JSF Tools Reference Guide

Version: 3.2.1.GA

1. Introduction 1
1.1. Key Features of JSF Tools 1
1.2. Other relevant resources on the topic 2
2. JavaServer Faces Support 3
2.1. Facelets Support 4
2.1.1. Facelets templates 4
2.1.2. Facelets components 6
2.1.3. Code assist for Facelets 6
2.1.4. Open On feature 8
3. Projects 11
3.1. Creating a New JSF Project 11
3.2. Importing Existing JSF Projects with Any Structure 17
3.3. Adding JSF Capability to Any Existing Eclipse Project 17
3.4. Adding Your Own Project Templates 23
3.5. Relevant Resources Links 25
4. Web.xml Editor 27
5. JSF Configuration File Editor 29
5.1. Diagram view 29
5.2. Tree View 34
5.3. Source View 52
5.4. Editor Features 54
5.4.1. Open On 54
5.4.2. Code Assist 54
5.4.3. Error Reporting 56
6. Managed Beans 59
6.1. Code Generation for Managed Beans 59
6.2. Add Existing Java Beans to a JSF Configuration File 68
7. Creation and Registration 71
7.1. Create and Register a Custom Converter 71
7.2. Create and Register a Custom Validator 78
7.3. Create and Register Referenced Beans 86
8. JSF Project Verification 95

Introduction

JSF Tools are especially designed to support JSF and JSF-related technologies. JSF Tools provide extensible tools for building JSF-based applications as well as adding JSF capabilities to existing web projects, importing JSF projects and choosing any JSF implementation while developing JSF application.

This guide provides the information on JSF tooling you need to allow you to quickly develop JSF applications with far fewer errors.

1.1. Key Features of JSF Tools

The table below lists the functionality provided by the JSF Tools.

Table 1.1. Key Functionality for JSF Tools

Feature	Benefit	Chapter
JSF and Facelets support	Step-by-step wizards for creating new JSF and Facelets projects with a number of predefined templates, importing existing ones and adding JSF capabilities to non-JSF web projects.	*Chapter 2, JavaServer Faces Support*
Flexible and customizable project template management	Jump-start development with the supplied templates or easily create customized templates for re-use.	*Chapter 3, Projects*
Support for JSF Configuration File	Work on a file using three modes: diagram, tree and source. Automatic synchronization between the modes and full control over the code. Easily move around the diagram using the Diagram Navigator.	*Chapter 5, JSF Configuration File Editor*
Support for Managed Beans	Adding new managed beans, generating code for attributes, properties and getter/setter methods.	*Chapter 6, Managed Beans*
Support for Custom Converters and Validators	Fast creation of custom converters and validators with a tree view of the `faces-config.xml` file.	*Chapter 7, Creation and Registration*
Verification and Validation	All errors will be immediately reported by verification feature, no matter in what view you are working. Constant validation and error checking allows you to catch many of the errors during development process that significantly reduces development time.	*Chapter 8, JSF Project Verification*

1.2. Other relevant resources on the topic

All JBoss Developer Studio and JBoss Tools release documentation can be found on the *RedHat Documentation* [http://docs.redhat.com/docs/en-US/JBoss_Developer_Studio/index.html] website.

Nightly documentation builds are available at *http://download.jboss.org/jbosstools/nightly-docs* [http://download.jboss.org/jbosstools/nightly-docs/].

JavaServer Faces Support

JSF Tools does not lock you into any one JavaServer Faces implementation. You can always specify the desired JavaServer Faces implementation while creating a new JSF project (see *Section 3.1, "Creating a New JSF Project"*), adding JSF capability to any existing Eclipse project (see *Section 3.3, "Adding JSF Capability to Any Existing Eclipse Project"*), or importing existing JSF projects (see *Section 3.2, "Importing Existing JSF Projects with Any Structure"*).

At this point the special wizard will prompt you to specify an appropriate JSF environment. It may be JSF 1.1.02 RI, JSF 1.2 or JSF 2.0. The wizard also lets you select JSF implementation with a component orientation such as JSF 1.2 with Facelets or MyFaces 1.1.4.

