Custom Settings Guide for Form Factory

Form Factory Settings have been refactored to allow better fine grained control over the management of the forms. A module can now provide unique settings functionality by extending ***fcmix:formSetting***.

This guide will walk you through the steps required to create a custom setting component.

In order to have form factory recognize your component, a definition extending ***fcmix:formSetting*** is required.

```
[fcnt:myCustomSetting] > jnt:content, fcmix:formSetting, mix:title, jmix:droppableContent,
jmix:hiddenType
```

Create a title for this component, that will be displayed in the settings, by creating a resource bundle key **ff.label.formsetting.*your-component-definition***. For this example it would be :

ff.label.formsetting.*my-custom-setting*

Now that the definition is created, It's time to setup the structure of the component.

Custom Settings follows a similar structure to inputs, validations, actions etc :

- myCustomSetting.wzd
  - Contains the properties required for this component.

```
formsetting {
   label "Mailchimp Settings"
   types "formsetting"
   wizard "<ff-mailchimp-settings></ff-mailchimp-settings>"
}
```

  - *Label* is used to create this component by using the value provided, split by the spaces and replaced by dashes. I.e : *My Custom Setting -> my-custom-setting*

- *Wizard* represents the directive tag that will be compiled
- *Types* is inherited from the generic serialization and deserialization schema and can be used to group settings.

- myCustomSetting.directive.jsp
  - Custom Directive that will be retrieved and compiled by Form Factory.
  - Use the ***$onInit*** function to do any initialization procedures.

```
<%@ page contentType="text/javascript" %>
<%@ taglib prefix="formfactory" uri="http://www.jahia.org/formfactory/functions" %>
<%--@elvariable id="renderContext" type="org.jahia.services.render.RenderContext"--%>

(function(){
   var myCustomSetting = function(ffTemplateResolver) {
       return {
           restrict: 'E',
           scope: {},
           templateUrl: function(el, attrs) {
               return
ffTemplateResolver.resolveTemplatePath('${formfactory:addFormFactoryModulePath('/form-factory-f
ormsettings/my-custom-setting', renderContext)}', attrs.viewType);
           },
           controller: MyCustomSettingController,
           controllerAs: 'mcsc',
           bindToController: true,
           link: linkFunc
       };

       function linkFunc(scope, el, attr, ctrl) {}
   };
   angular
       .module('formFactory')
       .directive('ffMyCustomSetting', ['ffTemplateResolver', myCustomSetting]);

   MyCustomSettingController.$inject = [];

   function MyCustomSettingController () {
       var mcsc = this;

       mcsc.$onInit = function() {
           //Do initialization here
           mcsc.settingOneEnabled = false;
           mcsc.settingTwoEnabled = true;
           mcsc.settingThreeEnabled = true;
       }
   }
})();
```

- myCustomSetting.jsp
  - The view used by the custom directive to display and allow configuration of the custom settings

```

```

```
<div class="row">
    <div class="col-sm-12">
        <span>Setting One</span>
        <div style="float:right;">
            <switch ng-model="mcsc.settingOneEnabled"></switch>
        </div>
    </div>
    <div class="col-sm-12">
        <span>Setting Two</span>
        <div style="float:right;">
            <switch ng-model="mcsc.settingTwoEnabled"></switch>
        </div>
    </div>
    <div class="col-sm-12">
        <span>Setting Three</span>
        <div style="float:right;">
            <switch ng-model="mcsc.settingThreeEnabled"></switch>
        </div>
    </div>
</div>
<hr/>
```

When all the steps have been followed correctly and the module containing the new settings component has been successfully deployed, the custom settings component will be visible on the Form Factory Settings page.