



# JSF 2.0: Explicit Page Navigation and faces-config.xml

Originals of Slides and Source Code for Examples:  
<http://www.coreservlets.com/JSF-Tutorial/jsf2/>

**Customized Java EE Training:** <http://courses.coreservlets.com/>  
Java, JSF 2, PrimeFaces, Servlets, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android.  
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.



**For live training on JSF 2.x, please see courses at <http://courses.coreservlets.com/>.**



**Taught by the author of *Core Servlets and JSP*, *More Servlets and JSP*, and this tutorial. Available at public venues, or customized versions can be held on-site at your organization.**

- Courses developed and taught by Marty Hall
    - JSF 2, PrimeFaces, servlets/JSP, Ajax, jQuery, Android development, Java 6 or 7 programming, custom mix of topics
    - Ajax courses can concentrate on 1 library (jQuery, Prototype/Scriptaculous, Ext-JS, Dojo, etc.) or survey several
  - Courses developed and taught by coreservlets.com experts (edited by Marty)
    - Spring, Hibernate/JPA, EJB3, GWT, Hadoop, SOAP-based and RESTful Web Services
- Contact [hall@coreservlets.com](mailto:hall@coreservlets.com) for details**

# Topics in This Section

- **Explicit navigation rules**
- **Explicit bean declarations**
- **Advanced navigation options**
  - Wildcards in navigation rules
  - Conditional navigation rules
  - Dynamically computed to-ids
- **Static navigation**
- **Common navigation problems**

5

© 2012 Marty Hall

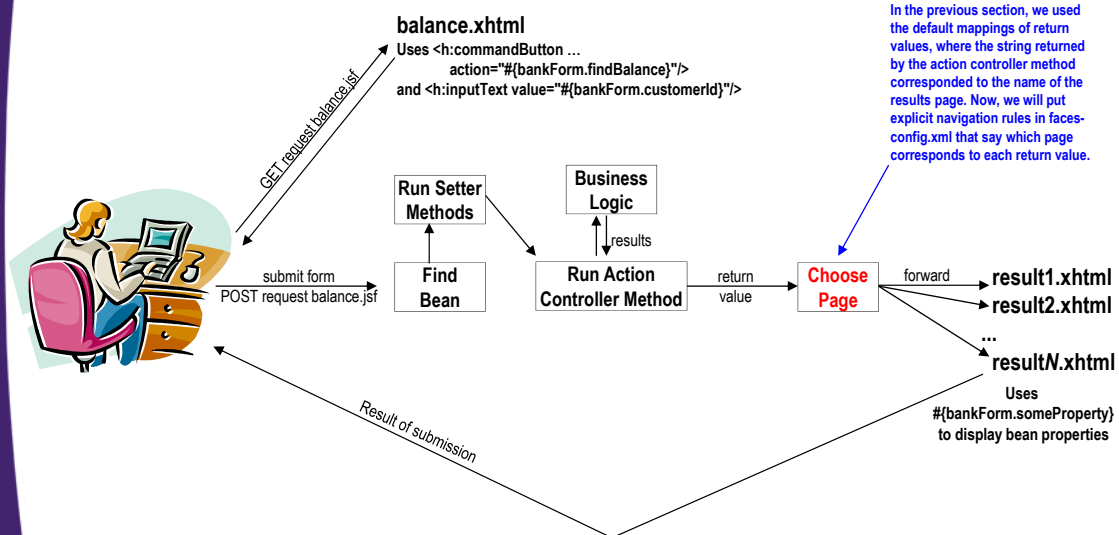


## Explicit Navigation Rules

**Customized Java EE Training: <http://courses.coreservlets.com/>**

Java, JSF 2, PrimeFaces, Servlets, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android.  
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

# JSF Flow of Control (Simplified)



7

## faces-config.xml: Overview

- **Location**
  - WEB-INF/faces-config.xml
- **Purposes**
  - Give navigation rules
    - Map return conditions to results pages
  - Declare beans
    - Map bean names to bean classes
  - Inject bean properties
  - Define locales
  - Register validators
  - Declare renderers
  - Register custom components
  - ...

