NetUI Form Control Tags

Table of contents

1. Introduction

The following table summarizes the basic NetUI tags which render as form controls. There are basic types, simple controls and group controls. The simple controls map directly to a single HTML element. The group controls map to more than one HTML element, for example the `<select>` and `<option>` elements represent a select box.

The table summarizes the natural data type the control is usually bound to. Typically, a form will post to an action that receives a subclass of `FormData` which is populated with the values from the form by the framework before the action is called.

NetUI Form Control	Binding Data Type
<netui:anchor>	None
<netui:button>	None
<netui:checkBox>	`boolean` or `java.lang.Boolean` or `java.lang.String`
<netui:checkBoxGroup>	`java.lang.String[]`
<netui:checkBoxOption>	None, See the `checkBoxGroup` tag
<netui:fileUpload>	org.apache.struts.upload.FormFile
<netui:form>	None
<netui:hidden>	`java.lang.String` or `java.lang.String[]`
<netui:imageButton>	None
<netui:radioButtonGroup>	`java.lang.String`
<netui:radioButtonOption>	None, See the `radioButtonGroup` tag
<netui:select>	`java.lang.String` or `java.lang.String[]`
<netui:selectOption>	None, See the `select` tag
<netui:textArea>	`java.lang.String[]`
<netui:textBox>	`java.lang.String[]`

2. NetUI Form Controls Example

All of the code fragments in this topic come from a single example. The example source can be found in the Form Controls Example.

3. Anchor

Though not technically a form control, the `<netui:anchor>` can be used to submit a form. It does this through a JavaScript function that is dynamically added to the page. In the following example the anchor is found inside of the `<netui:form>`, which is a requirement for using a anchor to submit a form. In order to submit a form, the `formSubmit` attribute is set to `true`. In addition, there is no requirement to set an `action` or `href` on the anchor. It is possible to submit a form when the anchor appears outside of the form. This requires setting the `getJavaScript` attribute on the `<netui:form>` tag. Then you need to add the following code to the anchor `onClick="anchor_submit_form(name-of-form, action);return false;"`.

The anchor does not bind to a `dataSource` of any type.

```
<netui:form>
   ...
   <netui:anchor  formSubmit="true">Submit Form Through a
Link</netui:anchor>
</netui:form>
```

There is a bit of generated JavaScript in the rendered page that will submit the form. See Tags Support for JavaScript for details on how the framework generated JavaScript is supported. Below is the rendered HTML `<a>` tag, which calls the framework `generated anchor_submit_form` function.

```
<a href="/dev/formControls/submit.do"
    onclick="anchor_submit_form('Netui_Form_0','/dev/formControls/submit.do');return
false;">Submit Form Through a Link</a>
```

4. Button

The `<netui:button>` tag acts as the standard button used to submit a form. As a default the button acts as an HTML `type="submit"` button. (The `type` attribute may also have a value of `reset` or `button`.) In the example below, when the users presses the generated button it will cause the form to be submitted. The button must appear within the `<netui:form>`

The button does not bind to a `dataSource` of any type.

```
<netui:form>
```

```
   ...
   <netui:button type="submit">Submit the Form</netui:button>
</netui:form>
```

5. CheckBox

The `<netui:checkBox>` binds a boolean value to a `dataSource`. The `<netui:checkBox>` will generate an `<input type="checkbox">` HTML element. The natural data binding type for the `dataSource` is a `boolean` or `java.lang.Boolean`. In addition, it is possible to bind to a `java.lang.String`.