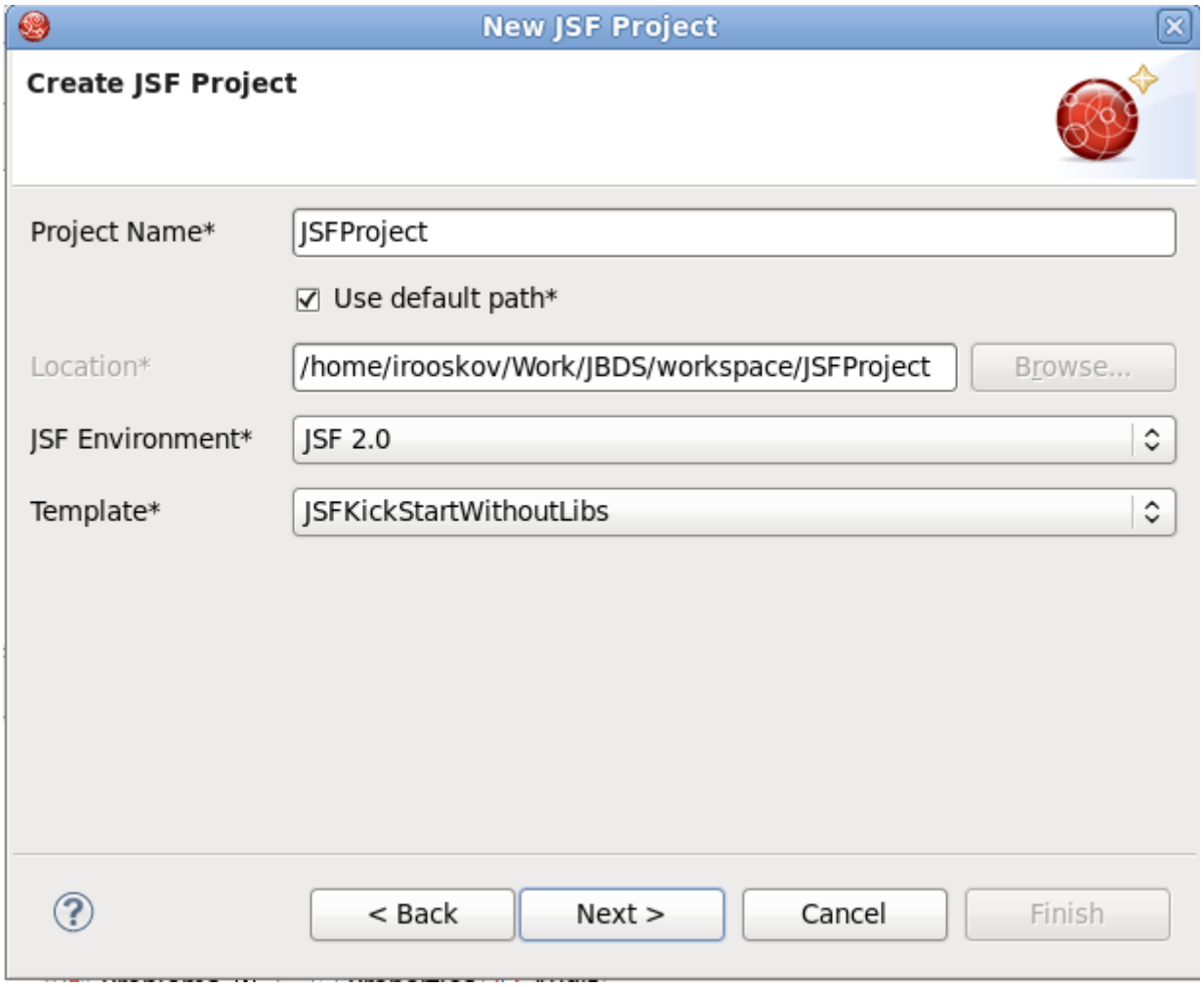

Figure 2.1. Choosing JSF Environment

After specifying an appropriate JSF environment, all the required libraries associated with the selected version will be added to your project.

