# NetUI Repeating Form Control Tags

## Table of contents

## 1. Introduction

The three NetUI group tags offer a repeating mode that increases the control the developer has over their output. For the `<netui:checkBoxGroup>` and the `<netui:radioButtonGroup>` tags the control extends to the layout of the resulting markup. For the `<netui:select>` tag the developer can control the order of the options and the `value` and `label` when an `optionsDataSource` is being used.

This is an advanced tag topic. For more information on the basic operations of these tags see the NetUI Form Control Tags topic.

## 2. CheckBoxGroup and RadioButtonGroup

This section demonstrates how to control the markup generated by both the `<netui:checkBoxGroup>` and `<netui:radioButtonGroup>` tags. Both tags have an attribute `repeater` which, when set to `true`, turns the tag into a repeating tag. The tag will then loop over each element found in the `optionsDataSource` and evaluate the tag's body against the item. Before the body is evaluated, the implicit object `container` is setup. For more information see the data binding container implicit object topic.

This gives the developer more ability to control the layout and style associated with the options when they are using an `optionsDataSource`. Below are examples of both the `<netui:checkBoxGroup>` and the `<netui:radioButtonGroup>` tags.

### 2.1. CheckBoxGroup

This example will create a horizontal layout for a set of checkboxes. Individual styles will be applied to the labels of the checkbox. The layout is done using an HTML table.

CheckBoxGroup with horizontal layout

In the JSP fragment below, the `<netui:checkBoxGroup>` tag is bound to an `optionsDataSource` in the page flow. The `optionsDataSource` is an array of a class that contains the style, label value and option value for each element to be displayed in the group. Notice that inside the body of the `checkBoxGroup` all the binding expressions start with `${container.item.XXX}`. The body will be repeated for each element in the `optionsDataSource`.

**Note:** The `dataSource` also directly binds to a page flow variable. Typically, the `dataSource` would bind to an `actionForm` variable.

```
<table>
```

```
<caption class="normalAttr">CheckBox Group</caption>
<tr><td>
<netui:checkBoxGroup dataSource="pageFlow.results"
optionsDataSource="${pageFlow.opts}"
            repeater="true" >
     <netui:span styleClass="${container.item.style}"
value="${container.item.name}" />
     <netui:checkBoxOption value="${container.item.optionValue}" /> 
</netui:checkBoxGroup>
</td></tr>
</table>
```

The following style information is found in the JSP. These styles affect the presentation of label output. This is done by setting the styles on the `<netui:span>` tags that act as the labels for the generated checkboxes.

```
<style type="text/css">
    .normalAttr {color: #cc0099;font-family:Verdana;
font-size:8pt;margin:0,0,0,0;}
    .normal {color: #cc9999;font-family:Verdana;
font-size:8pt;margin:0,0,0,0;}
    .normal2 {color: #00cc99;font-family:Verdana;
font-size:8pt;margin:0,0,0,0;}
    .normal3 {color: #99cc99;font-family:Verdana;
font-size:8pt;margin:0,0,0,0;}
</style>
```

Below is the page flow controller that supports the example above. In the `onCreate` method, we initialize the `optionsDataSource`. In addition, we provide a `java.lang.String[]` that will recieve the results of posting the form back. Finally, an inner class defines a Java bean that contains the information used by the options.

```
package repeating;

import org.apache.beehive.netui.pageflow.PageFlowController;
import org.apache.beehive.netui.pageflow.Forward;
import org.apache.beehive.netui.pageflow.annotations.Jpf;

@Jpf.Controller(
    simpleActions={
        @Jpf.SimpleAction(name="begin", path="index.jsp")
    }
)
public class Controller extends PageFlowController
{
    private Options[] opts;
    private String[] results;

    public Options[] getOpts()
    {
        return opts;
    }
```

```
public void setOpts(Options[] opts)
{
    this.opts = opts;
}

public String[] getResults()
{
    return results;
}

public void setResults(String[] resultsOne)
{
    this.results = resultsOne;
}

protected void onCreate()
{
    // initialize the opts
    opts = new Options[3];
    opts[0] = new Options("Option One","opt-1", "normal");
    opts[1] = new Options("Option Two","opt-2", "normal2");
    opts[2] = new Options("Option Three","opt-3", "normal3");
}

/**
 * @jpf:action
 * @jpf:forward name="index" path="Results.jsp"
 */
@Jpf.Action(
    forwards = {
        @Jpf.Forward(
            name = "index",
            path = "Results.jsp")
    })
protected Forward post()
{
    return new Forward("index");
}

public static class Options implements java.io.Serializable {
    private String _name;
    private String _optionValue;
    private String _style;

    public Options(String name, String value, String style) {
        _name = name;
        _optionValue = value;
        _style = style;
    }

    public void setName(String name) {
        _name = name;
    }
```

```
        public String getName() {
            return _name;
        }

        public void setOptionValue(String optionValue) {
            _optionValue = optionValue;
        }
        public String getOptionValue() {
            return _optionValue;
        }

        public void setStyle(String style) {
            _style = style;
        }
        public String getStyle() {
            return _style;
        }
    }
}
```

## 2.2. RadioButtonGroup

**Note:** This example is very similar to the previous example using the CheckBoxGroup. The only real difference is that the layout of the radio buttons is vertical. For a detailed discussion, you should read that example.