8

## faces-config.xml: Syntax Summary

- **General format**

```
<?xml version="1.0"?>
<faces-config ... version="2.0">
  ...
</faces-config>
```

- **Navigation rules**

```
<navigation-rule>
  <from-view-id>/some-start-page.xhtml</from-view-id>
  <navigation-case>
    <from-outcome>return-condition-1</from-outcome>
    <to-view-id>/result-page-1.xhtml</to-view-id>
  </navigation-case>
  More navigation-case entries for other conditions
</navigation-rule>
```

9

## faces-config.xml: Syntax Details

- **General format**

```
<?xml version="1.0"?>
<faces-config xmlns="..." xmlns:xsi="..."
  xsi:schemaLocation="..." version="2.0">
  ...
</faces-config>
```

- **Notes**

- JSF 1.x faces-config.xml files are still supported, so you can run JSF 1.x applications in JSF 2.0 implementations without changes.
  - However, you must use 2.0 version of faces-config.xml if you use the new 2.0 features
- Copy a starting-point faces-config.xml file from the jsf-blank application on the Web site

10

# Navigation Rules: Syntax Details

```
<?xml version="1.0"?>
<faces-config ... version="2.0">
  <navigation-rule>
    <from-view-id>/starting-page.xhtml</from-view-id>
    <navigation-case>
      <from-outcome>return-value-1</from-outcome>
      <to-view-id>/result-page-1.xhtml</to-view-id>
    </navigation-case>
    <navigation-case>
      <from-outcome>return-value-2</from-outcome>
      <to-view-id>/result-page-2.xhtml</to-view-id>
    </navigation-case>
    ...
  </navigation-rule>
</faces-config>
```

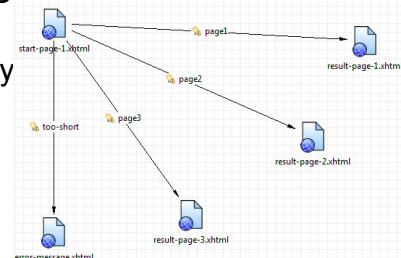
Interpretation: "If you start on such-and-such a page, press a button, and the action controller method returns such-and-such a value, go to page 1. If the action controller returns some other value, go to page 2. And so on."

Each form has one navigation-rule with a single from-view-id, but each navigation-rule can have many navigation-cases.

11

# Pros and Cons of Explicit Mappings

- **Default mappings (return values=pages)**
  - Simpler to start with. Faster to make quick test cases.
  - Redundant if return values are 1-to-1 to results pages
- **Explicit mappings in faces-config.xml**
  - Better understanding of project.
    - Lets you look in one place for navigation rules, instead of searching many Java files
      - There are even tools that automatically make charts of page flow based on faces-config.xml
  - More flexible
    - Can remap conditions to other pages later
    - Can use wildcards for starting page
    - Can use wildcards for outcomes



I find the arguments in favor of explicit mappings to be strong, and recommend using them on most real projects.

12

# Example

- **Overview**

- Collect a message, then navigate to one of three possible results pages that display the message.
- If the message is missing or only one character long, show an error page

- **Implementation**

- Action controller returns four possible strings: page1, page2, page3, and too-short
- Navigation rules in faces-config.xml map each of those return conditions to results page

13

# Example: Starting Page (start-page-1.xhtml)

```
<!DOCTYPE ... >
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html">
<h:head><title>JSF 2.0: Basic Navigation Rules</title>
...
</h:head>
<h:body>
...
<h:form>
  Your message:
  <h:inputText value="#{simpleController.message}"/>
  <br/>
  <h:commandButton value="Show Results"
                    action="#{simpleController.doNavigation}"/>
</h:form>
...
</h:body></html>
```

14

# Example: Managed Bean

```
@ManagedBean
public class SimpleController {
    private String message="";

    // getMessage and setMessage

    public String doNavigation() {
        if (message.trim().length() < 2) {
            return("too-short");
        } else {
            String[] results =
                { "page1", "page2", "page3" };
            return(RandomUtils.randomElement(results));
        }
    }
}
```

This example uses explicit navigation rules, but does not use explicit bean declarations. So, the @ManagedBean entry is needed so that the references in the facets pages to #{simpleController.blah} point at this class.