In the following example, there are three checkboxes which are bound to a `boolean`, `java.lang.Boolean` and a `java.lang.String`.

```
<table>
    <tr><th align="right">CheckBox [boolean]</th>
        <td><netui:checkBox dataSource="actionForm.checkBox"/></td></tr>
    <tr><th align="right">CheckBox [java.lang.Boolean]</th>
        <td><netui:checkBox
dataSource="actionForm.checkBoxBoolean"/></td></tr>
    <tr><th align="right">CheckBox [java.lang.String]</th>
        <td><netui:checkBox
dataSource="actionForm.checkBoxString"/></td></tr>
</table>
```

The NetUI tags above bind to the following three FormData properties.

```
        private boolean checkBox;
        private Boolean checkBoxBoolean;
        private String  checkBoxString;

        public boolean isCheckBox() {
            return checkBox;
        }
        public void setCheckBox(boolean checkBox) {
            this.checkBox = checkBox;
        }

        public Boolean getCheckBoxBoolean() {
            return checkBoxBoolean;
        }
        public void setCheckBoxBoolean(Boolean checkBoxBoolean) {
            this.checkBoxBoolean = checkBoxBoolean;
        }

        public String getCheckBoxString() {
            return checkBoxString;
        }
        public void setCheckBoxString(String checkBoxString) {
            this.checkBoxString = checkBoxString;
        }
```

6. CheckBoxGroup

The `<netui:checkBoxGroup>` creates a group of boolean values. Multiple values may be selected at one time. The natural type for the `dataSource` is a `java.lang.String[]`. Each `<netui:checkBoxOption>` contains a value that will be posted back if the item is selected. See the CheckBoxOption for more information on how options are specified. In addition, the repeating version of the <netui:checkBoxGroup> is covered in the advanced topic on Repeating CheckBoxGroup.

There are two methods for creating the options which create the items inside of the group. The body of the tag may contain `<netui:checkBoxOption>` tags or the `optionsDataSource` may bind to an array of values. When using the `optionsDataSource` this tag may act as a repeater. This allows for control of both the layout and the presentation of the options.

In the following example, a <netui:checkBoxGroup> contains four children defined through <netui:checkBoxOptions>. When the form is posted, all of the values that are selected (checked) will be posted back to an array in the `FormData`.

```
<netui:checkBoxGroup dataSource="actionForm.checkBoxGroup"
orientation="vertical">
    <netui:checkBoxOption value="Check One"/>
    <netui:checkBoxOption value="Check Two"/>
    <netui:checkBoxOption value="Check Three"/>
    <netui:checkBoxOption value="Check Four"/>
</netui:checkBoxGroup>
```

The following property in the `FormData` will recieve the results of the posted CheckBoxGroup.

```
private String[] checkBoxGroup;

public String[] getCheckBoxGroup() {
    return checkBoxGroup;
}
public void setCheckBoxGroup(String[] checkBoxGroup) {
    this.checkBoxGroup = checkBoxGroup;
}
```

7. CheckBoxOption

The `<netui:checkBoxOption>` is a tag that defines an item within a `<netui:checkBoxGroup>`. The tag has a required `value` attribute which defines the value that will be posted back when the generated checkbox is selected. There is also an

optional `label` attribute that defines a text label for the checkbox. If the `label` is not specified the `value` attributes value will be used.

In the CheckBoxGroup defined below, there are four `<netui:checkBoxOption>` tags. All of them specify just the `value` attribute which is used for both the value and label in the generated HTML.

```
<netui:checkBoxGroup dataSource="actionForm.checkBoxGroup"
orientation="vertical">
    <netui:checkBoxOption value="Check One"/>
    <netui:checkBoxOption value="Check Two"/>
    <netui:checkBoxOption value="Check Three"/>
    <netui:checkBoxOption value="Check Four"/>
</netui:checkBoxGroup>
```

8. FileUpload

The `<netui:fileUpload>` tag allows a file to be uploaded to the server. By default, this tag is not enabled because multipart request handling is disabled. The only supported data type that can be bound to is `org.apache.struts.upload.FormFile`. If multipart handling is enabled, when the form is submitted, the contents of the file will be posted to the server.

By default, multipart request handling is disabled. There are two ways to enable handling: (1) you may turn it on for the WebApp by setting a configuration option in the `beehive-netui-config.xml` file or (2) you may turn it on for a page flow by setting the Jpf.MultipartHandler annotation. (These two options are detailed below.)

