Beehive NetUI Tutorial

Table of contents

1. Introduction

The NetUI tutorial is provided as a way to become familiar with NetUI's Page Flow controllers and JSP tags. The tutorial walks through creating, building, and deploying a sample project page flow that submits data from a browser to the server.

1.1. Tutorial Goals

- How to create a basic NetUI web application.
- How to perform simple user navigation with the `@Jpf.SimpleAction` annotation.
- How to coordinate user navigation with the `@Jpf.Forward` annotation and the `Forward` object.
- How to handle data submission and processing with databinding and form beans.
- How to create a user interface with the NetUI JSP tag library.
- How page flows help to separate data processing and data presentation.
- How to make a web application a client of a Java control.
- How to use declarative validation with data submission.
- How to collect data from a nested page flow and 'return' it to the calling page flow.
- How to make an action available to multiple page flows.

2. Step 1: Set up a Beehive-enabled Web Application

2.1. Set Shell Variables

Complete all of the necessary and optional steps in the following topic: Beehive Installation and Setup

2.2. Create a Beehive-enabled Web Project

In order to follow the steps in this tutorial, it's necessary to create a Beehive-enabled web application. Beehive-enabled web applications are described here. A skeleton Beehive-enabled web project is provided in the samples/ directory as netui-blank. This contains a basic Ant build file and an example Page Flow controller. To create the tutorial's project, we'll copy and then rename the netui-blank project using these steps:

1. Create a directory `/beehive_projects` (on Windows, this would be `C:\beehive_projects`).
2. Run this Ant target to create a new NetUI project: `ant -f <beehive-root>/beehive-imports.xml new.netui.webapp` and provide a fully-qualified web project root directory named `netui-tutorial`. Note,

`<beehive-root>` is the directory that contains a Beehive distribution; a typical value might be `/apache/apache-beehive-1.0`.

3. Before continuing, confirm that the following directory structure exists:

```
beehive_projects/
    netui-tutorial/
        src/
            Controller.java
        web/
            index.jsp
            resources/
            WEB-INF/
        build.properties
        build.xml
```

Note, this directory structure is just an example; you are free to put the `netui-tutorial` directory anywhere on disk. In the remainder of this tutorial, the directory `beehive_projects/netui-tutorial` will simply be referred to as `netui-tutorial`.

2.3. Configure Build Properties

The `build.properties` file contains several project-related properties that must be set in order to build the web application. Specifically, the paths to your Beehive distribution and to the JSP / Servlet API JARs for your application container must be set. The following steps will set these properties for the `netui-tutorial` webapp.

1. Open the file `netui-tutorial/build.properties` in a text editor.
2. Edit the `beehive.home` property so it points to the top-level directory of your Beehive distribution.
3. Edit the `context.path` to use the value `netui-tutorial`.

Note:

The `context.path` property determines both (1) the name of the application WAR file and (2) the application URL.
If `context.path=netui-tutorial`, then the following WAR file will be produced:
`netui-tutorial``.war`
and the following URL will invoke the web application:
`http://<some server>/`**`netui-tutorial`**

For example, if your Beehive distribution is located in `/apache/apache-beehive-1.0`, then your `build.properties` file would appear as follows.

```
beehive.home=/apache/apache-beehive-1.0

servlet-api.jar=${os.CATALINA_HOME}/common/lib/servlet-api.jar
jsp-api.jar=${os.CATALINA_HOME}/common/lib/jsp-api.jar
```

```
context.path=netui-tutorial
```

Note:
Properties files should use the '/' character to separate drive, directory, and file names.

If you are using an application container other than Tomcat, be sure to set the `servlet-api.jar` and `jsp-api.jar` properties to reference the JAR your server provides which contains the JSP and Servlet API classes.

2.4. Start the Server

If you are using Tomcat, enter the following at the command prompt:

```
$CATALINA_HOME/bin/startup.bat
```

If you aren't using Tomcat, start your application container as per its directions.

2.5. Using URLs in the Examples

In the Beehive tutorials, you will often encounter URLs like `http://localhost:8080/netui-tutorial/myFlow/Controller.jpf`. When you see these URLs, they are meant to be accessed via your web browser to demonstrate functionality built in the tutorial. These URLs are set up to run on Tomcat, so if you are unable to run these URLs, be sure that they are appropriate for your application server. Specifically, check the port numbers to make sure that your server is running on the referenced port.

3. Step 2: Create your First Page Flow Controller

3.1. Overview

In this step you will create a controller class and a JSP. These are the basic files in a Beehive NetUI web application. Each page flow contains one controller class and any number of pages -- JSPs in this case. A controller class is a Java class that controls how your web application functions and what it does. The methods and annotations in the controller class determine how users navigate from page to page, how user requests are handled, and how the web application accesses back-end resources. The JSPs determine what a visitor to the web application sees in the browser.