15

# Example: faces-config.xml

```
<?xml version="1.0"?>
<faces-config ...>
  <navigation-rule>
    <from-view-id>/start-page-1.xhtml</from-view-id>
    <navigation-case>
      <from-outcome>too-short</from-outcome>
      <to-view-id>/error-message.xhtml</to-view-id>
    </navigation-case>
    <navigation-case>
      <from-outcome>page1</from-outcome>
      <to-view-id>/result-page-1.xhtml</to-view-id>
    </navigation-case>
    <navigation-case>
      <from-outcome>page2</from-outcome>
      <to-view-id>/result-page-2.xhtml</to-view-id>
    </navigation-case>
    <navigation-case>
      <from-outcome>page3</from-outcome>
      <to-view-id>/result-page-3.xhtml</to-view-id>
    </navigation-case>
  </navigation-rule>
</faces-config>
```

16

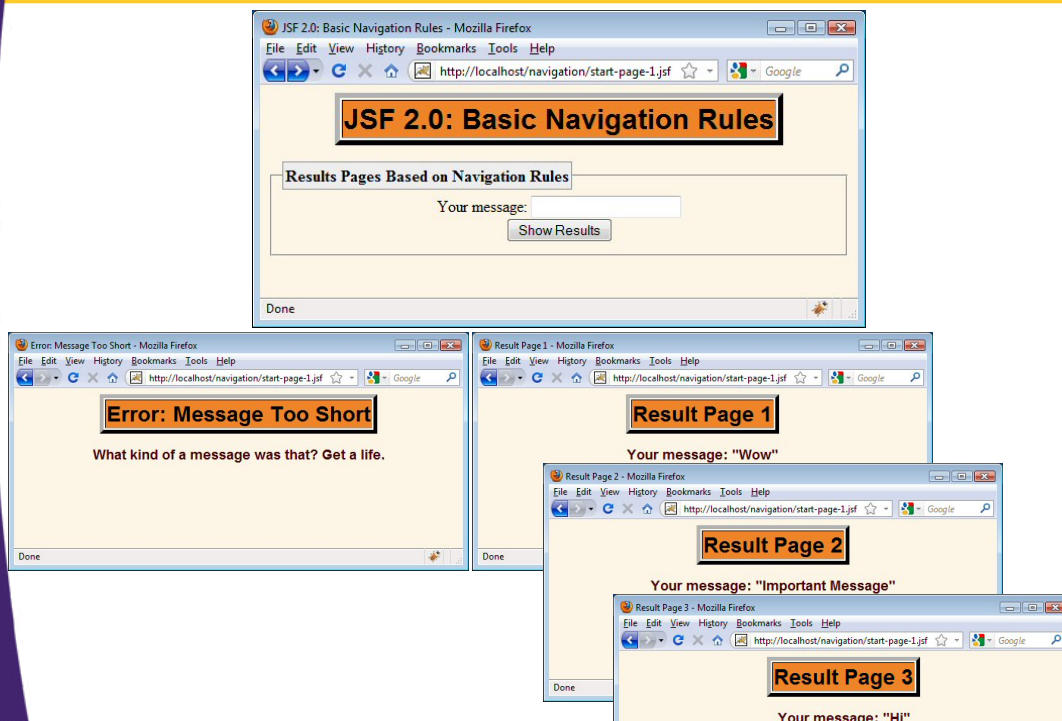
# Example: First Results Page (result-page-1.xhtml)

```
<!DOCTYPE ...>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html">
  <h:head><title>Result Page 1</title>
  <link href="./css/styles.css"
        rel="stylesheet" type="text/css"/>
</h:head>
<h:body>
  <div align="center">
    <table border="5">
      <tr><th class="title">Result Page 1</th></tr>
    </table>
    <p/>
    <h2>Your message: "#{simpleController.message}"</h2>
  </div></h:body></html>
```

Other results pages are similar.