2.1. Facelets Support

In this section we will focus all the concepts that relate to working with Facelets.

Facelets extend JavaServer Faces by providing a lightweight framework that radically simplifies the design of JSF presentation pages. Facelets can be used in a variety of ways that we will consider further in this section.

2.1.1. Facelets templates

If you want to build an application using Facelets, create a project with Facelets based on version 1.2 of the JSF Reference Implementation, i. e. select the **JSF 1.2 with Facelets** option in the **JSF Environment** section of the **New JSF Project** wizard.

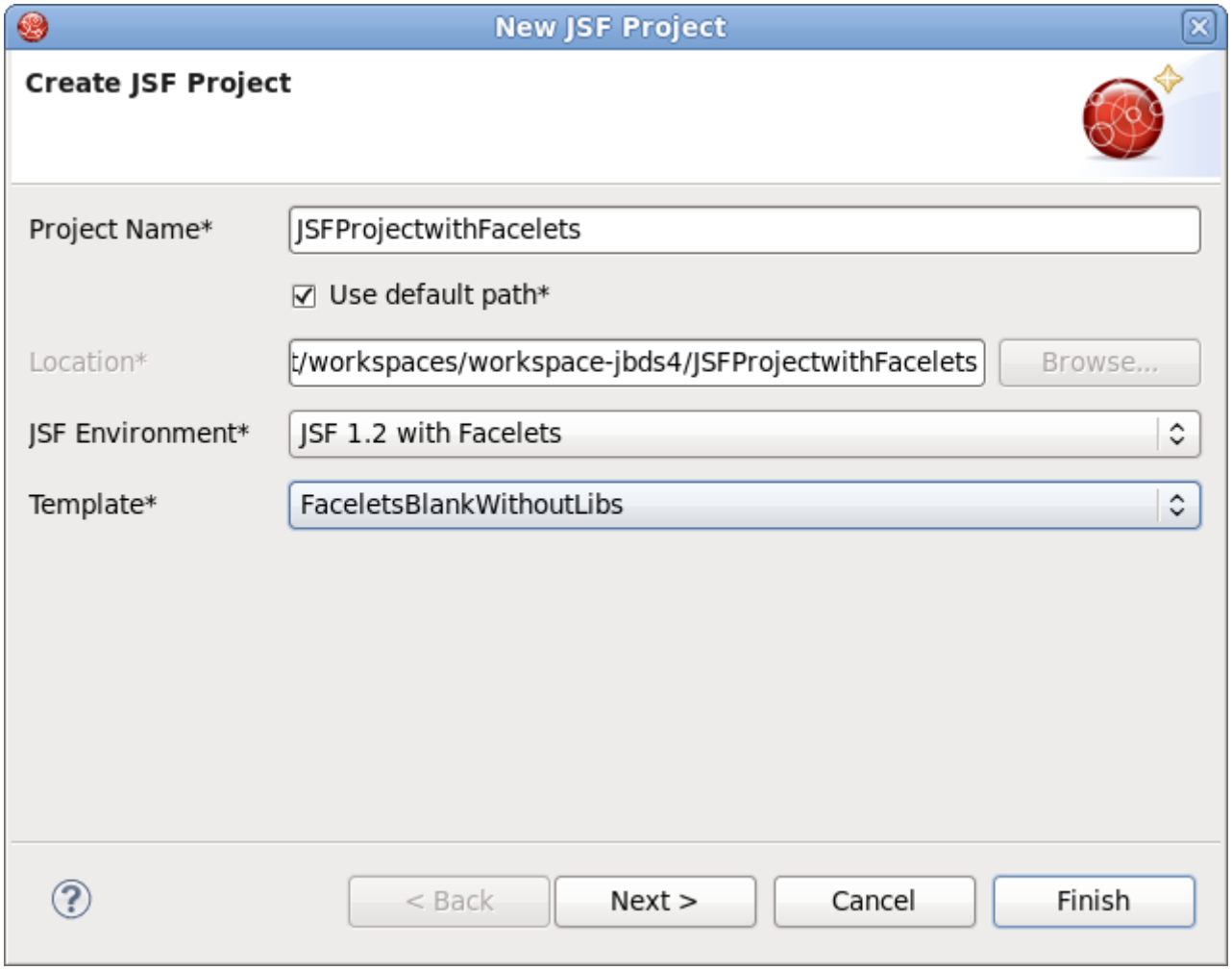

Figure 2.2. Choosing Facelets Environment

Once you have selected the environment, it is possible to specify one of three available templates:

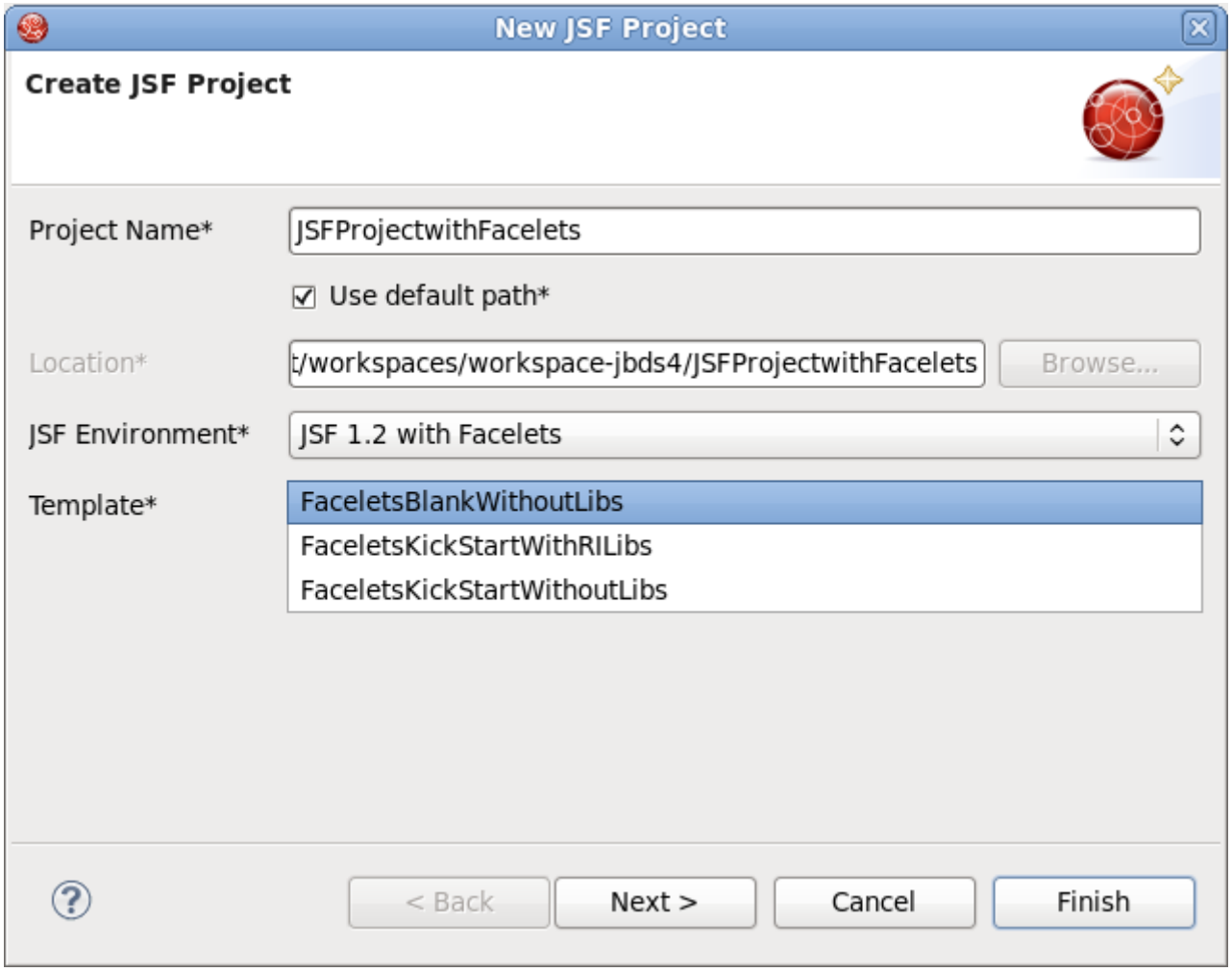

Figure 2.3. Choosing Facelets Template

The following table lists the templates that can be used with Facelets for any JSF project, and gives a detailed description for each one.