In terms of the Model-View-Controller paradigm for web applications, the Controller.java file is the Controller (naturally) and the JSPs are the View. The web application's Model in this tutorial is very simple: it consists of a JavaBean with three fields that represent the user's

name, age and selected sport activity.

Controller classes contain **actions**, which are methods or annotations. An action may do something simple, such as forward a user from one JSP to another; or it may do a complex set of tasks, such as receive user input from a JSP, interact with back-end resources based on the user input, and forward the user to a JSP where the results are displayed.

The controller class in this step contains one action. This simple navigational action forwards users to the `index.jsp` page. In the next step, you will create a more complex action.

3.2. Create the Page Flow Files

You will create two files in a new page flow: `Controller.java` and `index.jsp`. First, create a directory called `myFlow` under the `src` directory. Then add the following page flow controller class:

src/myFlow/Controller.java

```
package myFlow;

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

@Jpf.Controller(
    simpleActions={
        @Jpf.SimpleAction(name="begin", path="index.jsp")
    }
)
public class Controller
    extends PageFlowController
{
}
```

Every Page Flow controller class must extend `PageFlowController` and be decorated by the annotation `@Jpf.Controller`. The class you created is the simplest controller possible; it has a single `begin` action that forwards to `index.jsp`.

The controller class is activated when a user hits it via the URL:

```
http://localhost:8080/netui-tutorial/myFlow/Controller.jpf
```

The URL above means this: "Run the `begin` action of the `Controller` class in the `myFlow` directory of the `netui-tutorial` web application."

Now, create a directory called `myFlow` under the `web` directory. Then, create the `index.jsp` that will be shown when you hit the page flow:

web/myFlow/index.jsp

```
<%@ page language="java" contentType="text/html;charset=UTF-8"%>
<%@ taglib uri="http://beehive.apache.org/netui/tags-html-1.0"
prefix="netui"%>
<netui:html>
  <head>
    <title>Web Application Page</title>
    <netui:base/>
  </head>
  <netui:body>
    <p>
      New Web Application Page
    </p>
  </netui:body>
</netui:html>
```

Note:

Note, we placed the pages in a **parallel web content directory**. NetUI assumes that the pages in `web/myFlow` go with the controller class in `src/myFlow`.

3.3. Compile and Deploy the Web Application

You are now ready to compile the page flow and deploy it to Tomcat.

The following Ant command assumes that you are in the `netui-tutorial/` directory. At the command prompt, enter:

```
ant clean build war
```

This will build the webapp by running the Beehive annotation processors and will produce class files in `WEB-INF/classes`. Now, the application is ready to deploy to your server. On Tomcat, copy the WAR file into Tomcat's `$CATALINA_HOME/webapps` directory.

On Windows:

```
copy netui-tutorial.war %CATALINA_HOME%\webapps /Y
```

Note:

On Windows, there are file-locking issues that Tomcat versions 5.5.x and above are sensitive to. In particular, any web application that uses Struts will **fail to redeploy** if you copy in a new .war file as described here. The Commons Digester team is adding a workaround for the issue (see this bug), but in the meantime, you can work around it with the `antiResourceLocking` option in Tomcat. Just add a file called `context.xml` in a directory called `META-INF` inside the `web` directory before building (so it will end up as `META-INF/context.xml` in your `netui-tutorial.war`):

```
    <?xml version="1.0" encoding="UTF-8"?>
    <Context antiResourceLocking="true">
    </Context>
```

Everywhere else:

```
cp netui-tutorial.war $CATALINA_HOME/webapps
```

If you are asked to overwrite the old WAR file, enter 'yes'. Note, when doing redeployment, you may have to wait a few seconds for Tomcat to redeploy the WAR file. Once deployment or redeployment has completed, the webapp can be accessed through a browser.

If you are not using Tomcat, follow your server's web application deployment instructions to deploy the webapp.

3.4. Test the NetUI Web Application

Visit the following address:

http://localhost:8080/netui-tutorial/myFlow/Controller.jpf

You will be directed to the `index.jsp` page.

4. Step 3: Navigation

4.1. Create a Destination JSP

In the directory `web/myFlow`, create a file named `page2.jsp`:

`web/myFlow/page2.jsp`

```
<%@ page language="java" contentType="text/html;charset=UTF-8"%>
<%@ taglib uri="http://beehive.apache.org/netui/tags-html-1.0"
prefix="netui"%>
<netui:html>
  <head>
    <title>page2.jsp</title>
    <netui:base/>
  </head>
  <netui:body>
    <p>
      Welcome to page2.jsp!
    </p>
  </netui:body>
</netui:html>
```

Save `page2.jsp`.

4.2. Create a Link to the Destination Page

In this step you will create a link from the JSP, `index.jsp` to a new Simple Action that you will add to the controller class. The new action will forward to `page2.jsp`.

Open the file `web/myFlow/index.jsp`.

Edit `index.jsp` so it appears as follows. The code to add appears in bold type.