RadioButtonGroup with vertical layout

Below is a fragment of a JSP that will layout a RadioButtonGroup in a table vertically. All of the options have an individually applied style. Notice that the `repeater` attribute is set on the `<netui:radioButtonGroup>` tag to repeat over the items defined in the `optionsDataSource`.

```
<table width="200pt">
    <caption class="normalAttr">RadioButton Group</caption>
    <netui:radioButtonGroup dataSource="pageFlow.results"
optionsDataSource="${pageFlow.opts}" repeater="true">
        <tr align="center"><td align="right" width="25%">
            <netui:radioButtonOption value="${container.item.optionValue}"
/></td>
            <td align="left"><netui:span
styleClass="${container.item.style}" value="${container.item.name}" />
        </td></tr>
    </netui:radioButtonGroup>
</table>
```

This example can use the same page flow controller as the previous example with one simple modification. A radio button group will post back a single value. By modifying the `results` property on the page flow controller we convert if from a `String[]` to just a `String`.

```
private String results;
public String getResults()
{
    return results;
}
public void setResults(String resultsOne)
{
    this.results = resultsOne;
}
```

## 3. Select

Repeating in a `<netui:select>` is much different than repeating in either the `<netui:checkBoxGroup>` or `<netui:radioButtonGroup>`. There is no layout or style information applied to the individual options because typically they are displayed as a single select box control inside the browser.

Repeating in the `<netui:select>` enables the developer to control the order of the options and to control the HTML `<option>` element rendered for each type of option. A `<netui:select>` tag actually creates it's options by iterating over multiple sets of data. For each unique `value` found, an HTML `<option>` element is rendered. The following table describes each source of data that can be used to create the output options.

| Stage Name | Source property | Description |
|---|---|---|
| `option` | `optionsDataSource` | This is an array of some class. Each element of the array will be output as an option. Typically the class can just be a `String` or it may be a Java bean. |
| `default` | `defaultValue` | This is a single value that will be output as an option. This value is always a `String` |
| `dataSource` | `dataSource` | When the select is rendered, if the variable bound to contains data, it will be added to the options. This value may be a single `String` or an array of `Strings`. |
| `null` | `nullableOptionText` | This is a special value that indicates a null value. You must set the `nullable` attribute to `true` to enable this. |

## 3.1. Select Example

The following example is a complex Select control `<netui:select>` that controls both the order and how each of the HTML `<option>` elements are rendered. The source for the JSP is followed by the controller.

**Note:** The select box posts its value back to the page flow. The typical use would be to post a value back through a `FormData` subclass.

**index.jsp**

```
<%@ page language="java" contentType="text/html;charset=UTF-8"%>
<%@ taglib uri="http://beehive.apache.org/netui/tags-html-1.0"
prefix="netui"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<netui:html>
    <head>
        <title>Order Repeating Select</title>
        <style type="text/css">
            .normalAttr {color: #cc0099;font-family:Verdana;
font-size:8pt;margin:0,0,0,0;}
        </style>
    </head>
    <netui:body>
        <h4>Order Repeating Select</h4>
        <p style="color:green">This example demonstrates using a repeating
select.
        </p>
        <netui:form action="post">
            <netui:select dataSource="pageFlow.results"
defaultValue="default Value"
                    optionsDataSource="${pageFlow.opts}" repeater="true"
                    repeatingOrder="null, default, option" nullable="true">
                <c:if test="${container.metadata.optionStage}">
                    <netui:selectOption repeatingType="option"
                            value="${container.item.optionValue}"
styleClass="normalAttr">
                        ${container.item.name}
                    </netui:selectOption>
                </c:if>
                <c:if test="${container.metadata.defaultStage}">
                    <netui:selectOption repeatingType="default"
                            value="${container.item}"
styleClass="normalAttr">
                        ${container.item}
                    </netui:selectOption>
                </c:if>
                <netui:selectOption repeatingType="null" value="null-opt"
                        styleClass="normalAttr">
                    [Nothing]
                </netui:selectOption>
```

```
            </netui:select>
            <p><netui:button>Submit</netui:button></p>
        </netui:form>
    </netui:body>
</netui:html>
```