In the example below, the `<netui:fileUpload>` tag is used to upload a file to the server. In order to use the file upload tag you must set the `enctype` on the `<netui:form>` to `multipart/form-data`. This is required to enable multipart data. The `dataSource` is bound to a property of type `org.apache.struts.upload.FormFile`.

```
<netui:form action="submit"  enctype="multipart/form-data">
   ...
   <netui:fileUpload dataSource="actionForm.fileUpload"/>
</netui:form>
```

The following `FormData` properties will receive the information and data associated with the uploaded file.

```
private FormFile fileUpload;

public FormFile getFileUpload() {
```

```
    return fileUpload;
}
public void setFileUpload(FormFile fileUpload) {
    this.fileUpload = fileUpload;
}
```

In this example, multipart handling was enabled for in-memory support using an annotation on the page flow.

```
@Jpf.Controller(
    simpleActions={
        @Jpf.SimpleAction(name="begin", path="formTest.jsp")
    },
    multipartHandler=Jpf.MultipartHandler.memory
)
```

The following modification to the `beehive-netui-config.xml` file will turn on in-memory processing of multipart requests for the entire WebApp. See the beehive-netui-config.xml File Reference for information on creating an initial `beehive-netui-config.xml`.

```
<pageflow-config>
    <multipart-handler>memory</multipart-handler>
</pageflow-config>
```

9. Form

The `<netui:form>` tag creates the HTML `<form>` element. It specifies the action to run in the page flow. If the action specifies a parameter that is a subclass of `FormData` a bean will be populated by the page flow framework with values posted by the controls contained in the form. In the example below, the form will call the page flow's `submit` action.

```
<netui:form action="submit">
   ...
</netui:form>
```

NOTE: the `enctype="multipart/form-data"` from the FileUpload example is present because the `<netui:fileUpload>` tag is being used and requires multipart request handling. This is typically not used for simple form posting.

The following action defined in the page flow controller will be called when the form is submitted. An instance of `FormBean` is created and populated by the contents of the request.

```
@Jpf.Action(
    forwards={@Jpf.Forward(name="success",
           path="results.jsp",
           actionOutputs={@Jpf.ActionOutput(name="bean",
type=FormBean.class, required=true)}
          )
    }
)
public Forward submit(FormBean formBean) {
    Forward fwd = new Forward("success", "bean", formBean);
    return fwd;
}
```

10. Hidden

The `<netui:hidden>` tag creates the HTML `<input type="hidden">` element. There is no visual representation of a hidden field, but it's `value` will be posted to the server when the form is posted. The `<netui:hidden>` is a bit different than other form controls. It contains both a `dataInput` and a `dataSource` attribute. This allows binding a value from a source that differs from what is posted back to the server. If the `dataInput` is not set, the `dataSource` is used as a read/write binding.

In the example below, there are a number of hidden fields defined. All of them bind values out of the `pageFlow` implicit object (see the Data binding to NetUI Implicit Objects for more information.) In the example, the hidden fields bind to different types of data objects. The first simply binds the value back to a `java.lang.String` value. The second and third hidden fields bind to the same array of `java.lang.String`s. The final hidden field binds to a boolean value.

```
<table>
    <tr><th align="right">Hidden</th>
        <td><netui:hidden dataInput="${pageFlow.hidden}"
dataSource="actionForm.hidden"/></td></tr>
    <tr><th align="right">Hidden Array[0]</th>
        <td><netui:hidden dataInput="${pageFlow.hiddenArray[0]}"
dataSource="actionForm.hiddenArray"/></td></tr>
    <tr><th align="right">Hidden Array[1]</th>
        <td><netui:hidden dataInput="${pageFlow.hiddenArray[1]}"
dataSource="actionForm.hiddenArray"/></td></tr>
    <tr><th align="right">Hidden</th>
        <td><netui:hidden dataInput="${pageFlow.hiddenBoolean}"
dataSource="actionForm.hiddenBoolean"/></td></tr>
</table>
```

