Adding a Complex Field to a Component

In This Topic

In Adding an Input Field to a Component, it is explained how to add simple input fields to the several component types that XperienCentral supports. This topic explains how to add complex fields to a component.

Editing Multi-value Property with Multi-selection Fields

This topic discusses how to add a field to a panel component that binds zero or more values to one single property using a multiple selection field in the User Interface. As an example we define a "hobbies" property in the form backing object which must be editable by a multiple selection field containing 9 fixed possible values.

The first step is to define the property in the form backing object:

```
private String[] myHobbies = null;

public String[] getHobbies() {
return myHobbies;
}

public void setHobbies(String[] hobbies) {
        myHobbies = hobbies;
}
```

We need to define what hobbies are available to select from. We define these statically also in the Form backing object class:

```
public static final String[] HOBBIES = new String[] {"Candlemaking", "Games", "Gardening", "Collecting",
"Juggling", "Photography", "Pottery", "Scrapbooking", "Writing"};
```

The next step is to modify the JSP in order to display the checkboxes. Since the hobbies contained by the form backing object will not be accessible from the JSP and we did not define a getter for it, we add this to the reference data in the panel controller.

```
public Map<String, Object> referenceData(HttpServletRequest request, Object command, Errors errors) throws
Exception {
        Map<String, Object> data = super.referenceData(request, command, errors);
        data.put("allhobbies", CustomTabFBO.HOBBIES);
        return data;
}
```

The only thing left to do is to add the multiple selection fields in the edit JSP and bind them onto the "hobbies" property. Since there is no standard JSP tag for doing this, we bind the value from the selection field manually onto this property as follows:

```
<spring:bind path="command.hobbies">
        <select name="${status.expression}" multiple>
                <c:forEach var="hobby" items="${allhobbies}">
                        <option value="${hobby}" <c:if test="${wmfn:contains(hobby, command.hobbies)}">
selected</c:if>> ${hobby}</option>
                </c:forEach>
        </select>
</spring:bind>
```

Because we use the `wmfn` tag library, we must explicitly import that library in the JSP:

```
<%@taglib uri="http://www.gx.nl/taglib/functions" prefix="wmfn" %>
```

Back to Top

Editing Multi-value Property with Multiple Checkboxes

This part discusses how to add a field to a panel component that binds zero or more values to one single property using a fixed set of checkboxes in the user Interface. As an example, we define a "hobbies" property in the form backing object which must be editable by checkboxes for each of the 10 fixed possible values.

Earlier in this topic we added the "hobbies" property to the form backing object and wrote all hobbies to the `allhobbies` attribute in the reference data. The only difference is that we now want the "hobbies" property to use multiple checkboxes instead of one multiple selection field. Again, there is no standard JSP tag for doing this, so we bind the value from the checkboxes manually onto this property as follows:

```
<spring:bind path="command.hobbies">
        <c:forEach var="hobby" items="${allhobbies}">
                <input type="hidden" name="_${status.expression}" value="visible" />
                <input type="checkbox" name="${status.expression}" value="${hobby}"
                        <c:if test="${wmfn:contains(hobby, command.hobbies)}"> checked="checked"
                        </c:if>
                />
                 ${hobby}<br/>
        </c:forEach>
</spring:bind>
```

As a result on each submit `setHobbies` will be invoked with only those hobbies that were selected.

Back to Top

Editing Multi-value Property with Multi-selection Field and Multiple Checkboxes

Earlier parts of this topic described the approach for using assign and delete. In this case a multiple selection field is used to assign new values to the property. Below this selection field, the current values of the property are displayed and can removed by enabling the "delete" checkbox displayed behind it.

For the implementation of this approach, the same hobbies property of the form backing object and `allhobbies` attribute in the reference data are used (as described in the previously). The only thing that must be changed is the JSP and the way we bind the posted form values onto the "hobbies" property.

The first step is to create the multiple selection field from which values can be added to the "hobbies" property. Also for this particular case, no standard JSP tag exists and we will have to do the spring binding manually.