17

# Example: Results



18

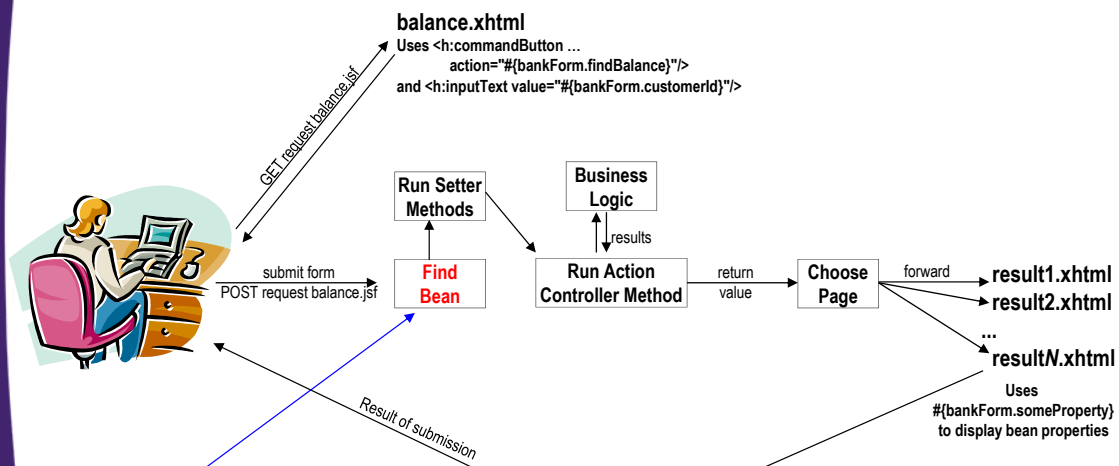


# Explicit Bean Declarations

Customized Java EE Training: <http://courses.coreservlets.com/>

Java, JSF 2, PrimeFaces, Servlets, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android.  
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

# JSF Flow of Control (Simplified)



In the previous section, we used `@ManagedBean` to give a name to the bean based on the class name (or, we used `@ManagedBean(name="beanName")`). Now, we declare the bean name explicitly in `faces-config.xml`.

# Bean Declarations: Syntax

```
<?xml version="1.0"?>
<faces-config ...>
  <managed-bean>
    <managed-bean-name>someName</managed-bean-name>
    <managed-bean-class>
      somePackage.SomeClass
    </managed-bean-class>
    <managed-bean-scope>request</managed-bean-scope>
  </managed-bean>
  ...
</faces-config>
```

Scopes are request, session, application, view, none, custom. They are discussed elsewhere. Unlike with `@ManagedBean`, there is no default, so you cannot omit `<managed-bean-scope>` altogether.

You can also define ("inject") bean properties in `faces-config`. This is discussed in separate tutorial section.

21

# Pros and Cons of Explicit Bean Declarations

- **@ManagedBean**
  - Simpler/faster to start with
  - Java developer knows the bean name
- **Explicit declarations in faces-config.xml**
  - Easier for facelets developer to find bean
    - If you have a large project with many packages, even knowing the bean class name (as with `@ManagedBean`) requires you to search many packages to find the class
  - Can use multiple instances of the same bean class in the same page. See temperature converter in first PrimeFaces lecture for example.
  - Can use same class in different pages with different scopes
  - Better understanding of beans used in project
    - One central file lists all managed beans

I find the arguments in favor of explicit bean declarations to be much less compelling than the arguments for using explicit navigation rules. So, I would recommend starting off with `@ManagedBean`, and only changing when you find a situation where the explicit definitions clearly help you. The exceptions are if you want to inject properties, if you have a large project with many packages, or if you know you will use the same bean class more than once in a page and want independent values (as with temperature converter in PrimeFaces lecture). In those cases, start with the explicit definitions from the beginning.

22

# Example

- **Overview**

- Start with form that collects a message
- Go to either an error page (message too short) or a results page (here is the message)

- **Implementation**

- Reuse the same bean as in last example
- Use managed-bean
  - Declare bean explicitly in faces-config.xml, so that the bean name used in the form comes from the config file
- Use navigation-rule
  - Remap the return conditions so that page1, page2, and page3 all point to the same results page

23

# Example: Starting Page (start-page-2.xhtml)

```
<!DOCTYPE ... >
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html">
<h:head>
...
</h:head>
<h:body>
...
<h:form>
  Your message:
  <h:inputText value="#{messageHandler.message}"/>
  <br/>
  <h:commandButton value="Show Results"
                    action="#{messageHandler.doNavigation}"/>
</h:form>
...
</h:body></html>
```

24

## Example: Managed Bean

```
package coreservlets;

public class SimpleController2
    extends SimpleController {

    // Inherits getMessage, setMessage,
    // and doNavigation. doNavigation returns
    // "too-short", "page1", "page2", or "page3".

}
```

