

First step with JavaServer Faces using Eclipse

This tutorial facilitates the first steps with the quite new framework JavaServer Faces (JSF). Step by step an example application (a library) will be created, which illustrate the different elements of the framework.

The example application will provide the following functionality.

- Display a book overview (list of books)
- Add, edit and delete a book

Generals

Author:

Sascha Wolski

<http://www.laliluna.de/tutorials.html> Tutorials for Struts, EJB, xdoclet, JSF, JSP and eclipse.

Date:

December, 21 2004

Source code:

The sources do not contain any project files of eclipse or libraries. Create a new project following the tutorial, add the libraries as explained in the tutorial and then you can copy the sources in your new project.

<http://www.laliluna.de/assets/tutorials/first-java-server-faces-tutorial.zip>

PDF Version des Tutorials:

<http://www.laliluna.de/assets/tutorials/first-java-server-faces-tutorial-en.pdf>

Development Tools

Eclipse 3.x

MyEclipse plugin 3.8

(A cheap and quite powerful Extension to Eclipse to develop Web Applications and EJB (J2EE) Applications. I think that there is a test version available at MyEclipse.)

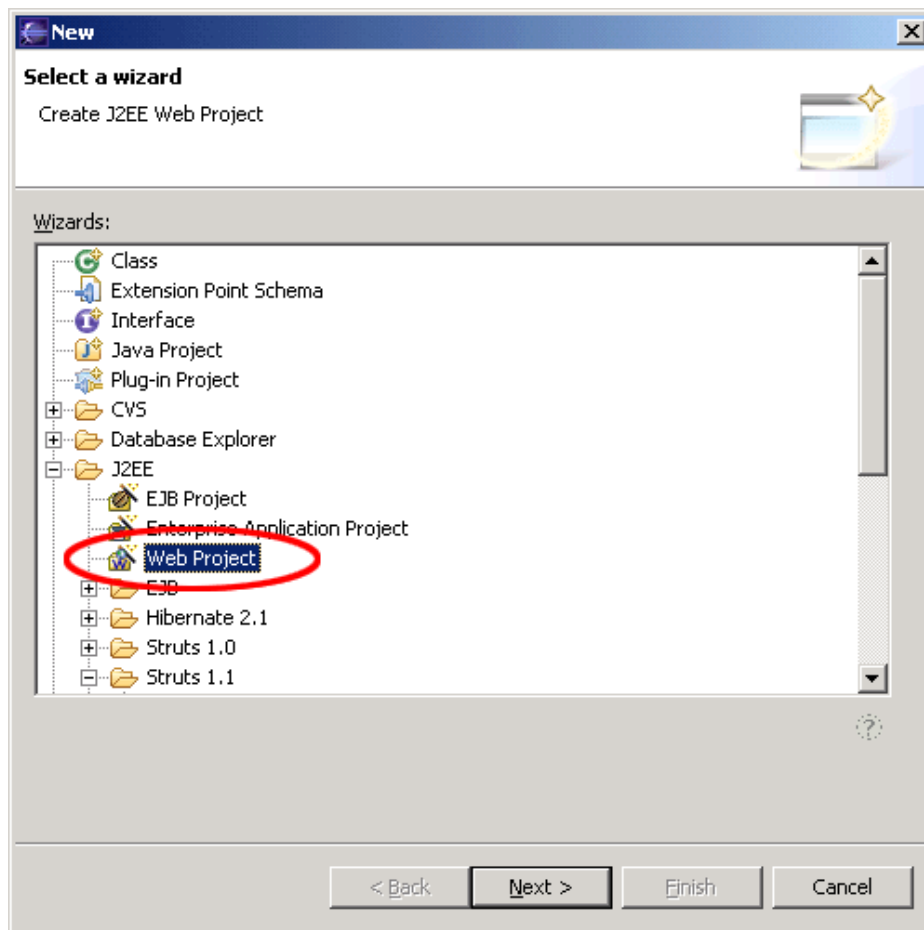
Application Server

Jboss 3.2.5

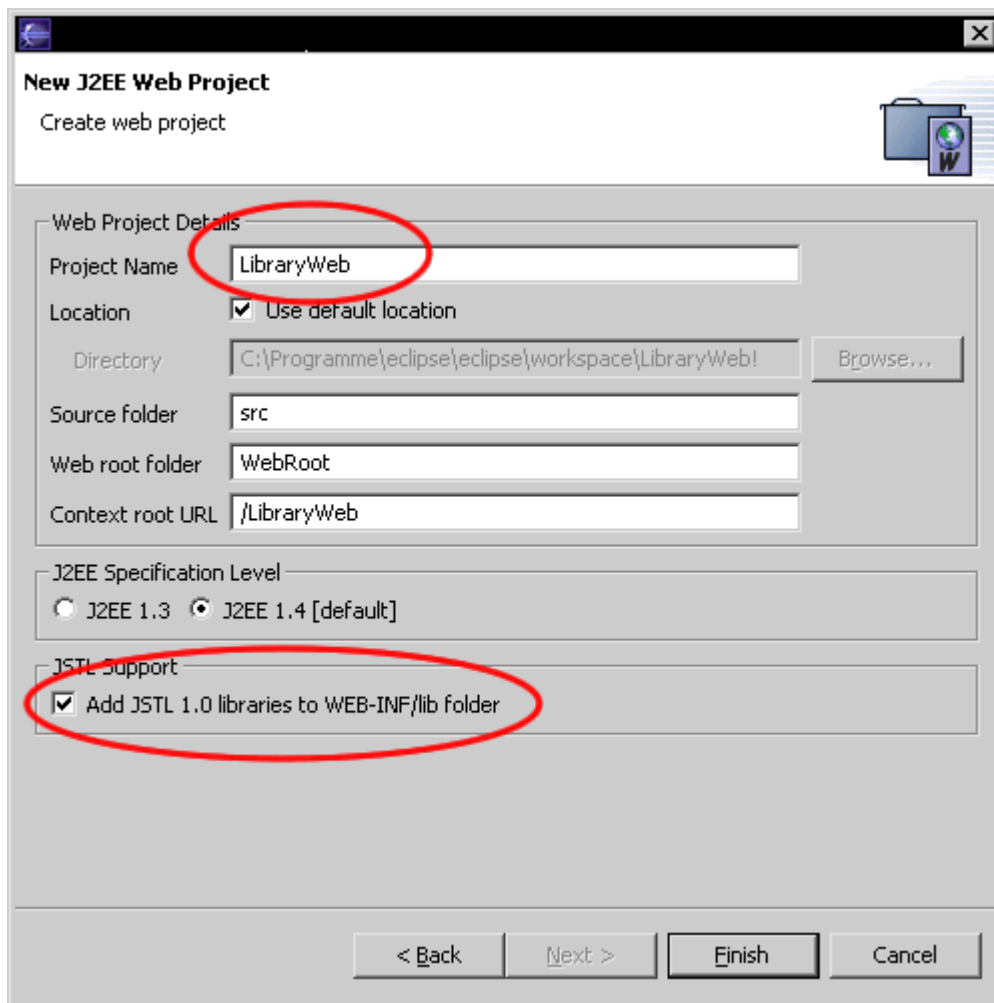
You may use Tomcat here if you like.

Create a JavaServer faces project

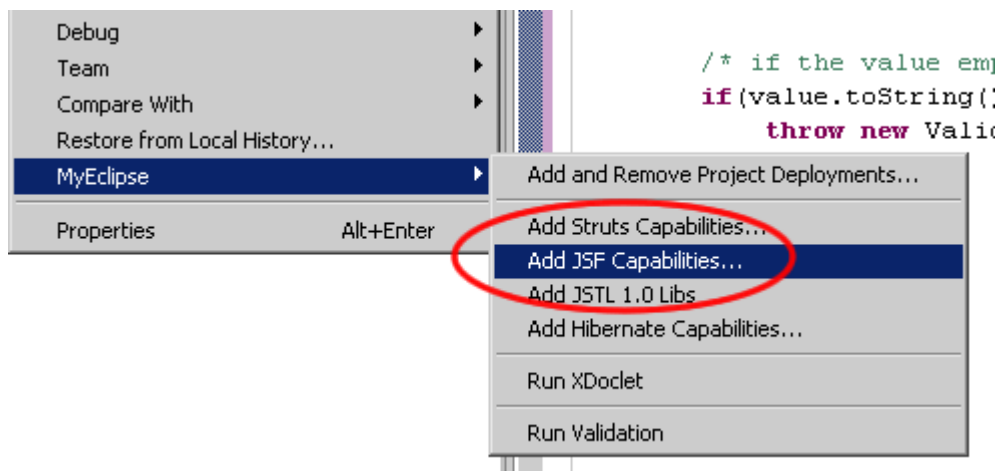
Create a new web project. `File > New > Project.`



Set a nice name and add the JSTL Libraries to the project.