`index.jsp`

```
<%@ page language="java" contentType="text/html;charset=UTF-8"%>
<%@ taglib uri="http://beehive.apache.org/netui/tags-html-1.0"
prefix="netui"%>
<netui:html>
  <head>
    <title>Web Application Page</title>
    <netui:base/>
  </head>
  <netui:body>
    <p>
      New Web Application Page
    </p>
    <p>
      <netui:anchor action="toPage2">Link to page2.jsp</netui:anchor>
    </p>
  </netui:body>
</netui:html>
```

Save `index.jsp`.

4.3. Add a Simple Action to Handle the Link

Open the file `src/myFlow/Controller.java`.

Edit `Controller.java` so it appears as follows. Don't forget the comma after the first `Jpf.SimpleAction(...)` element!

`Controller.java`

```
package myFlow;

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

@Jpf.Controller(
    simpleActions={
        @Jpf.SimpleAction(name="begin", path="index.jsp"),
        @Jpf.SimpleAction(name="toPage2", path="page2.jsp")
    }
)
public class Controller
    extends PageFlowController
{
}
```

Save `Controller.java`.

4.4. Compile and Redeploy the Web Application

Compile and deploy the page flow using the same Ant and copy commands used in step 2.

If you are asked to overwrite the old WAR file, enter 'Yes'.

Wait a few seconds for Tomcat to redeploy the WAR file, then move on to the next step.

4.5. Test the NetUI Web Application

Visit the following link:

http://localhost:8080/netui-tutorial/myFlow/Controller.jpf

You will be directed to the `index.jsp` page.

Click the link. You will be directed to page2.jsp.

5. Step 4: Submitting Data

5.1. Create a Submission Form

This step illustrates the use of NetUI tags to render an HTML form tag and link it to a Page Flow action. In a later step, the new action (`processData`) will be added to the controller class to handle the data submission.

Edit the file `web/myFlow/page2.jsp` so it appears as follows.

`page2.jsp`

```
<%@ page language="java" contentType="text/html;charset=UTF-8"%>
<%@ taglib uri="http://beehive.apache.org/netui/tags-html-1.0"
prefix="netui"%>
<netui:html>
  <head>
    <title>page2.jsp</title>
    <netui:base/>
  </head>
  <netui:body>
    <p>
      Welcome to page2.jsp!
    </p>
    <p>
      <netui:form action="processData">
```

```
        Name: <netui:textBox dataSource="actionForm.name"/>
        <br/>
        Age: <netui:textBox dataSource="actionForm.age"/>
        <br/>
        <netui:button type="submit" value="Submit"/>
      </netui:form>
    </p>
  </netui:body>
</netui:html>
```

Save `page2.jsp`.

5.2. Create a Server Side Representation of the Submission Form (a.k.a. Create a Form Bean)

In this step you will create a Java class that accepts data from the JSP submission form created in the previous task. When the form data is submitted, the Java class will be instantiated, and the form data will be loaded into the JavaBean properties of the new instance.

In the directory `src` create a directory named **`forms`**.

In the directory `src/forms` create a JAVA file named **`ProfileForm.java`**.

Edit `src/forms/ProfileForm.java` so it contains getters and setters for two JavaBean properties (`name` and `age`), as follows.

`ProfileForm.java`

```
package forms;

public class ProfileForm
    implements java.io.Serializable {

    private int age;
    private String name;

    public void setName(String name) {
        this.name = name;
    }

    public String getName() {
        return this.name;
    }

    public void setAge(int age) {
        this.age = age;
    }

    public int getAge() {
```

```
        return this.age;
    }
}
```

Save and close `ProfileForm.java`.

5.3. Edit the Controller Class to Handle the Submitted Data

Now you will add a new action and use your new form bean to handle the data submitted from the HTML form.

Open the file `src/myFlow/Controller.java`

Edit `Controller.java` so it appears as follows. Code to add appears in bold type.

`Controller.java`

```
package myFlow;

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

@Jpf.Controller(
    simpleActions={
        @Jpf.SimpleAction(name="begin", path="index.jsp"),
        @Jpf.SimpleAction(name="toPage2", path="page2.jsp")
    }
)
public class Controller
    extends PageFlowController
{

    @Jpf.Action(
        forwards = {
            @Jpf.Forward(name="success", path="page2.jsp")
        }
    )
    public Forward processData(ProfileForm form) {
        System.out.println("Name: " + form.getName());
        System.out.println("Age: " + form.getAge());

        Forward fwd = new Forward("success");
        return fwd;
    }
}
```

Save `Controller.java`.

5.4. Recompile and Redeploy the Web Application

Compile and (re)deploy the web application using the same steps as described here.

5.5. Test the NetUI Web Application

Visit the following link:

http://localhost:8080/netui-tutorial/myFlow/Controller.jpf

You will be directed to the `index.jsp` page.

Click the link.

You will be directed to page2.jsp.

Enter values in the Name and Age fields, and click Submit.

Notice the name and age values you entered are displayed in the Tomcat console.