25

## Example: faces-config.xml (Bean Declaration)

```
...
<managed-bean>
  <managed-bean-name>messageHandler</managed-bean-name>
  <managed-bean-class>
    coreservlets.SimpleController2
  </managed-bean-class>
  <managed-bean-scope>request</managed-bean-scope>
</managed-bean>
...
```

26

## Example: faces-config.xml (Navigation Rules)

```
<navigation-rule>
  <from-view-id>/start-page-2.xhtml</from-view-id>
  <navigation-case>
    <from-outcome>too-short</from-outcome>
    <to-view-id>/error-message.xhtml</to-view-id>
  </navigation-case>
  <navigation-case>
    <from-outcome>page1</from-outcome>
    <to-view-id>/message-page.xhtml</to-view-id>
  </navigation-case>
  <navigation-case>
    <from-outcome>page2</from-outcome>
    <to-view-id>/message-page.xhtml</to-view-id>
  </navigation-case>
  <navigation-case>
    <from-outcome>page3</from-outcome>
    <to-view-id>/message-page.xhtml</to-view-id>
  </navigation-case>
</navigation-rule>
```

27

## Example: Main Results Page (message-page.xhtml)

```
<!DOCTYPE ...>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html">
  <h:head><title>Message Page</title>
  <link href="./css/styles.css"
        rel="stylesheet" type="text/css"/>
</h:head>
<h:body>
  <div align="center">
  <table border="5">
    <tr><th class="title">Message Page</th></tr>
  </table>
  <p/>

  <h2>Your message: "#{messageHandler.message}"</h2>

</div></h:body></html>
```

28

# Example: Results

The image displays three overlapping browser windows from Mozilla Firefox, illustrating the results of a JSF 2.0 application. The top window, titled "JSF 2.0: Explicit Bean Declarations", shows a page with a form labeled "Bean Names Based on 'managed-bean' Entries" containing a text input field for "Your message:" and a "Show Results" button. The middle window, titled "Error: Message Too Short", displays an error message: "What kind of a message was that? Get a life." The bottom window, titled "Message Page", shows the result of a successful submission: "Your message: 'Blah blah'".

29

© 2012 Marty Hall



## Wildcards in Navigation Rules

And Other Advanced Navigation Capabilities

Customized Java EE Training: <http://courses.coreservlets.com/>

Java, JSF 2, PrimeFaces, Servlets, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android.  
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

# Summary

- **Overview**

- \* for from-view-id matches any starting page
- Omitting from-outcome: all other return conditions match
  - Except for null, which always means redisplay form

- **Application**

- \* for from-view-id
  - Multiple forms can share some of the results pages without repeating entries in navigation-rule
- Omitting from-outcome
  - Can have multiple return values point at the same results page. Useful when you want to change results pages independently of Java code
    - Temporary results pages with details coming later
    - Reusing existing Java code but different rules on results pages

31

# Example: \* for from-view-id

- **Problem**

- Both of the previous examples used the same error page
- The navigation-case was repeated

- **Solution**

- Make a shared entry that maps “too-short” to the error page

32

## faces-config.xml: Before

```
<navigation-rule>
  <from-view-id>/start-page-1.xhtml</from-view-id>
  <navigation-case>
    <from-outcome>too-short</from-outcome>
    <to-view-id>/error-message.xhtml</to-view-id>
  </navigation-case>
  <!-- Entries for page1, page2, page 3 -->
</navigation-rule>
<navigation-rule>
  <from-view-id>/start-page-2.xhtml</from-view-id>
  <navigation-case>
    <from-outcome>too-short</from-outcome>
    <to-view-id>/error-message.xhtml</to-view-id>
  </navigation-case>
  <!-- Entries for page1, page2, page 3 -->
</navigation-rule>
```

33

## faces-config.xml: After

```
<navigation-rule>
  <from-view-id>*</from-view-id>
  <navigation-case>
    <from-outcome>too-short</from-outcome>
    <to-view-id>/error-message.xhtml</to-view-id>
  </navigation-case>
</navigation-rule>
<navigation-rule>
  <from-view-id>/start-page-1.xhtml</from-view-id>
  <!-- Entries for page1, page2, page 3 -->
</navigation-rule>
<navigation-rule>
  <from-view-id>/start-page-2.xhtml</from-view-id>
  <!-- Entries for page1, page2, page 3 -->
</navigation-rule>
```