Add the JavaServer Faces Capabilities. Right click on the project and choose MyEclipse > Add JSF Capabilities.



New ✕

JavaServer Faces Support for MyEclipse Web Project

Enable project for JavaServer Faces development 

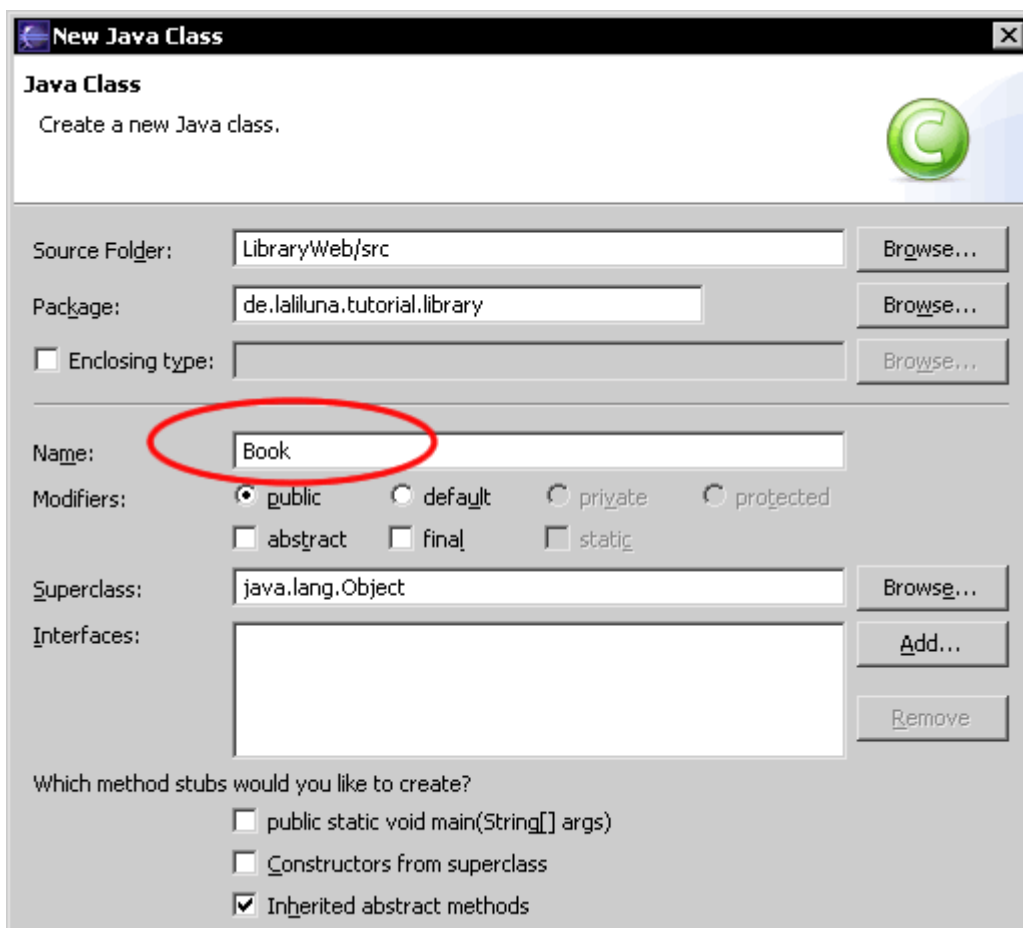
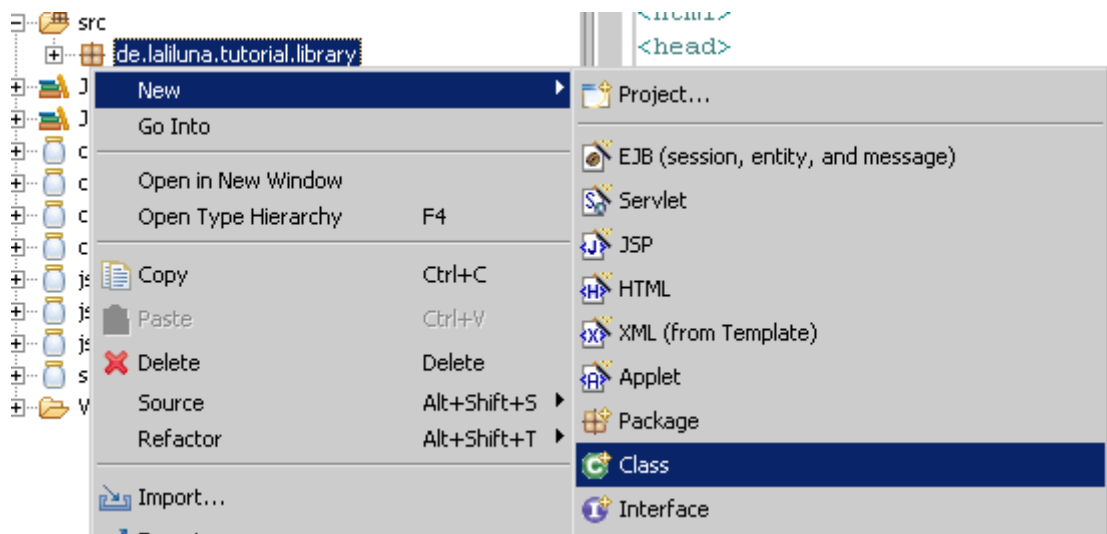
Web project: LibraryWeb
Web-root folder: /WebRoot
Servlet specification: 2.4
JSF specification: 1.1