6. Step 5: Processing and Displaying Data

6.1. Create a JSP to Display Submitted Data

In this step you will create a new JSP to present the results from processing the data submission.

In the directory `web/myFlow` create a file named **`displayData.jsp`**.

Edit `displayData.jsp` so it appears as follows. The "page inputs" that are being displayed will come from the action in your page flow (next step).

`displayData.jsp`

```
<%@ page language="java" contentType="text/html;charset=UTF-8"%>
<%@ taglib uri="http://beehive.apache.org/netui/tags-html-1.0"
prefix="netui"%>
<netui:html>
  <head>
    <title>displayData.jsp</title>
    <netui:base/>
  </head>
  <netui:body>
    <p>
      You submitted the following information:
    </p>
    <p>
      Name: ${pageInput.name}
      <br/>
```

```
      Age: ${pageInput.age}
    </p>
  </netui:body>
</netui:html>
```

Save and close `displayData.jsp`.

6.2. Process the Submitted Data

Edit the `processData` method in the `Controller.java` file so it appears as follows. Code to add appears in bold. Change the value of the path attribute in the Forward annotation to `displayData.jsp`. Note that the "action outputs" you are adding here will be read as "page inputs" on the page.

`Controller.java`

```
    ...

    @Jpf.Action(
        forwards = {
            @Jpf.Forward(name="success", path="displayData.jsp")
        }
    )
    public Forward processData(ProfileForm form) {
        System.out.println("Name: " + form.getName());
        System.out.println("Age: " + form.getAge());

        Forward fwd = new Forward("success");
        fwd.addActionOutput("name", form.getName());
        fwd.addActionOutput("age", form.getAge());
        return fwd;
    }

    ...
```

Save `Controller.java`.

6.3. Recompile and Redeploy the Web Application

Compile and (re)deploy the web application using the same steps as described here.

6.4. Test the NetUI Web Application

Visit the following link:

http://localhost:8080/netui-tutorial/myFlow/Controller.jpf

You will be directed to the `index.jsp` page.

Click the link.

You will be directed to page2.jsp.

Enter values in the Name and Age fields. Click the Submit button.

You will be forwarded to the `displayData.jsp` page. Notice the values you entered are displayed.

Note:

In this step, you used "action outputs" and "page inputs" to get data from your page flow controller to your JSP. You can get more formal about both by declaring expected types and whether the values are required; to do this, you use annnotations and JSP tags as described here.

7. Step 6: Add a Control

In this step you will add a simple 'Hello World' control to your web application.

You will edit the web application to become a client of the control. The web app will pass the user submitted name to the control, and the control will return a simple 'Hello World' message back to the web app. For more details on how this control works see the control tutorial.

7.1. Create the HelloWorld Control

Inside the `netui-tutorial/src` directory, create a new directory named `controls`.

Inside the directory `netui-tutorial/src/controls`, create a new file named `HelloWorldImpl.java`

Edit `HelloWorldImpl.java` so it appears as follows:

```
package controls;

import org.apache.beehive.controls.api.bean.ControlImplementation;

@ControlImplementation(isTransient=true)
public class HelloWorldImpl implements HelloWorld {

    public String hello() {
        return "hello!";
    }

    public String helloParam(String name) {
        return "Hello, " + name + "!";
    }
}
```

Inside the directory `netui-tutorial/src/controls`, create a new file named `HelloWorld.java`

Edit `HelloWorld.java` so it appears as follows:

```
package controls;

import org.apache.beehive.controls.api.bean.ControlInterface;

@ControlInterface
public interface HelloWorld {

    String hello();

    String helloParam(String name);
}
```

7.2. Edit the Page Flow Controller

Edit the page flow Controller file so it appears as follows. Code to add appears in bold:

```
package myFlow;

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

import org.apache.beehive.controls.api.bean.Control;
import controls.HelloWorld;

@Jpf.Controller(
    simpleActions={
        @Jpf.SimpleAction(name="begin", path="index.jsp"),
        @Jpf.SimpleAction(name="toPage2", path="page2.jsp")
    }
)
public class Controller
    extends PageFlowController
{
    @Control
    private HelloWorld helloWorld;

    @Jpf.Action(
        forwards = {
            @Jpf.Forward(name="success", path="displayData.jsp")
        }
    )
    public Forward processData(ProfileForm form) {
        System.out.println("Name: " + form.getName());
        System.out.println("Age: " + form.getAge());

        Forward fwd = new Forward("success");
```

```
        fwd.addActionOutput("name", form.getName());
        fwd.addActionOutput("age", form.getAge());
        fwd.addActionOutput("message",
helloWorld.helloParam(form.getName()));
        return fwd;
    }
}
```

7.3. Add a New pageInput

Add new pageinput to displayData.jsp. Code to add appears in bold:

```
<%@ page language="java" contentType="text/html;charset=UTF-8"%>
<%@ taglib uri="http://beehive.apache.org/netui/tags-html-1.0"
prefix="netui"%>
<netui:html>
  <head>
    <title>displayData.jsp</title>
    <netui:base/>
  </head>
  <netui:body>
    <p>
      You submitted the following information:
    </p>
    <p>
      Name: ${pageInput.name}
      <br/>
      Age: ${pageInput.age}
      <br/>
      Message: ${pageInput.message}
    </p>
  </netui:body>
</netui:html>
```

7.4. Recompile and Redeploy the Web Application

Compile and (re)deploy the web application using the same steps as described here.