The following properties are defined in the `FormData` and are set by the above hidden elements when the form is submitted.

```
// hidden
```

```
private String hidden;
private String[] hiddenArray;
private boolean hiddenBoolean;

public String getHidden() {
    return hidden;
}
public void setHidden(String hidden) {
    this.hidden = hidden;
}

public String[] getHiddenArray() {
    return hiddenArray;
}
public void setHiddenArray(String hiddenArray[]) {
    this.hiddenArray = hiddenArray;
}

public boolean isHiddenBoolean() {
    return hiddenBoolean;
}
public void setHiddenBoolean(boolean hiddenBoolean) {
    this.hiddenBoolean = hiddenBoolean;
}
```

11. ImageButton

The `<netui:imageButton>` creates the HMTL `<input type="image">` element. The result is an image used to submit the form. The source of the image is specified in the `src` attribute. The `<netui:imageButton>` does not bind to any data. In addition, it must appear inside of the form.

```
<netui:form>
   ...
   <netui:imageButton  src="insert.gif"/>
</netui:form>
```

12. RadioButtonGroup

The `<netui:radioButtonGroup>` is similar to the `<netui:checkBoxGroup>`. It binds to a group of boolean values, but instead of allowing multiple selection it only allows a single selection. The underlying HTML produced is a set of HTML `<input type="radio">` elements where the name is the same value for each option, thus creating a group of radio buttons, of which only one may be selected at a time. The natural binding for the `<netui:radioButtonGroup>` is a `java.lang.String`. See the RadioButtonOption topic for information on creating the options. In addition, the repeating version of the `<netui:radioButtonGroup>` is covered in the advanced topic on

Repeating RadioButtonGroup.

There are two methods for creating the options which render the radio buttons inside of the group. The body of the tag may contain `<netui:RadioButtonOptions>` tags or the `optionsDataSource` my bind to an array of values. When using the `optionsDataSource`, this tag may act as a repeater. This allows for control of both the layout and presentation of the options.

In the following example, the `<netui:radioButtonGroup>` contains four options defined through the `<netui:radioButtonOption>` tags. When the form is posted, if one of the options is selected, its `value` will be posted back.

```
<netui:radioButtonGroup dataSource="actionForm.radioButtonGroup"
orientation="vertical">
    <netui:radioButtonOption value="Radio One"/>
    <netui:radioButtonOption value="Radio Two"/>
    <netui:radioButtonOption value="Radio Three"/>
    <netui:radioButtonOption value="Radio Four"/>
</netui:radioButtonGroup>
```

The natural binding for a `<netui:radioButtonGroup>` is a `java.lang.String`. The following property defined in the `FormData` receives the results of posting the form.

```
private String radioButtonGroup;

public String getRadioButtonGroup() {
    return radioButtonGroup;
}
public void setRadioButtonGroup(String radioButtonGroup) {
    this.radioButtonGroup = radioButtonGroup;
}
```

13. RadioButtonOption

The `<netui:radioButtonOption>` is a tag that defines an item within a `<netui:radioButtonGroup>`. The tag has a required `value` attribute which defines the value that will be posted back when the radio button option is selected. There is also an optional `label` attribute that defines a text label for the radio button. If the `label` is not specified, the `value` attribute will be used for the label.

In the RadioButtonGroup defined below, there are four `<netui:radioButtonOption` tags. All of them specify just the value attribute which is used for both the value and label in the generated HTML.

```
<netui:radioButtonGroup dataSource="actionForm.radioButtonGroup"
orientation="vertical">
```

```
        <netui:radioButtonOption value="Radio One"/>
        <netui:radioButtonOption value="Radio Two"/>
        <netui:radioButtonOption value="Radio Three"/>
        <netui:radioButtonOption value="Radio Four"/>
    </netui:radioButtonGroup>
```

14. Select

The `<netui:select>` is a control allowing a user to select one or more items from a list of items. There are two primary modes, single selection and multiple selections. There are two ways to specify the items of the select control, you may use `<netui:selectOption>` tags or the `optionsDataSource` attribute. For more information on using `<netui:selectOption>` tags see the SelectOption topic. In addition, the select supports a repeating mode that allows control over the order of the options when using the `optionsDataSource`. For more information see the Repeating Select topic.