JSF config path:
Faces servlet name:
URL pattern *.faces /faces/*

Install JSF jars Install JSF TLDs

The class Book

Add an new package `de.laliluna.tutorial.library` und create a new class named `Book`.

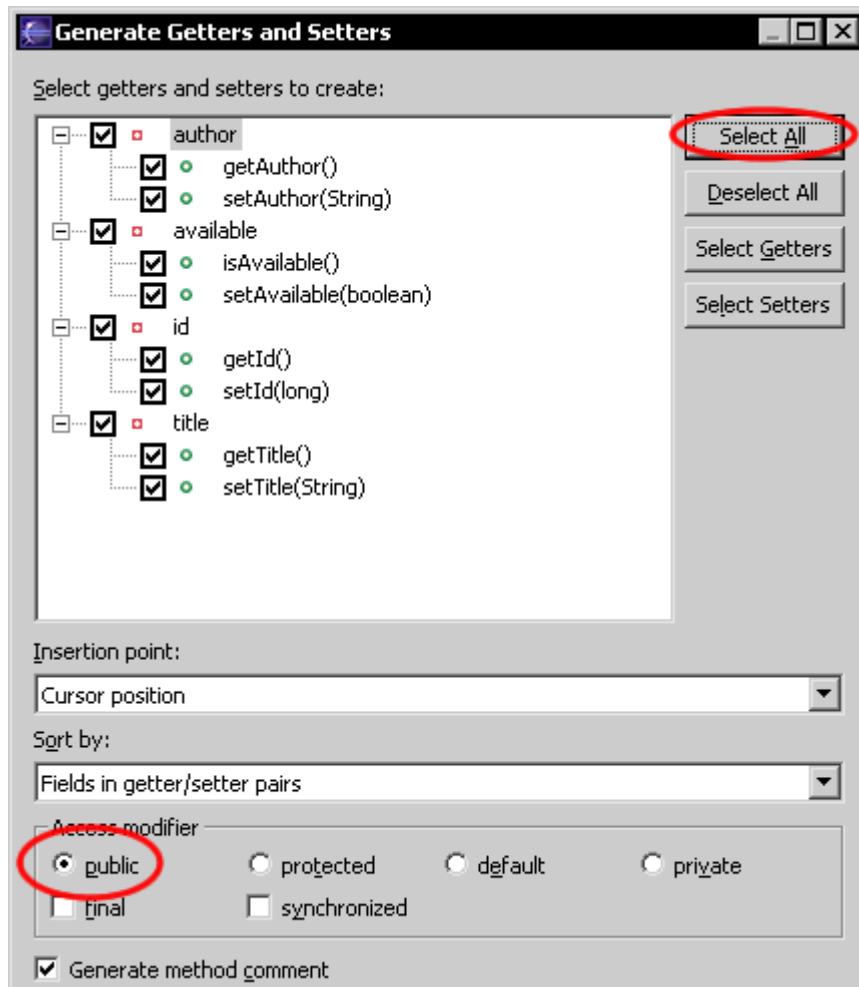


Open the class and add the following private properties:

- id
- author
- title
- available

Generate a getter- and setter-method for each property. Right click on the editor window and

choose Source > Generate Getter- and Setter Methods.