Table 2.1. Facelets Templates

Template	Description
FaceletsBlankWithoutLibs	Some servers already provide JSF libs and you risk library conflicts while deploying your project. To avoid such conflicts, use a template without libs if you have a server with its own JSF libraries.
FaceletsKickStartWithRILibs	A sample application with Facelets that is ready to run.
FaceletsKickStartWithoutLibs	A sample application without libraries.

2.1.2. Facelets components

The JBoss Tools Palette comes with the Facelets components ready to use. A useful tip appears when you hover the mouse cursor over the tag; this tip includes a detailed description of the tag component, the syntax and available attributes.

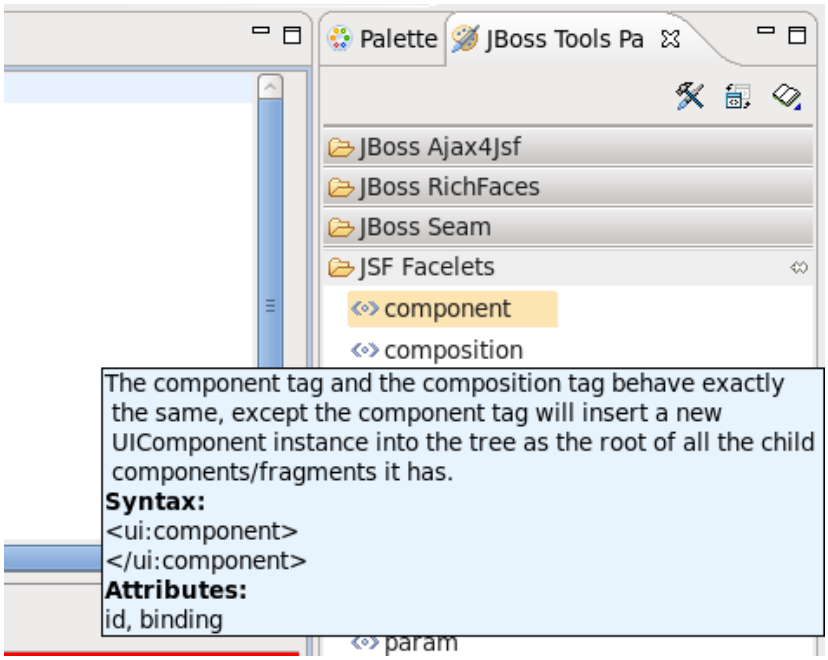

Figure 2.4. Facelets Components

2.1.3. Code assist for Facelets

JSF Tools provides Facelets code assistance, which can be accessed by pressing **Ctrl**+**Space**. It is available for Facelets tags while editing `.xhtml` files.