34

# Precedence for Wildcards

- **Specific from-view-id wins over wildcard**

```
<navigation-rule>
  <from-view-id*></from-view-id>
  <navigation-case>
    <from-outcome>some-result</from-outcome>
    <to-view-id>/result-page-1.xhtml</to-view-id>
  </navigation-case>
</navigation-rule>
```

```
<navigation-rule>
  <from-view-id>/some-start-page.xhtml</from-view-id>
  <navigation-case>
    <!-- No from-outcome -->
    <to-view-id>/result-page-2.xhtml</to-view-id>
  </navigation-case>
```

- The right rule takes precedence, so all results from some-start-page will be mapped to result-page-2, including "some-result".

- **Consequence**

- You cannot mix wildcards for from-view-id with defaults for outcomes (by omitting from-outcome) for same start page!

35

# Precedence for Wildcards

- **Longer from-view-id wins over shorter**

```
<navigation-rule>
  <from-view-id>/banking*</from-view-id>
  <navigation-case>
    <from-outcome>some-result</from-outcome>
    <to-view-id>/result-page-1.xhtml</to-view-id>
  </navigation-case>
</navigation-rule>
```

```
<navigation-rule>
  <from-view-id*></from-view-id>
  <navigation-case>
    <from-outcome>some-result</from-outcome>
    <to-view-id>/result-page-2.xhtml</to-view-id>
  </navigation-case>
</navigation-rule>
```

- Left rule takes precedence if both rules match.
  - For a start page of /banking/blah.xhtml and a return value of "some-result", the result page is result-page-1 (left rule), not result-page-2 (right rule).

36

## Example: Omitting from-outcome

- **Problem**

- In the second example, we reused Java code from the first example (good). But, we repeated virtually the same navigation-case three times (bad).

- **Solution**

- Omit the from-outcome. This means that any return values not explicitly mentioned are mapped to the same results page.
  - The exception is a return value of null, which always means to redisplay input form (except for the rare case where you omit from-outcome and also have an <if>). Returning null will be discussed in the section on validation.
  - **Wildcards for from-view-id will no longer apply!**

37

## faces-config.xml: Before

```
<navigation-rule>
  <from-view-id>/start-page-2.xhtml</from-view-id>
  <navigation-case>
    <from-outcome>too-short</from-outcome>
    <to-view-id>/error-message.xhtml</to-view-id>
  </navigation-case>
  <navigation-case>
    <from-outcome>page1</from-outcome>
    <to-view-id>/message-page.xhtml</to-view-id>
  </navigation-case>
  <navigation-case>
    <from-outcome>page2</from-outcome>
    <to-view-id>/message-page.xhtml</to-view-id>
  </navigation-case>
  <navigation-case>
    <from-outcome>page3</from-outcome>
    <to-view-id>/message-page.xhtml</to-view-id>
  </navigation-case>
</navigation-rule>
```

38

# faces-config.xml: After

You canNOT move this to a wildcard as we did before, because the rule below takes precedence over wildcard rules. So, without this case here, all conditions including too-short would be mapped to message-page.xhtml.

```
<navigation-rule>
  <from-view-id>/start-page-2.xhtml</from-view-id>
  <navigation-case>
    <from-outcome>too-short</from-outcome>
    <to-view-id>/error-message.xhtml</to-view-id>
  </navigation-case>
  <navigation-case>
    <to-view-id>/message-page.xhtml</to-view-id>
  </navigation-case>
</navigation-rule>
```

All conditions except for too-short are mapped to message-page.xhtml. If you knew you would do this when you designed the Java code, you would have just had a single return value and would not have needed this default mapping. But, using this default (where you omit from-outcome and all return conditions not specifically mapped in the navigation rule go to the same page) lets you change the page mappings without having to change the Java code.