7.5. Test the NetUI Web Application

Visit the following link:

http://localhost:8080/netui-tutorial/myFlow/Controller.jpf

You will be directed to the `index.jsp` page.

Click the link.

You will be directed to page2.jsp.

Enter values in the Name and Age fields. Click the Submit button.

You will be forwarded to the `displayData.jsp` page. Notice the values you entered are displayed along with a 'Hello World' message based on the name you submitted.

8. Step 7: Input Validation

8.1. Add Declarative Validation to the Form Bean

In this step you will use declarative validation to define the set of rules for each field, to be applied during input validation. You will add a `ValidatableProperty` annotation for the `name` property of the form so that it will (1) be required, and (2) have a maximum length of 30 characters. The `age` property must have a value in the range 1 to 130.

Open the file `src/forms/ProfileForm.java`

Add validation annotations. Code to add appears in bold (notice the additional `import` statement and the `FormBean` annotation).

`src/forms/ProfileForm.java`

```
package forms;

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

@Jpf.FormBean()
public class ProfileForm
    implements java.io.Serializable {

    private int age;
    private String name;

    public void setName(String name) {
        this.name = name;
    }

    @Jpf.ValidatableProperty(
        displayName = "Name",
        validateRequired = @Jpf.ValidateRequired(),
        validateMaxLength = @Jpf.ValidateMaxLength(chars = 30)
    )
    public String getName() {
        return this.name;
    }

    public void setAge(int age) {
        this.age = age;
    }
```

```
    @Jpf.ValidatableProperty(
        displayName = "Age",
        validateRange = @Jpf.ValidateRange(minInt = 1, maxInt = 130)
    )
    public int getAge() {
        return this.age;
    }
}
```

Save and close `ProfileForm.java`.

Note:

The validation annotations above do **not** produce localized messages. If you want your messages to be localizable, you would do two things: (1) add the `messageBundle` attribute to the `@Jpf.FormBean` annotation on the `ProfileForm` class, with its value set to a valid message bundle, and (2) use the `displayNameKey` attribute instead of `displayName` on your validation annotations.
If you don't want to use the default validation messages provided by NetUI, you can use the `message` or `messageKey` attributes on any of the validation annotations.

Next, add a `validationErrorForward` to the `processData` action in `src/myFlow/Controller.java`. Code to add appears in bold. Don't forget the comma after the `forwards={...}` element!

src/myFlow/Controller.java

```
package myFlow;

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

import org.apache.beehive.controls.api.bean.Control;
import controls.HelloWorld;

@Jpf.Controller(
    simpleActions={
        @Jpf.SimpleAction(name="begin", path="index.jsp"),
        @Jpf.SimpleAction(name="toPage2", path="page2.jsp")
    }
)
public class Controller
    extends PageFlowController
{
    @Control
    private HelloWorld helloWorld;

    @Jpf.Action(
        forwards = {
            @Jpf.Forward(name="success", path="displayData.jsp")
        },
        validationErrorForward=@Jpf.Forward(name="fail", path="page2.jsp")
    )
```

```
    public Forward processData(ProfileForm form) {
        System.out.println("Name: " + form.getName());
        System.out.println("Age: " + form.getAge());

        Forward fwd = new Forward("success");
        fwd.addActionOutput("name", form.getName());
        fwd.addActionOutput("age", form.getAge());
        fwd.addActionOutput("message",
helloWorld.helloParam(form.getName()));
        return fwd;
    }
}
```

Save `Controller.java`.

The annotation you just added causes navigation to flow back to `page2.jsp` if any validation error occurs in the form bean for action `processData`.

8.2. Modify the JSP to Display Validation Errors

Add the `<netui:error>` tag to display validation error messages on the page.

Edit the file `web/myFlow/page2.jsp` so it appears as follows.

`web/myFlow/page2.jsp`

```
<%@ page language="java" contentType="text/html;charset=UTF-8"%>
<%@ taglib uri="http://beehive.apache.org/netui/tags-html-1.0"
prefix="netui"%>
<netui:html>
  <head>
    <title>page2.jsp</title>
    <netui:base/>
  </head>
  <netui:body>
    <p>
      Welcome to page2.jsp!
    </p>
    <p>
      <netui:form action="processData">
        Name: <netui:textBox dataSource="actionForm.name"/>
        <b><netui:error key="name"/></b>
        <br/>
        Age: <netui:textBox dataSource="actionForm.age"/>
        <b><netui:error key="age"/></b>
        <br/>
        <netui:button type="submit" value="Submit"/>
      </netui:form>
    </p>
  </netui:body>
</netui:html>
```

Save and close `page2.jsp`.