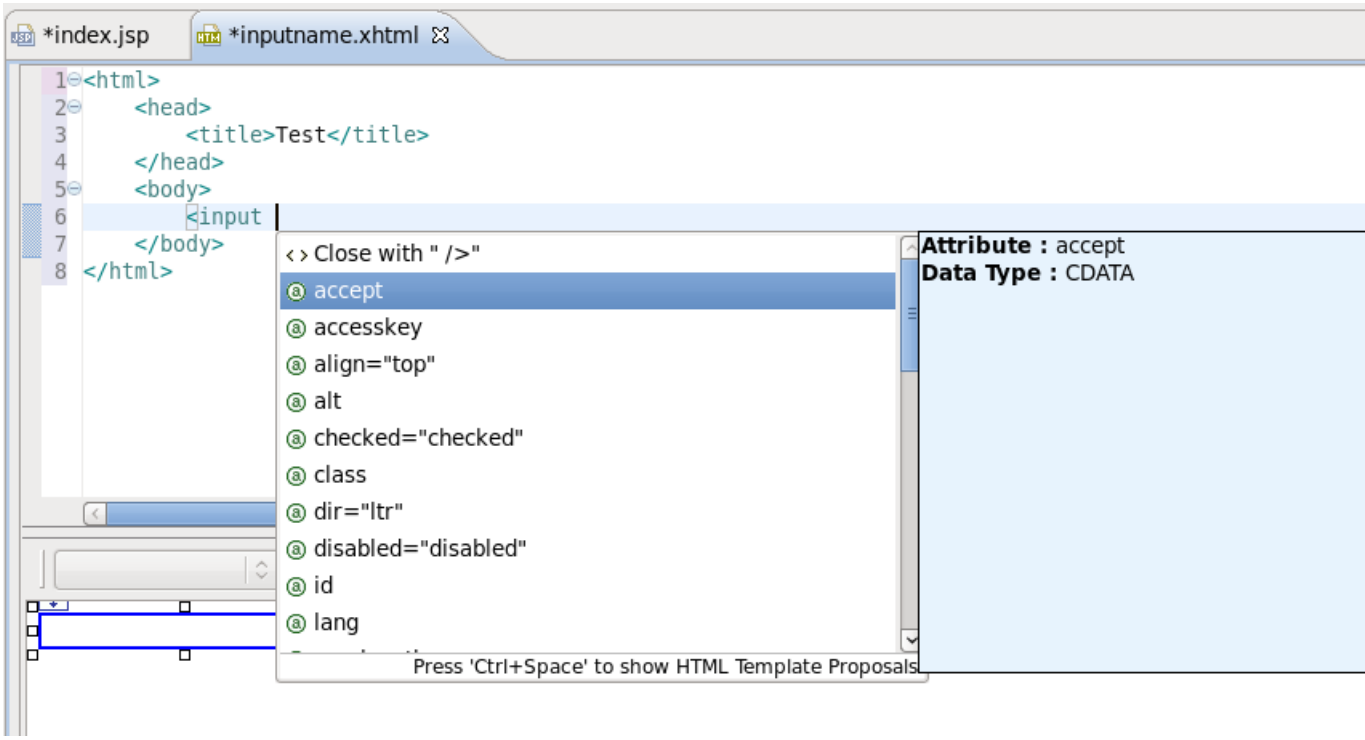

Figure 2.5. XHTML File Code Assist

Code assist is also available for `jsfc` attributes in any HTML tag.

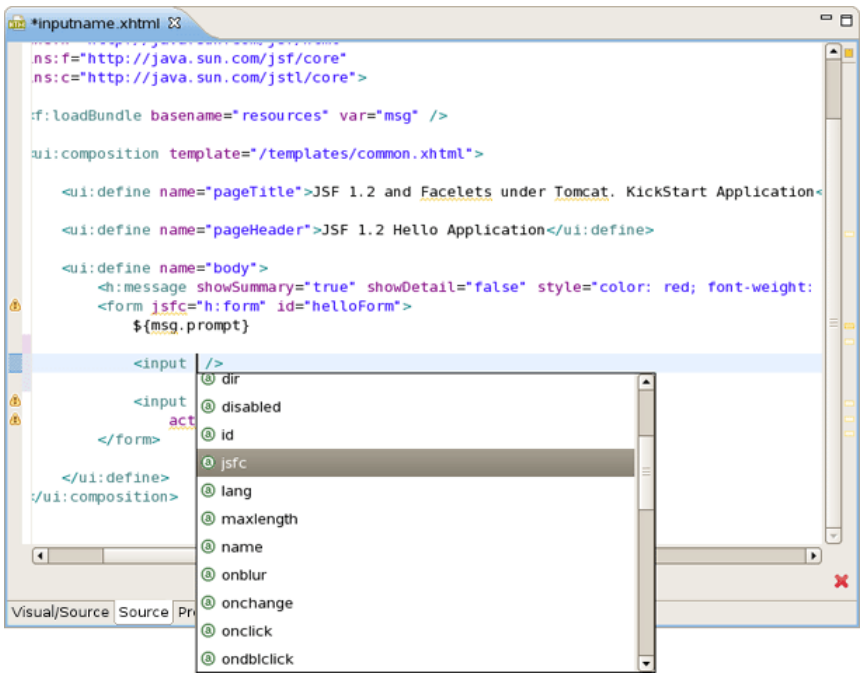

Figure 2.6. Code Assist for JSFC Attribute

After selecting an `jsfc` attribute, the code assist feature will list the JSF components available on a page.