The values of the option fields in the selection field can simply be bound onto the "hobbies" property directly. We must take care to filter out the values that have already been assigned. The code snippet below shows how to bind the values from the multiple selection field onto the "hobbies" property.

```
<spring:bind path="command.hobbies">
        <select name="${status.expression}" multiple>
                <c:forEach var="hobby" items="${allhobbies}">
                        <c:if test="${!wmfn:contains(hobby, command.hobbies)}">
                                <option value="${hobby}"> ${hobby}</option>
                         </c:if>
                 </c:forEach>
        </select>
        <wmedit:button type="submit" key="add"/>
</spring:bind>
```


The HTML is stripped from this code snippet since it is not the intention of this topic to discuss how to properly render the "selection" field and corresponding "Submit" button.

The next step is to display the current values of the property and add the Delete checkboxes to delete those values. This is a slightly more complicated process. The Spring binding mechanism binds values posted from checkboxes only if they are enabled. This works fine if we want to use the checkbox to assign values, but it is less suitable for this case because we want the value to be removed if the checkbox is enabled. The way Spring binding deals with checkboxes must be inverted, that is, values should only be bound onto the "hobbies" property if the checkbox is not checked.

In order to do so, an additional hidden input field has to be added that contains the value to be set on the "hobbies" property if the delete checkbox is not selected:

```
<input type="hidden" name="${status.expression}" value="${hobby}"/>
```

Subsequently, the `onchange` method of the checkbox must be overridden in order to disable this hidden input field when the checkbox is selected. This prevents the field from being posted and thus bound to the "hobbies" property by Spring. The following JavaScript function is added to support this conditional disabling of the hidden input field:

```
<script type="text/javascript" language="javascript">
        function disableCheckbox(id) {
                var checkbox = document.getElementById(id);
                        if (checkbox.checked) {
                                checkbox.disabled=false;
                        }
                        else {
                                checkbox.disabled=true;
                        }
                }
</script>
```

Finally, the code snippet below creates the hidden input fields for the current values of the "hobbies" property and disables each of these values when the corresponding delete checkboxes are selected.

```
<spring:bind path="command.hobbies">
        <c:forEach var="hobby" items="${command.hobbies}" varStatus="loop">
                <input type="hidden" id="${status.expression}_${loop.index}" name="${status.expression}"
value="${hobby}"/>
                ${hobby}
                <input type="checkbox"
                        onchange="disableCheckbox('${status.expression}_${loop.index}');"/>
                <fmt:message key="delete " /><br/>
        </c:forEach>
</spring:bind>
```

As a result, only those values are bound by Spring onto the "hobbies" property which are either added by the multiple selection field or were already assigned and the corresponding delete checkbox was not checked.

This approach does not require any specific implementation of the controller or form backing object.

Back to Top

Editing Multi-value Property with Single Selection Field and Multiple Checkboxes

A slightly different variant from the approach described earlier in this topic is to use a single selection field for assigning new values to the "hobbies" property instead of a multiple selection field. Although this may seem to be a trivial case of simply removing the "multiple" from the selection form field, it is more complex than that.

One < 'dummy' value > Select a hobby" is added to the selection field and this is exactly where the problem occurs. Even an empty or null value of an option will be posted by the HTML form and bound onto the "hobbies" property by Spring. This results in an additional undesired empty value in the "hobbies" property each time the form is submitted with no real hobby selected.

To provide a workaround for this, a function must be defined that is invoked just before submitting the form. This function must check whether the value of this selection field is empty and, if so, disable it so that the field value is not posted. In case of a panel, the JavaScript function would look like this:

```
<script type="text/javascript" language="javascript">
        function saveForm() {
                var select = document.getElementById('empty_select');
                if (select.value == '') {
                        select.disabled=true;
                }
                else {
                        select.disabled=false;
                }
                formobj = document.getElementById("saveform");
                formobj.submit();
        }
```

The code snippet above assumes the empty dummy field of our selection field has the ID `empty_select`. The code snippet below shows how to render the single selection field:

```
<select name="${status.expression}">
        <option id="empty_select" value="">&gt; Select a hobby</option>
                <c:forEach var="hobby" items="${allhobbies}">
                        <c:if test="${!wmfn:contains(hobby, command.hobbies)}">
                                <option value="${hobby}"> ${hobby}</option>
                        </c:if>
                </c:forEach>
</select>
```

Back to Top