Now, any validation error that happens for the `name` property will appear next to the "Name:" input field. Likewise for the `age` property.

8.3. Recompile and Redeploy the Web Application

Compile and (re)deploy the web application using the same steps as described here.

8.4. Test the NetUI Web Application

Visit the following link:

http://localhost:8080/netui-tutorial/myFlow/Controller.jpf

You will be directed to the `index.jsp` page.

Click the link.

You will be directed to page2.jsp.

Leave the Name field empty and enter a negative integer value in the Age field. Click the Submit button.

You will be returned to the `page2.jsp` page. Notice the error messages for the values you entered.

9. Step 8: Collect Data from a Nested Page Flow

Nested page flows allow you to insert *entire flows* in the middle of the current flow. One use for this is to collect data in another flow, but still come back to the current flow to use the data.

9.1. Link to the Nested Page Flow

In this task you will modify the HTML form tag to (1) add a new data field and (2) add a button that launches a nested page flow. (Later you will create the nested page flow to handle the data collection.)

Edit the file `web/myFlow/page2.jsp` so it appears as follows. Code to add appears in bold.

`web/myflow/page2.jsp`

```
<%@ page language="java" contentType="text/html;charset=UTF-8"%>
<%@ taglib uri="http://beehive.apache.org/netui/tags-html-1.0"
prefix="netui"%>
```

```
<netui:html>
  <head>
    <title>page2.jsp</title>
    <netui:base/>
  </head>
  <netui:body>
    <p>
      Welcome to page2.jsp!
    </p>
    <p>
      <netui:form action="processData">
        Name: <netui:textBox dataSource="actionForm.name"/>
        <netui:error key="name"/>
        <br/>
        Age: <netui:textBox dataSource="actionForm.age"/>
        <netui:error key="age"/>
        <br/>
        Sport: <netui:textBox dataSource="actionForm.sport"/>
        <br/>
        <netui:button type="submit" action="getSport" value="Select
Sport"/>
        <netui:button type="submit" value="Submit"/>
      </netui:form>
    </p>
  </netui:body>
</netui:html>
```

Save `page2.jsp`.

9.2. Update the Form Bean

In this task you will update the Java class that represents the submission form with the additional data field created in the previous task. When the nested page flow returns, the new member of the Form Bean class instance can be loaded with the value collected.

Edit `src/forms/ProfileForm.java` and add the following member variable and methods.

`ProfileForm.java`

```
    ...

    private String sport;

    public void setSport(String sport) {
        this.sport = sport;
    }

    public String getSport() {
        return this.sport;
    }
```

```
    ...
```

Save and close `ProfileForm.java`.

9.3. To Launch and Return from the Nested Page Flow

In this task you will add Action methods: one to handle forwarding to the nested page flow and another to implement the return Action when the nested page flow completes.

Open the file `src/myFlow/Controller.java`

Edit `Controller.java` so it appears as follows. Code to add appears in bold type. Don't forget to add the `useFormBean` property to the `@Jpf.Action` annotation of the `processData` method. The ProfileForm is page flow-scoped for this example, using the same Form Bean instance in multiple Action methods.

src/myFlow/Controller.java

```
package myFlow;

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

import org.apache.beehive.controls.api.bean.Control;
import controls.HelloWorld;

@Jpf.Controller(
    simpleActions={
        @Jpf.SimpleAction(name="begin", path="index.jsp"),
        @Jpf.SimpleAction(name="toPage2", path="page2.jsp")
    }
)
public class Controller
    extends PageFlowController
{

    @Control
    private HelloWorld helloWorld;

    private ProfileForm profileForm;

    /**
     * This action forwards to the nested page flow to collect the sport
     * name.  Note that it takes a ProfileForm so we can update the form
     * with the sport name returned from the nested page flow, but we've
     * explicitly turned validation off for this action, since the form
     * may be incomplete.
     */
```

```
    @Jpf.Action(
        useFormBean="profileForm",
        forwards={
            @Jpf.Forward(name="getSportFlow",
path="sports/SportsController.jpf")
        },
        doValidation=false
    )
    protected Forward getSport(ProfileForm form) {
        return new Forward("getSportFlow");
    }

    /**
     * This action takes the sport name returned from the nested page flow
     * and updates the field in the form and returns to the original page.
     */
    @Jpf.Action(
        forwards={
            @Jpf.Forward(name="success",
navigateTo=Jpf.NavigateTo.currentPage)
        }
    )
    protected Forward sportSelected(String sport) {
        profileForm.setSport(sport);
        Forward success = new Forward("success", profileForm);
        return success;
    }

    @Jpf.Action(
        useFormBean="profileForm",
        forwards = {
            @Jpf.Forward(name="success", path="displayData.jsp")
        },
        validationErrorForward = @Jpf.Forward(name="fail",
path="page2.jsp")
    )
    public Forward processData(ProfileForm form) {
        System.out.println("Name: " + form.getName());
        System.out.println("Age: " + form.getAge());

        Forward fwd = new Forward("success");
        fwd.addActionOutput("name", form.getName());
        fwd.addActionOutput("age", form.getAge());
        fwd.addActionOutput("message",
helloWorld.helloParam(form.getName()));
        fwd.addActionOutput("sport", form.getSport());
        return fwd;
    }
}
```

Save `Controller.java`.