Furthermore you have to add a constructor, which set the properties if you initialize the instance variable of the newly object.

The following source code show the class book.

```
public class Book implements Serializable {

    // ----- Properties -----
    private long id;
    private String author;
    private String title;
    private boolean available;

    // ----- Constructors -----
    public Book(){}
    public Book(long id, String author, String title, boolean available){
        this.id = id;
        this.author = author;
        this.title = title;
        this.available = available;
    }

    // ----- Getter and setter methods -----

    public String getAuthor() {
        return author;
    }
    public void setAuthor(String author) {
```

```

        this.author = author;
    }
    public boolean isAvailable() {
        return available;
    }
    public void setAvailable(boolean available) {
        this.available = available;
    }
    public long getId() {
        return id;
    }
    public void setId(long id) {
        this.id = id;
    }
    public String getTitle() {
        return title;
    }
    public void setTitle(String title) {
        this.title = title;
    }
}

```

Add a getter and setter for the class.

```

/**
 * Set the properties
 * @param book
 */
public void setBook(Book book) {
    this.setId(book.getId());
    this.setAuthor(book.getAuthor());
    this.setTitle(book.getTitle());
    this.setAvailable(book.isAvailable());
}

/**
 * @return book object
 */
public Book getBook() {
    return new Book(this.getId(),
                    this.getAuthor(),
                    this.getTitle(),
                    this.isAvailable());
}

```

The database class

We use a class to provide some test data without using a database. Download the sample application of this tutorial and copy the class `SimulateDB.java` find in the folder `src/de/laliluna/tutorial/library/` in the package `de.laliluna.tutorial.library`.

The class BookList

Create a futher class `BookList` in the package `de.laliluna.library`. This class includes the property books, which represent the list of books. Generate a getter- and seter-method for the property books and change the getter-method like the following.

```

public class BookList {

    // ----- Properties -----
    Collection books;

    // ----- Getter and Setter -----
}

```

```

/**
 * @return collection of books
 */
public Collection getBooks(){

    SimulateDB simulateDB = new SimulateDB();

    /* Holt sich die Session auf dem Externen Context
    */
    Map session = FacesContext.getCurrentInstance().getExternalContext
().getSessionMap();

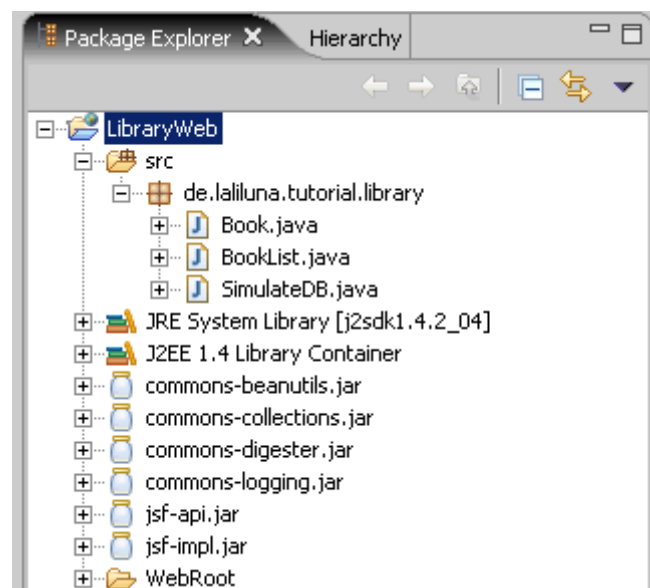
    /* Lies alle Bücher auf der simulierten Datenbank aus
    */
    books = simulateDB.getAllBooks(session);

    return books;
}

/**
 * @param books The books to set.
 */
public void setBooks(Collection books) {
    this.books = books;
}
}

```

Your package explorer will look like the picture below.