14.1. Single Selection

Single selection is the default behavior for a select control. It allows a single value from the list to be selected. The natural data type that is bound to is a `java.lang.String`.

In the example below, a select box contains four options.

```
<netui:select dataSource="actionForm.singleSelect">
    <netui:selectOption value="SelectOne">Select One</netui:selectOption>
    <netui:selectOption value="SelectTwo">Select Two</netui:selectOption>
    <netui:selectOption value="SelectThree">Select
Three</netui:selectOption>
    <netui:selectOption value="SelectFour">Select Four</netui:selectOption>
</netui:select>
```

When the above select control is posted it is bound to a `java.lang.String` property on the `FormData`.

```
private String singleSelect;

public String getSingleSelect() {
    return singleSelect;
}
public void setSingleSelect(String singleSelect) {
    this.singleSelect = singleSelect;
}
```

14.2. Multiple Selection

To create a multiple select control you specify `multiple="true"` in the `<netui:select>` tag. A multiple select control allows zero or more items to be selected and posted back to the server. The natural binding type is `java.lang.String[]`.

The example below defines a multiple select control by setting the `multiple` attribute to `true`. This selection has four options.

```
<netui:select multiple="true"  dataSource="actionForm.multiSelect">
    <netui:selectOption value="SelectOne">Select One</netui:selectOption>
    <netui:selectOption value="SelectTwo">Select Two</netui:selectOption>
    <netui:selectOption value="SelectThree">Select
Three</netui:selectOption>
    <netui:selectOption value="SelectFour">Select Four</netui:selectOption>
</netui:select>
```

The above select control will post zero or more values into the following property on the `FormData`.

```
private String[] multiSelect;

public String[] getMultiSelect() {
    return multiSelect;
}
public void setMultiSelect(String[] multiSelect) {
    this.multiSelect = multiSelect;
}
```

15. SelectOption

The `<netui:selectOption>` is used to create statically defined options for a `<netui:select>` control. The `value` attribute is required and specifies the value that will be posted if the option is selected. In addition, the body of the tag may contain a value that will be used to display the option in the select control. If the body is empty, then the `value` will be used.

```
<netui:select multiple="true"  dataSource="actionForm.multiSelect">
    <netui:selectOption value="SelectOne">Select One</netui:selectOption>
    <netui:selectOption value="SelectTwo">Select Two</netui:selectOption>
    <netui:selectOption value="SelectThree">Select
Three</netui:selectOption>
    <netui:selectOption value="SelectFour">Select Four</netui:selectOption>
</netui:select>
```

16. TextArea

The `<netui:textArea>` creates a multiple line text entry control. It maps to the HTML

<textArea> element. It is very common to set the rows and cols attributes to set the size of the text area on the page. The natural data type that is bound to is java.lang.String. In the code below a text area is defined and binds to a property in the actionForm.

```
<netui:textArea dataSource="actionForm.textArea" />
```

The following property of the FormData receives the posted value:

```
private String textArea;

public String getTextArea() {
    return textArea;
}
public void setTextArea(String textArea) {
    this.textArea = textArea;
}
```

17. TextBox

The <netui:textBox> creates a single line text entry control. The tag supports either text or password by setting the type attribute. The default is text. The natural data type bound to by a text box is java.lang.String. The code below creates a text box and binds it to a property in the actionForm

```
<netui:textBox dataSource="actionForm.textBox" />
```

The following property of the FormData receives the posted value:

```
private String textBox;

public String getTextBox() {
    return textBox;
}
public void setTextBox(String textBox) {
    this.textBox = textBox;
}
```