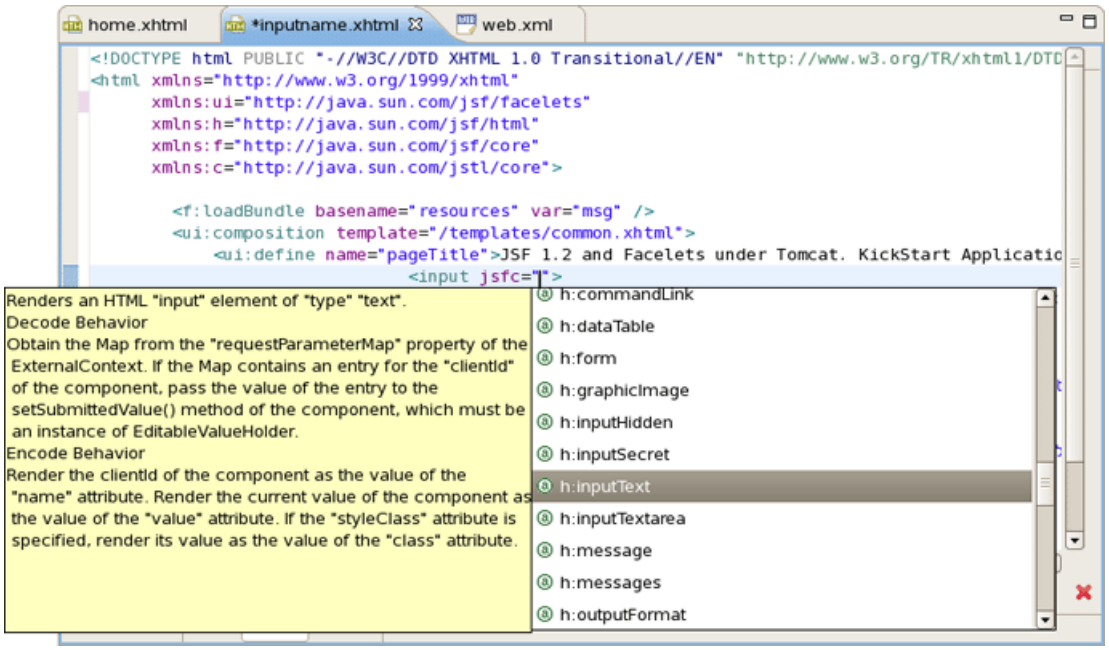

Figure 2.7. Code Assist for JSF Components

When a component is selected you will see all available attributes for it.