Action listener methods

To provide that a user can add, edit or delete a book, we have to include the appropriate functionality. This functionality will be implemented in action listener methods / classes. If an event occur (ex.: a user clicks on a link) an action listener method / class will be called and processed.

Open the class Book and add four methods , which process the following functionality.

- Initialize a book
- Edit a book
- Save a book
- Delete a book

Initialize a book

```
/**
 * Initial the properties of the class with null
 * @param event
 */
public void initBook(ActionEvent event){

    /*
     * init the book object
     */
    this.setBook(new Book());
}
```

Edit a book

```
/**
 * Get the book to edit and assign it to the bean
 *
 * @param event
 */
public void selectBook(ActionEvent event){

    SimulatedDB simulatedDB = new SimulatedDB();

    /*
     * Get the session map of the external context
     */
    Map session = FacesContext.getCurrentInstance().getExternalContext().
getSessionMap();

    /*
     * Find the UIParameter component by expression
     */
    UIParameter component = (UIParameter) event.getComponent().findComponent
("editId");

    /*
     * parse the value of the UIParameter component
     */
    long id = Long.parseLong(component.getValue().toString());

    /*
     * get the book by id and set it in the local property
     */
    this.setBook(simulatedDB.loadBookById(id, session));
}
```

Save a book

```
/**
 * Add or update the book in the simulated database.
 * If the book id is not set the book will be added
 * otherwise the book will be updated
 *
 * @param event
 */
public void saveBook(ActionEvent event){

    SimulatedDB simulatedDB = new SimulatedDB();

    /*
     * Get the session map of the external context
     */
    Map session = FacesContext.getCurrentInstance().getExternalContext().
getSessionMap();
```

```
    /*
     * Add or update the book in the simulated database
     */
    simulateDB.saveToDB(this.getBook(), session);
}
```

Delete a book

```
/**
 * Delete a book in the simulated database
 *
 * @param event
 */
public void deleteBook(ActionEvent event){

    SimulateDB simulateDB = new SimulateDB();

    /*
     * Get the session map of the external context
     */
    Map session = FacesContext.getCurrentInstance().getExternalContext().
getSessionMap();

    /*
     * Find the UIParameter component by expression
     */
    UIParameter component = (UIParameter) event.getComponent().findComponent
("deleteId");

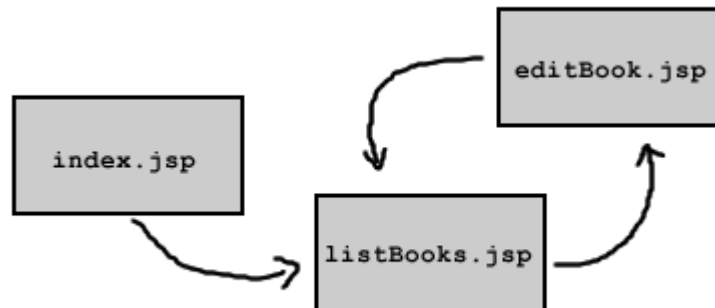
    /*
     * parse the value of the UIParameter component
     */
    long id = Long.parseLong(component.getValue().toString());

    /*
     * delete the book by id
     */
    simulateDB.deleteBookById(id, session);
}
```

The file faces-config.xml

The faces-config.xml is the central configuration file of JavaServer faces. In this file you define the workflow of the application (on which action which site will be processed) , the managed bean classes by JSF and something more.

The workflow of the library application looks like the following.



We define a navigation rule for this workflow.

Open the file faces-config.xml and add the following configuration.

```
<faces-config>

  <!-- Navigation rules -->
  <navigation-rule>
    <description>List of books</description>
    <from-view-id>/listBooks.jsp</from-view-id>
    <navigation-case>
      <from-outcome>editBook</from-outcome>
      <to-view-id>/editBook.jsp</to-view-id>
    </navigation-case>
  </navigation-rule>

  <navigation-rule>
    <description>Add or edit a book</description>
    <from-view-id>/editBook.jsp</from-view-id>
    <navigation-case>
      <from-outcome>listBooks</from-outcome>
      <to-view-id>/listBooks.jsp</to-view-id>
      <redirect/>
    </navigation-case>
  </navigation-rule>
</faces-config>
```

```
<navigation-rule>
```

Define a navigation rule

```
<from-view-id>/listBooks.jsp</from-view-id>
```

Define the jsp file for which the containing navigation rule is relevant.

```
<navigation-case>
```

Define a navigation case

```
<from-outcome>editBook</from-outcome>
```

Define a name for this navigation case

```
<to-view-id>/listBooks.jsp</to-view-id>
```

Refers to the setted JSP File

```
<redirect/>
```

All parameters saved in the request will be losed when you set this tag.