9.4. Create a Nested Page Flow

In this task you will create a nested page flow with actions to select and confirm the data to return to the main ("nesting") page flow. The new nested controller class contains an inner Form Bean classs for the data collection. It has only a single field for the user's choice of sport activity. The options to be displayed are declared as member data of this nested page flow. After the user confirms the data, the nested page flow returns a `String` to the main page flow.

In the directory `src/myFlow` create a directory named **`sports`**.

In the directory `src/myFlow/sports` create a Java file named **`SportsController.java`**.

Edit `src/myFlow/sports/SportsController.java` so it appears as follows.

`SportsController.java`

```
package myFlow.sports;

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

@Jpf.Controller(
    nested = true,
    simpleActions = {
        @Jpf.SimpleAction(name="begin", path="index.jsp")
    }
)
public class SportsController
    extends PageFlowController {

    private String selectedSport;
    private String[] sports = {"sailing", "surfing", "diving",
"volleyball", "bicycling"};

    public String[] getSports() {
        return sports;
    }

    public String getSelectedSport() {
        return selectedSport;
    }

    @Jpf.Action(
        forwards = {
            @Jpf.Forward(name="confirm", path="confirm.jsp")
        }
    )
    public Forward selectSport(SportForm form) {
        selectedSport = form.getSport();
```

```
        return new Forward("confirm");
    }

    @Jpf.Action(
        forwards = {
            @Jpf.Forward(
                name="success",
                returnAction="sportSelected",
                outputFormBeanType=String.class)
        }
    )
    public Forward confirm() {
        return new Forward("success", selectedSport);
    }

    public static class SportForm
        implements java.io.Serializable {

        private String sport;

        public void setSport(String sport) {
            this.sport = sport;
        }

        public String getSport() {
            return this.sport;
        }
    }
}
```

Save and close `SportsController.java`.

9.5. To Present and Collect Data using a Form

This task illustrates the use of custom tags to render a radio button group in an HTML form and link it to the nested page flow `selectSport` Action method.

In the directory `web/myFlow` create a directory named **`sports`**.

In the directory `web/myFlow/sports`, create a file named `index.jsp`.

Edit index.jsp so it appears as follows.

`index.jsp`

```
<%@ page language="java" contentType="text/html;charset=UTF-8"%>
<%@ taglib uri="http://beehive.apache.org/netui/tags-html-1.0"
prefix="netui"%>
<netui:html>
  <head>
    <title>Select Sport</title>
    <netui:base/>
```

```
  </head>
  <netui:body>
    <p>
      Select Sport Activity
    </p>
    <p>
      <netui:form action="selectSport">
        <table>
          <tr>
            <td>Sports:</td>
            <td>
              <netui:radioButtonGroup dataSource="actionForm.sport"
                      optionsDataSource="${pageFlow.sports}"/>
            </td>
          </tr>
        </table>
        <netui:button type="submit">Submit</netui:button>
      </netui:form>
    </p>
  </netui:body>
</netui:html>
```

Save `index.jsp`.

9.6. Confirm the Selected Data

In the directory `web/myFlow/sports`, create a file named `confirm.jsp`.

Edit confirm.jsp so it appears as follows.

`confirm.jsp`

```
<%@ page language="java" contentType="text/html;charset=UTF-8"%>
<%@ taglib uri="http://beehive.apache.org/netui/tags-html-1.0"
prefix="netui"%>
<netui:html>
  <head>
    <title>Confirm Sport Activity</title>
    <netui:base/>
  </head>
  <netui:body>
    <p>
      Confirm Sport Activity
    </p>
    <p>
      Sport: ${pageFlow.selectedSport}
    </p>
    <netui:form action="confirm">
        <netui:button type="submit" value="Confirm"/>
    </netui:form>
  </netui:body>
</netui:html>
```

Save `confirm.jsp`.

9.7. Update the JSP to Display Submitted Data

In this step you will update the JSP to present the results from processing the data submission.

In the directory `web/myFlow` update the file named **`displayData.jsp`** and add code to display the additional data. The code to add appears in bold type.

Edit `displayData.jsp` so it appears as follows.

`displayData.jsp`

```
<%@ page language="java" contentType="text/html;charset=UTF-8"%>
<%@ taglib uri="http://beehive.apache.org/netui/tags-html-1.0"
prefix="netui"%>
<netui:html>
  <head>
    <title>displayData.jsp</title>
    <netui:base/>
  </head>
  <netui:body>
    <p>
      You submitted the following information:
    </p>
    <p>
      Name: ${pageInput.name}
      <br/>
      Age: ${pageInput.age}
      <br/>
      Message: ${pageInput.message}
      <br/>
      Sport: ${pageInput.sport}
    </p>
  </netui:body>
</netui:html>
```

Save and close `displayData.jsp`.

