBox<>(DemoData.createCountryList()), new
CC().width("250px").wrap());
pane.add(new Label("Zip Code"));
pane.add(new TextField(), new CC().maxWidth("80px").wrap());
pane.add(new Label("Password"));
pane.add(new PasswordField(), new CC().width("250px").wrap());
pane.add(new Label("Confirm Password"));
pane.add(new PasswordField(), new CC().width("250px").wrap());
```

```
pane.add(new Label(""));
pane.add(new CheckBox("Yes, I agree to the term of use"), new CC().wrap("20px"));
pane.add(new Label(""));
pane.add(new Button("Sign Up"), new CC().wrap());

return new GroupBox("Create a new account", pane);
```

Here is what the form it looks like.

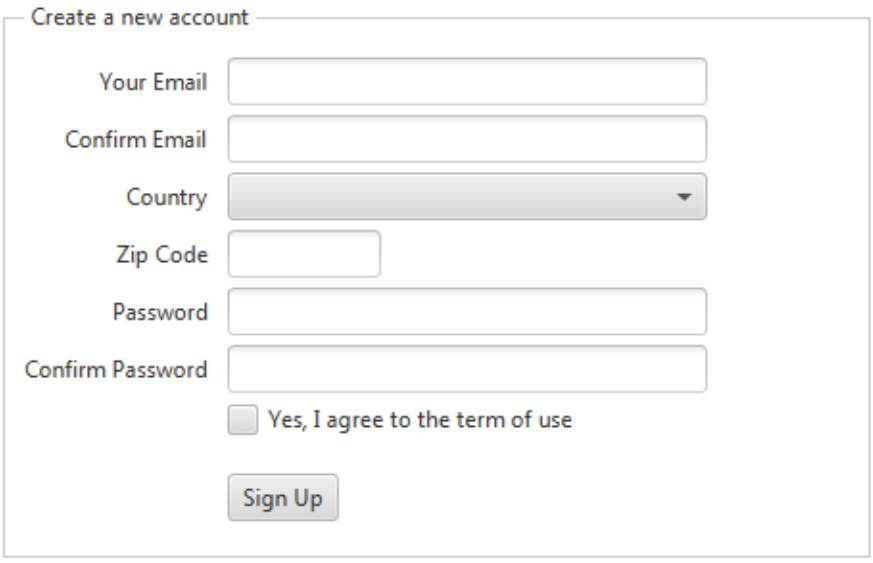

Figure 1 A form without validation

Here is what the same form looks like with all kinds of validation decorations.

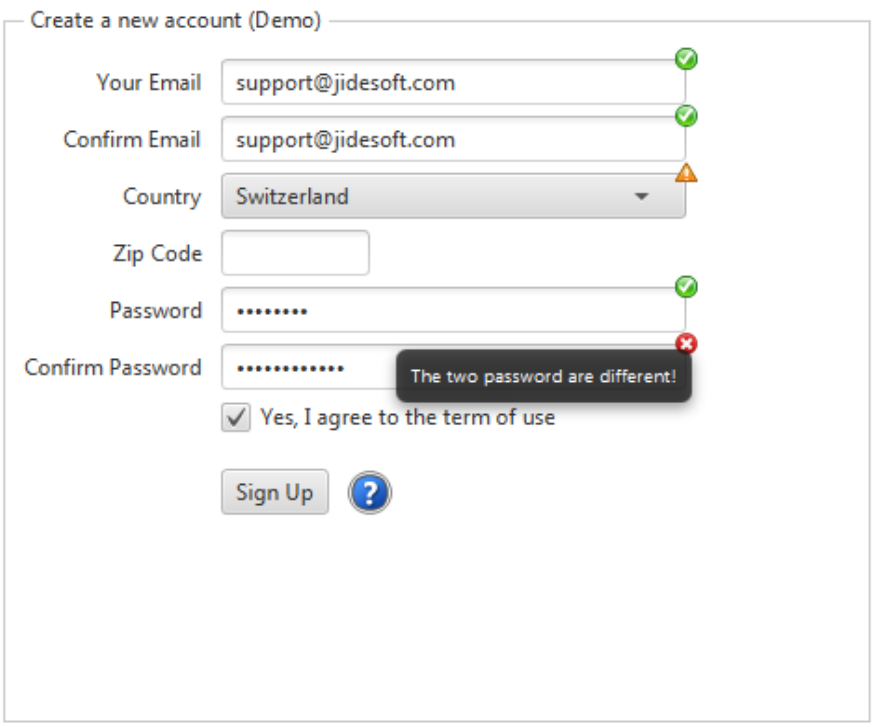

Figure 2 A form after validation