If you want to access the bean classes in your JSP files, you have to register the bean classes in `faces-config.xml`

Add the following source code.

```
<!-- Managed beans -->
<managed-bean>
  <description>
    Book bean
  </description>
  <managed-bean-name>bookBean</managed-bean-name>
  <managed-bean-class>de.laliluna.tutorial.library.Book</managed-bean-class>
  <managed-bean-scope>request</managed-bean-scope>
</managed-bean>

<managed-bean>
  <description>
    BookList Bean
  </description>
  <managed-bean-name>bookListBean</managed-bean-name>
  <managed-bean-class>de.laliluna.tutorial.library.BookList</managed-bean-
class>
  <managed-bean-scope>session</managed-bean-scope>
</managed-bean>
```

```
<managed-bean>
```

Define a managed bean

```
<managed-bean-name>bookBean</managed-bean-name>
```

Define a name for the managed bean. This name is used in the JSP file.

```
<managed-bean-class>de.laliluna.tutorial.library.Book</managed-bean-class>
```

Define the class which represent the bean.

```
<managed-bean-scope>request</managed-bean-scope>
```

Define in which scope the bean is saved.

Create the JSP files

In the first step we create a JSP file named `index.jsp`, which forwards the user to the list of books.

index.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <body>
    <jsp:forward page="/listBooks.faces" />
  </body>
</html>
```

In the second step we create the book overview.

listBooks.jsp

```
<%@ page language="java" %>
<%@ taglib uri="http://java.sun.com/jsp/html" prefix="h" %>
<%@ taglib uri="http://java.sun.com/jsp/core" prefix="f" %>

<%
String path = request.getContextPath();
String basePath = request.getScheme()+"://"+request.getServerName()+
"+"+request.getServerPort()+path+"/";
%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
```

```

<head>
  <base href="<%=basePath%>">

  <title>List of books</title>
</head>

<body>
  <f:view>
    <h:form id="bookList">
      <h:dataTable id="books"
                  value="#{bookListBean.books}"
                  var="book"
                  border="1">

        <h:column>
          <f:facet name="header">
            <h:outputText value="Author"/>
          </f:facet>
          <h:outputText value="#{book.author}" />
        </h:column>
        <h:column>
          <f:facet name="header">
            <h:outputText value="Title"/>
          </f:facet>
          <h:outputText value="#{book.title}" />
        </h:column>
        <h:column>
          <f:facet name="header">
            <h:outputText value="Available"/>
          </f:facet>
          <h:selectBooleanCheckbox disabled="true"
                                value="#{book.available}" />
        </h:column>
        <h:column>
          <f:facet name="header">
            <h:outputText value="Edit"/>
          </f:facet>
          <h:commandLink id="Edit"
                        action="editBook"
                        actionListener="#{bookBean.selectBook}">
            <h:outputText value="Edit" />
            <f:param id="editId"
                    name="id"
                    value="#{book.id}" />
          </h:commandLink>
        </h:column>
        <h:column>
          <f:facet name="header">
            <h:outputText value="Delete"/>
          </f:facet>
          <h:commandLink id="Delete"
                        action="listBooks"
                        actionListener="#{bookBean.deleteBook}">
            <h:outputText value="Delete" />
            <f:param id="deleteId"
                    name="id"
                    value="#{book.id}" />
          </h:commandLink>
        </h:column>
      </h:dataTable>

      <h:commandLink id="Add"
                    action="editBook"
                    actionListener="#{bookBean.initBook}">
        <h:outputText value="Add a book" />
      </h:commandLink>
    </h:form>
  </f:view>
</body>
</html>

```

```
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
```

With directive `taglib` we include the JSF tag libraries

```
<f:view>
```

Renders a view component. All others tags must be included within this tag.