39

# Conditional Navigation Rules

- **Idea**
  - You can put `<if>` tags that designate when rules apply

- **Example**

```
<navigation-case>
  <from-outcome>success</from-outcome>
  <if>#{user.returnVisitor}</if>
  <to-view-id>/welcome-back.xhtml</to-view-id>
</navigation-case>
<navigation-case>
  <from-outcome>success</from-outcome>
  <if>#{!user.returnVisitor}</if>
  <to-view-id>/welcome-aboard.xhtml</to-view-id>
</navigation-case>
```

40

# Dynamic To-Ids

- **Idea**

- You can compute the destination page directly in faces-config, rather than indirectly via return value of the action controller method

- **Example**

```
<navigation-rule>  
  <from-view-id>/exam-question.xhtml</from-view-id>  
  <navigation-case>  
    <to-view-id>#{exam.nextQuestionPage}</to-view-id>  
  </navigation-case>  
</navigation-rule>
```

41

© 2012 Marty Hall



## Static Navigation

Customized Java EE Training: <http://courses.coreservlets.com/>

Java, JSF 2, PrimeFaces, Servlets, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android.  
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

## Idea

- **Situation**

- Sometimes you don't want/need Java code
  - No input elements
  - No logic needed to determine results page
  - Sometimes used for simple testing. But also used in real projects when you want a button for navigation.

- **Approach**

- Instead of this
  - `<h:commandButton ... action="#{someBean.doNav}"/>`
  - Have the doNav method always return "fixed-page"
- Use this
  - `<h:commandButton ... action="fixed-page"/>`
    - You can use either default mapping or explicit navigation rules to determine the meaning of "fixed-page"

43

## Example

- **Overview**

- page-a.xhtml has a button that causes navigation to page-b.xhtml
- page-b.xhtml has a button that causes navigation to page-a.xhtml

- **Implementation**

- Page A
  - `<h:commandButton ... action="page-b"/>`
- Page B
  - `<h:commandButton ... action="page-a"/>`

44

## Example: page-a.xhtml

```
<!DOCTYPE ... >
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html">
<h:head><title>JSF 2.0: Static Navigation</title>
...
</h:head>
<h:body>
<div align="center">
<table border="5">
  <tr><th class="title">Page A</th></tr>
</table>
<br/>
<h:form>
  <h:commandButton value="Go to Page B"
                    action="page-b"/>
</h:form>
</div></h:body></html>
```

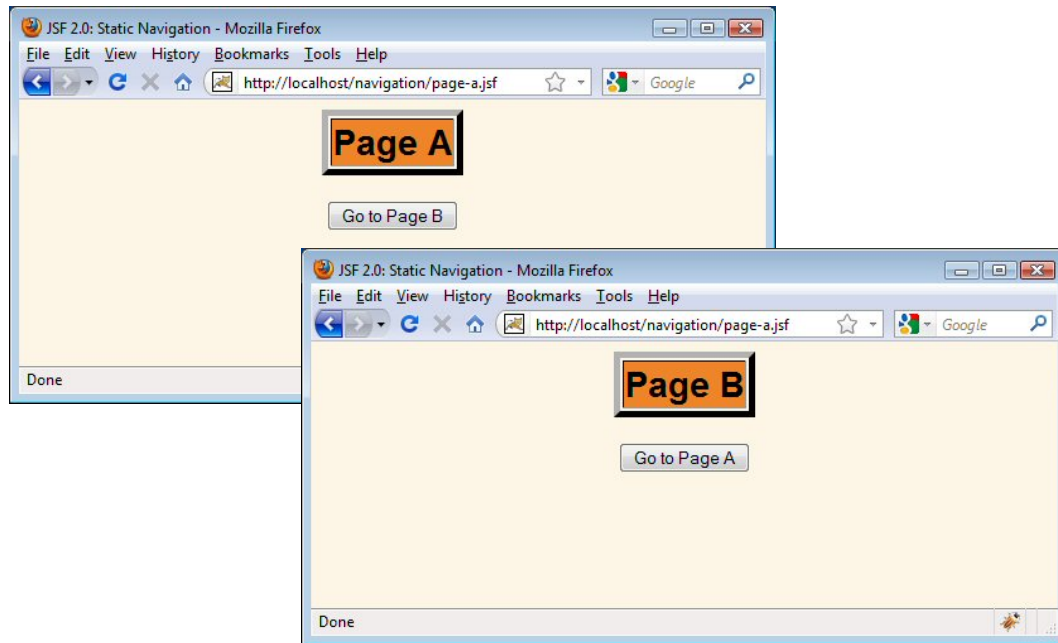
45

## Example: page-b.xhtml

```
<!DOCTYPE ... >
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html">
<h:head><title>JSF 2.0: Static Navigation</title>
...
</h:head>
<h:body>
<div align="center">
<table border="5">
  <tr><th class="title">Page B</th></tr>
</table>
<br/>
<h:form>
  <h:commandButton value="Go to Page A"
                    action="page-a"/>
</h:form>
</div></h:body></html>
```