The `<netui:select>` defines the `dataSource`, `defaultValue`, and an `optionsDataSource`. These may all be used to add values to the displayed options. In addition, the "null" option is also set because the `nullable` attribute is set to `true`. These are the four sources of options. The `repeater` attribute is set to `true`. For this reason, the body of the select will be repeated for each option to be rendered. Finally, the order of the options is specified using the `repeatingOrder` attribute. This is a comma separated list of the stage names from the table above. In this case, the order of the option will be "null", "default, "option" and the "dataSource" will not be displayed.

```
<netui:select dataSource="pageFlow.results" defaultValue="default Value"
    optionsDataSource="${pageFlow.opts}" repeater="true"
    repeatingOrder="null, default, option" nullable="true">
```

The example uses JSTL logic tags to control the evaluation of expressions for each of the stages. To include the JSTL core use the following tag library declaration in the JSP.

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
```

The body of the `<netui:select>` will be evaluted once for each option to be rendered. In the body, we use the JSTL `if` tag to control the evalutation of each option. This is done to prevent expression evaluation errors when the JSP EL expressions are being evalated for each case. In the `container` implicit object, the current stage name is exposed within the `metadata`. There is a boolean property for each stage that is true only when options for that stage are being generated. These boolean properties are `optionStage`, `defaultStage`, `dataSourceStage`, and `nullStage`.

```
<c:if test="${container.metadata.optionStage}">
    <netui:selectOption repeatingType="option"
            value="${container.item.optionValue}" styleClass="normalAttr">
        ${container.item.name}
    </netui:selectOption>
</c:if>
<c:if test="${container.metadata.defaultStage}">
    <netui:selectOption repeatingType="default"
            value="${container.item}" styleClass="normalAttr">
        ${container.item}
    </netui:selectOption>
</c:if>
<netui:selectOption repeatingType="null" value="null-opt"
        styleClass="normalAttr">
    [Nothing]
</netui:selectOption>
```

The code within the following JSTL `if` tag will execute only when the `optionsDataSource` is being evaluated. This allows the expressions used to set the properties to be evaluated against a known type and allows other stages to use different types.

```
<c:if test="${container.metadata.optionStage}">
    ...
</c:if>
```

Within the JSTL `if` tag, there is a `<netui:selectOption>`. The option must identify which stage is being run; this is done by setting the `repeatingType` attribute to the stage name. The rest of the values are set using the `container` implicit object. This is set to each item defined for that stage. In this case, it will iterate over the `optionsDataSource`.

```
<netui:selectOption repeatingType="option">
        value="${container.item.optionValue}" styleClass="normalAttr">
    ${container.item.name}
</netui:selectOption>
```

The option for the nullable value does not contain any expressions and therefore does not appear inside a JSTL `if` tag. It will only render an option when the stage becomes `null` and will be ignored in for all other options.

```
<netui:selectOption repeatingType="null" value="null-opt"
        styleClass="normalAttr">
    [Nothing]
</netui:selectOption>
```

The page flow controller is a simple controller that contains the properties exposing the options and a property that will be set when the form is posted. The options are built in the `onCreate` method. For the `optionsDataSource` each option is defined through an instance of the `Options` class.

### Controller.jpf

```
package select;

import org.apache.beehive.netui.pageflow.PageFlowController;
import org.apache.beehive.netui.pageflow.Forward;
import org.apache.beehive.netui.pageflow.annotations.Jpf;

@Jpf.Controller(
    simpleActions={
        @Jpf.SimpleAction(name="begin", path="index.jsp")
    }
)
public class Controller extends PageFlowController
{
    private Options[] opts;
    private String results;
```

```
public Options[] getOpts()
{
    return opts;
}

public void setOpts(Options[] opts)
{
    this.opts = opts;
}

public String getResults()
{
    return results;
}
public void setResults(String value)
{
    this.results = value;
}

protected void onCreate()
{
    // initialize the opts
    opts = new Options[3];
    opts[0] = new Options("Option One","opt-1");
    opts[1] = new Options("Option Two","opt-2");
    opts[2] = new Options("Option Three","opt-3");
}

@Jpf.Action(
    forwards = {
        @Jpf.Forward(
            name = "index",
            path = "Results.jsp")
    })
protected Forward post()
{
    return new Forward("index");
}

public static class Options implements java.io.Serializable {
    private String _name;
    private String _optionValue;

    public Options(String name, String value) {
        _name = name;
        _optionValue = value;
    }

    public void setName(String name) {
        _name = name;
    }
    public String getName() {
        return _name;
```

```
        }

        public void setOptionValue(String optionValue) {
            _optionValue = optionValue;
        }
        public String getOptionValue() {
            return _optionValue;
        }
    }
}
```