```
<h:form id="bookList">
```

Renders a HTML form.

```
<h:dataTable id="books" value="#{bookListBean.books}" var="book" border="1">
```

Renders a HTML table. The tag is used to loop over a list of data like a `for` loop. The parameter `value` assign a list of data, in our case the list of books of the library. With the parameter `var` you define the variable used to access a element (a book) of the list within the tag (loop).

```
<h:column>
  <f:facet name="header">
    <h:outputText value="Author"/>
  </f:facet>
  <h:outputText value="#{book.author}" />
</h:column>
```

Renders a column with a column header.

`<f:facet name="header">` display the header.

`<h:outputText value="Author"/>` print out a header label.

`<h:outputText value="#{book.author}" />` refers to the property `author` of the current element of the list.

```
<h:commandLink id="Edit"
  action="editBook"
  actionListener="#{bookBean.selectBook}">
```

Renders a HTML link, which submits the form. The parameter `action` define the navigation case, which is processed, after the form submits In our case the navigation case `editBook`, we have added before in the `faces-config.xml`, will be processed. We assign the action listener method to the link with the parameter `actionListener`. After the user submit the form the method will be processed.

The last JSP file includes a form to add and edit a book.

editBook.jsp

```
<%@ page language="java" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>

<%
String path = request.getContextPath();
String basePath = request.getScheme()+"://"+request.getServerName()+
":"+request.getServerPort()+path+"/";
%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
  <base href="<%=basePath%>">

  <title>Add / Edit a book</title>
</head>

<body>
  <f:view>
    <h:form>
      <h:inputHidden id="id" value="#{bookBean.id}"/>
```

```

<h:panelGrid columns="2" border="1">

    <h:outputText value="Author:" />
    <h:inputText id="author"
        value="#{bookBean.author}" />
</h:inputText>

    <h:outputText value="Title:" />
    <h:inputText id="title"
        value="#{bookBean.title}" />
</h:inputText>

    <h:outputText value="Available:" />
    <h:selectBooleanCheckbox id="available"
        value="#{bookBean.available}" />

</h:panelGrid>

<h:commandButton value="Save"
    action="listBooks"
    actionListener="#{bookBean.saveBook}" />

</h:form>
</f:view>
</body>
</html>

```

```
<h:inputHidden id="id" value="#{bookBean.id}" />
```

Renders a HTML hidden element. Value refers to the managed bean `bookBean` and its property `id`, which indicated in the `faces-config.xml`.

```
<h:panelGrid columns="2" border="1">
```

Renders a HTML table with two columns.

```
<h:inputText id="author" value="#{bookBean.author}" />
```

Renders a HTML text field. Value refers to the property `author` of our class `Book`.

```

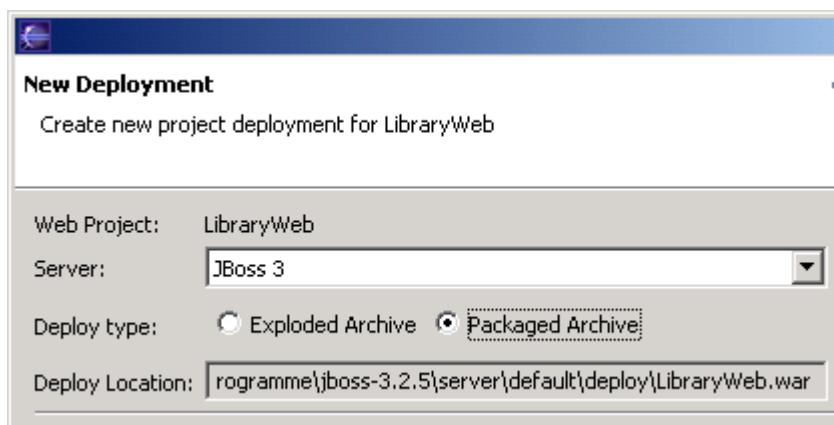
<h:commandButton value="Save"
    action="listBooks"
    actionListener="#{bookBean.saveBook}" />

```

Renders a HTML submit button with the value `save` and the action `listBooks`. The action listener method `saveBook` will be processed if the user submit the form.

Test the application

Start the jboss and deploy the project as `Packaged Archive`.



Call the project now <http://localhost:8080/LibraryWeb/>