IMPLEMENT FIELDS VALIDATION

FORM FACTORY V2

abcValidation

For this tutorial I will use **form-factory-snippets-extension** module, you may use a module previously created or create a new module with dependency on Form Factory.

WARNING: If you copy-paste code from this document you will have to make sure that all characters are properly transferred. Pay close attention to hyphens. Also, if you create your views using Studio make sure you remove all taglibs that get generated by default.

IMPORTANT: When working with JavaScript it may be necessary to flush browser cache after changes to Javascript files. Most modern browsers provide an add-on tool to perform cache flushes. To get such a tool for your browser you should search available add-ons with a search criteria such as “clear cache” or “empty cache”, select appropriate add-on and follow given instructions.

1) Create a definition

The first thing you need to do when creating a validation is to create a definition. Here’s a look at a standard Form Factory validation definition:

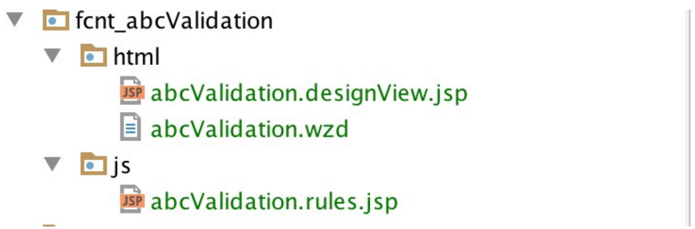
```
[fcnt:requiredValidation] > jnt:content, fcmix:validation, mix:title, jmix:droppableContent, jmix:hiddenType
```

Consequently, your **abcValidation**’s definition should look like this:

```
[fcnt:abcValidation] > jnt:content, fcmix:validation, mix:title, jmix:droppableContent, jmix:hiddenType
```

2) Create views

Once you have a definition create **fcnt_abcValidation** folder and place inside of it **html** and **js** folders. Inside **html** folder create *abcValidation.designView.jsp* and *abcValidation.wzd* files. Create *abcValidation.rules.jsp* inside the **js** folder. Don’t add any content to the files just yet. Once done you should see the following structure:



```
fcnt_abcValidation
  html
    abcValidation.designView.jsp
    abcValidation.wzd
  js
    abcValidation.rules.jsp
```

(change folder name to .rule.jsp)

The *designView* is is the view that you will see in the side panel when you setup your validation. The wzd file is the file that specifies properties for the validation. The *validationRules* view contains Javascript code that verifies the rule you want to enforce for this validation. In this case you want the input to be equal to “abc”.

3) Create a wizard (abcValidation.wzd)

As mentioned above wzd file is what defines your validation. Validation wizards come with two mandatory properties: label and types. Label is the name of your validation and types is the class of your validation. There are several classes of validations: regex, number, date, equal etc. You can easylly create you own class but remember that each input that we defined or you define must specify zero or more classes of supported validations. Since you would need to create an input or extend an existing one in order to test your own validation class, specify regex as validation type for **abcValidation**. This way it will be applicable to all inputs that support regex validations. Here's how the wizard file should look:

```
package fcnt_abcValidation.html

validation {
  label "Abc"
  types "regex"
  propertiesI18n {
      message "The field must be equal to abc"
  }
}
```

Note that **propertiesI18n** object contains a translatable message property, which may be different depending on user's locale. Also, note that validation wizards have support for **properties** objects which may contain any property you want and which will be available for you upon execution of the rule.

4) Create design view (abcValidation.designView.jsp)

In design view you need to add a field for editing of the translatable message property (as a side note you should know that you can test for when you are in translate mode with the **inTranslateMode** flag directly in design view). You may also want to add validation to the field to make sure that users cannot save your form with no validation message. Here's how the view should look:

```
<div class="row">
  <div class="col-md-12 form-group"
       ng-class="{'has-error': !vc.isFieldValid('message')}">
      <label class="control-label"
             message-key="ff.label.message"></label>
      <input type="text"
             class="form-control"
             ng-model="input.validations[validationName].message"
             ff-field-value-validation="any" field-id="message" case-name="{{validationName}}">
      <span class="help-block" ng-show="!vc.isFieldValid('message')">
          <span message-key="ff.label.required"></span>
      </span>
  </div>
</div>
```

Note that **ff-field-value-validation** directive is used for validating the message field. In this case it should also receive two additional parameters: **field-id** and **case-name**. Observe how the name of the message property is used to avoid potential validation conflicts in cases when you have multiple fields that need to be validated. Aside from this caveat the view is a simple composition of html elements.

You could easylly add a custom resource bundle keys here as well, but for the purpose of brevity we'll stick with the standard **ff.label.message** label.

5) Create validation function (abcValidation.rule.jsp)

The validation function will execute every time changes are detected in the model of the input (the function will be added to the input's validations array provided by AngularJS). The return value of the function will determine whether the field is valid or not so it must return a boolean. By design you can access your rule (with all its properties), scope, element and controllers directly from the validation function for any manipulations you may need to do. Here's what the function should look like in this case:

```
<%@ page contentType="text/javascript" %>
/**
```

```
* Applies the supplied regex to the entered value.
* Receives the rule definition and its scope.
* scope.$parent will contain the form to enable watch and access to other fields in the same step
* ctrls will be an array of controllers, first the ffFormController and then the ngModel of the      *
associated input
*
* @param {object} rule current rule definition
* @param {object} scope current input scope
* @param {object} el current html element
* @param {array} attr attributes of the html element
* @param {array} ctrls ctrls[0] = FFFormController ctrls[1]=ngModel
*/

ff.validationRules.abc = function (rule, scope, el, attr, ctrls) {
  return function(modelValue, viewValue){
      if(_.isUndefined(viewValue) || _.isEmpty(viewValue)){
          return false;
      }
      return viewValue === 'abc';
  }

};
```

All validation must be attached to the **ff.validationRules** object and must be referenced by their name. You can have any logic you want in this function as long as it does what you want it to do.

6) Add labels to resource bundles

In your resource bundles place the following label:

```
ff.label.validation.abc=Abc validation
```

By convention all your validation labels should start with *ff.label.validation* followed by dasherized validation name.

Now deploy your module and add it to your site and you should see your newly created validation available for inputs supporting regex validation.