46

# Example: Results



47

© 2012 Marty Hall



## Common Navigation Problems

Customized Java EE Training: <http://courses.coreservlets.com/>

Java, JSF 2, PrimeFaces, Servlets, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android.  
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

# Pressing Button and Nothing Happens

- **Issue**

- Many error conditions simply result in the system redisplaying the form with no warning or error messages

- **Debugging strategies**

- Set **PROJECT\_STAGE** to **Development** in **web.xml**
  - This is already set in jsf-blank
- Many of the errors cause the process to abort at certain points. Knowing how far things got is very helpful.
  - Use print statements or IDE breakpoints
  - Put a print statement in the action controller method
  - Put a print statement in the empty constructor
    - `public MyBean() { System.out.println("MyBean built"); }`
    - Bean should be instantiated *twice* for request scope
  - Put print statements in the bean setter methods

49

# Pressing Button and Nothing Happens: Common Cases

1. **Return value of controller method does not match from-outcome of navigation-case**

- Remember values are case sensitive

2. **Using from-action instead of from-outcome**

```
<navigation-case>  
  <from-action>accepted</from-action>  
  <to-view-id>/accept-registration.jsp</to-view-id>  
</navigation-case>
```

Should be from-outcome, not from-action

- This is really a special case of (1), since there is now *no* from-outcome
- This situation occurs frequently with Eclipse users that don't look carefully at the choices Eclipse offers in popup menu for the navigation-case entries.

50

## Pressing Button and Nothing Happens: Common Cases

### 3. Forgetting # in action of h:commandButton

```
<h:commandButton  
  value="Button Label"  
  action="{beanName.methodName}"/>
```

Should have # here

- This is really a special case of (1), since `action="{beanName.methodName}"` means the literal string `"{beanName.methodName}"` is the from-outcome
  - In this situation and several others, it is very helpful to put a print statement in controller method to see if/when it is invoked

### 4. Typo in from-view-id

- This is a special case of (1), since the from-outcome applies to nonexistent page

51

## Pressing Button and Nothing Happens: Common Cases

### 5. Controller method returns null

- This is often done on purpose to redisplay the form, but can be done accidentally as well.

### 6. Type conversion error

- You declare field to be of type `int`, but value is not an integer when you submit.
  - Behavior of redisplaying form is useful here. See validation section.

### 7. Missing setter method

- You associate textfield with bean property `foo`, but there is no `setFoo` method in your bean.
  - Debugging hint: You will see printout for bean being instantiated, but not for controller method

### 8. Missing h:form

- If you use `h:inputText` with no surrounding `h:form`, textfields will still appear but nothing will happen when you press submit button

52



# Wrap-Up

**Customized Java EE Training: <http://courses.coreservlets.com/>**

Java, JSF 2, PrimeFaces, Servlets, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android.  
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

## Summary

- **Explicit navigation rules**
  - Compared to default mappings, this is more flexible and lets you understand system page flow more easily
- **Explicit bean declarations**
  - It is less clear whether this is better than `@ManagedBean`, but in large projects with many packages, this might let you find beans more easily
- **Wildcards in navigation rules**
  - \* in from-view-id lets you share results pages across forms
  - Omitting from-outcome lets you map different return values to same results page.
    - Never mix both approaches for the same start page!
- **Static navigation**
  - For simple testing, you can use a static string (instead of an EL expression) for the action of `h:commandButton`
- **Navigation problems**
  - Many situations result in JSF redisplaying input form
    - Most due to return value of Java code not matching from-outcome
  - Debugging: set `PROJECT_STAGE` to Development, trace Java code



# Questions?

**Customized Java EE Training: <http://courses.coreservlets.com/>**

Java, JSF 2, PrimeFaces, Servlets, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android.  
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.