9.8. Recompile and Redeploy the Web Application

Compile and (re)deploy the web application using the same steps as described here.

9.9. To Test the NetUI Web Application

Visit the following link:

http://localhost:8080/netui-tutorial/myFlow/Controller.jpf

You will be directed to the `index.jsp` page.

Click the link.

You will be directed to page2.jsp.

Click the "Select Sport" button.

You will be directed to sports/index.jsp in the nested page flow.

Click a radio button and then the Submit button.

You will be directed to sports/confirm.jsp in the nested page flow.

Click the Confirm button.

You will be returned to the page2`.jsp` page. Notice that the value you selected is displayed in the Sport field.

10. Step 9: Adding Actions to a Shared Flow

10.1. To Create a Common Destination JSP

In the directory `web`, create a file named `help.jsp`.

Edit help.jsp so it appears as follows.

`help.jsp`

```
<%@ page language="java" contentType="text/html;charset=UTF-8"%>
<%@ taglib uri="http://beehive.apache.org/netui/tags-html-1.0"
prefix="netui"%>
<netui:html>
  <head>
    <title>Help Page</title>
    <netui:base/>
  </head>
  <netui:body>
    <p>
      Welcome to the Help Page
    </p>
  </netui:body>
</netui:html>
```

Save `help.jsp`.

10.2. Make an Action available to multiple Page Flows

In this task you will add a Simple Action to the existing Shared Flow. The Action forwards to the help page created in the previous task and will be available to multiple page flows in the application.

Open the existing Shared Flow file `src/shared/SharedFlow.java`

Edit the `@Jpf.Controller` annotation for the Shared Flow controller class, `SharedFlow`, in the `SharedFlow.java` file and add the `simpleActions` property. Code to add appears in bold. Don't forget the comma after the `catches={...}` element!

`SharedFlow.java`

```
...

@Jpf.Controller(
    catches={
       @Jpf.Catch(type=PageFlowException.class,
method="handlePageFlowException"),
       @Jpf.Catch(type=Exception.class, method="handleException")
    },
    simpleActions={
        @Jpf.SimpleAction(name="showHelp", path="/help.jsp")
    }
)
public class SharedFlow
    extends SharedFlowController {
    ...
}
```

Save `SharedFlow.java`.

10.3. Reference the Shared Flow from the Page Flow

Declare a shared flow reference in the page flow controller class, using the `@Jpf.SharedFlowRef` annotation. The declaration assigns a name to the referenced shared flow. This name can be used throughout the page flow to reference shared actions and state.

Open the file `src/myFlow/Controller.java`.

Edit `Controller.java` so it appears as follows. The code to add appears in bold type.

`Controller.java`

```
package myFlow;

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

```
import org.apache.beehive.netui.pageflow.Forward;
import forms.ProfileForm;

import org.apache.beehive.controls.api.bean.Control;
import controls.HelloWorld;

@Jpf.Controller(
    simpleActions={
        @Jpf.SimpleAction(name="begin", path="index.jsp"),
        @Jpf.SimpleAction(name="toPage2", path="page2.jsp")
    },
    sharedFlowRefs={
        @Jpf.SharedFlowRef(name="shared", type=shared.SharedFlow.class)
    }
)
public class Controller
    extends PageFlowController
{
    ...
}
```

Save `Controller.java`.

10.4. Link a Page to the Shared Flow Action

In this task you will create a link from the JSP, `index.jsp` to the Shared Flow Action.

Open the file `web/myFlow/index.jsp`.

Edit `index.jsp` so it appears as follows. The code to add appears in bold type.

`index.jsp`

```
<%@ page language="java" contentType="text/html;charset=UTF-8"%>
<%@ taglib uri="http://beehive.apache.org/netui/tags-html-1.0"
prefix="netui"%>
<netui:html>
  <head>
    <title>Web Application Page</title>
    <netui:base/>
  </head>
  <netui:body>
    <p>
      New Web Application Page
    </p>
    <p>
      <netui:anchor action="toPage2">Link to page2.jsp</netui:anchor>
    </p>
    <netui:anchor action="shared.showHelp" popup="true">Help
      <netui:configurePopup location="false" width="550" height="150">
      </netui:configurePopup>
    </netui:anchor>
  </netui:body>
```

```
</netui:html>
```

Save `index.jsp`.

10.5. Recompile and Redeploy the Web Application

Compile and (re)deploy the web application using the same steps as described here.

10.6. Test the NetUI Web Application

Visit the following link:

http://localhost:8080/netui-tutorial/myFlow/Controller.jpf

You will be directed to the `index.jsp` page.

Click the help link.

A popup window with the `help.jsp` page will be displayed.