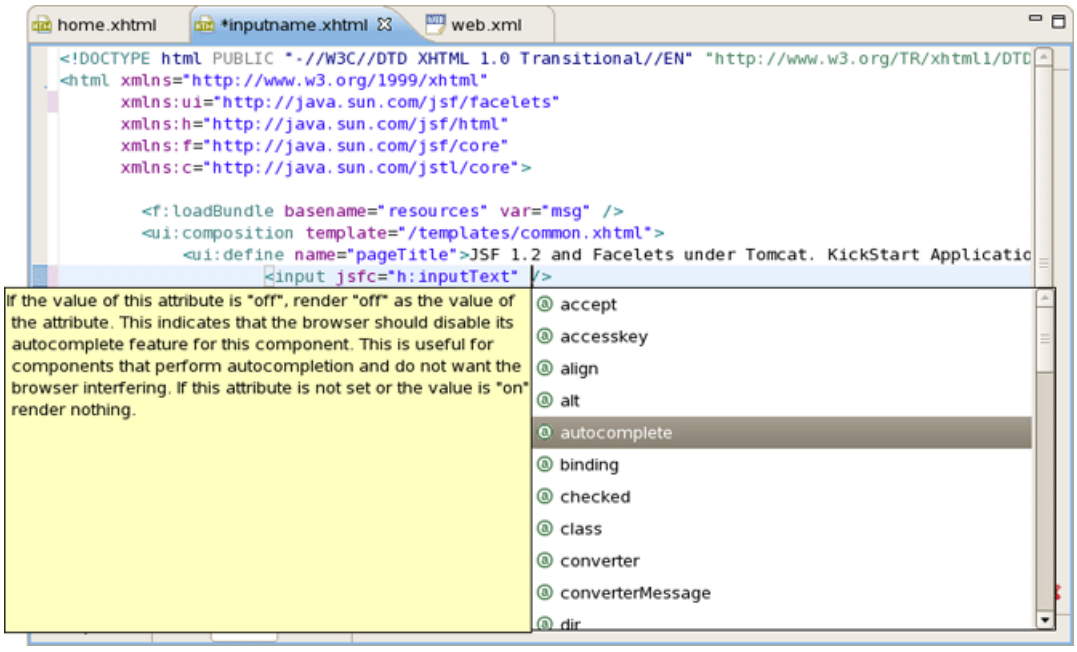

Figure 2.8. Available Attributes for the Component

2.1.4. Open On feature

Finally, JSF Tools supports Eclipse's OpenOn™ feature while editing Facelets files. Using this feature, you can easily navigate between the Facelets templates and other parts of your projects.

By holding down the **Ctrl** key while hovering the mouse cursor over a reference to a template, the reference becomes a hyperlink to navigate to that template.

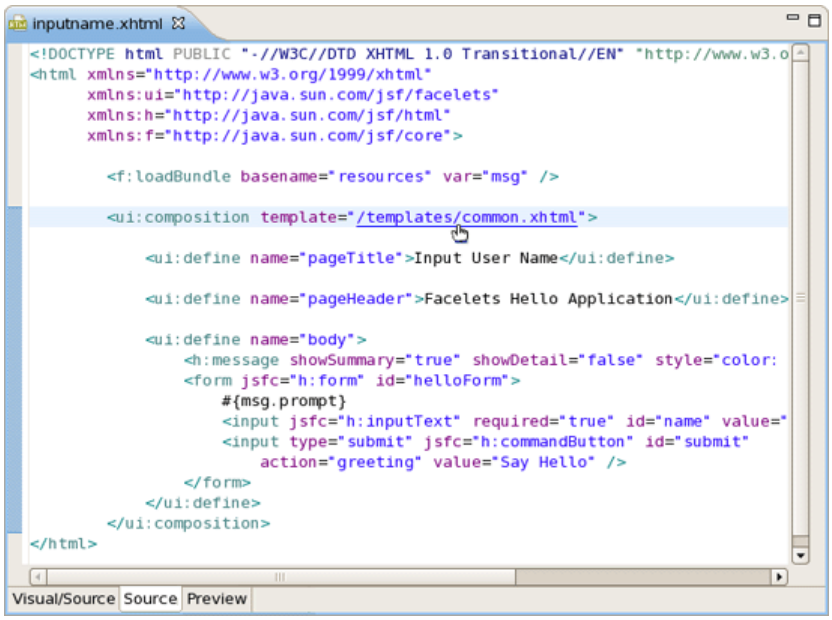

Figure 2.9. Template Hyperlink

Projects

To take an advantage of JSF you will need to perform one of the next steps:

- Create new JSF projects
- Import (open) existing JSF projects
- Add JSF capability to any existing Eclipse project
- Import and add JSF capability to any existing project created outside Eclipse.

This section will go into more detail for each step.

3.1. Creating a New JSF Project

It is easy to create a new project that contains all the JSF libraries, tag libraries and JSF configuration file with the aid of a special wizard. To get it, select **File** → **New** → **Other** → **JBoos Tools Web** → **JSF** → **JSF Project** and click the **Next** button.

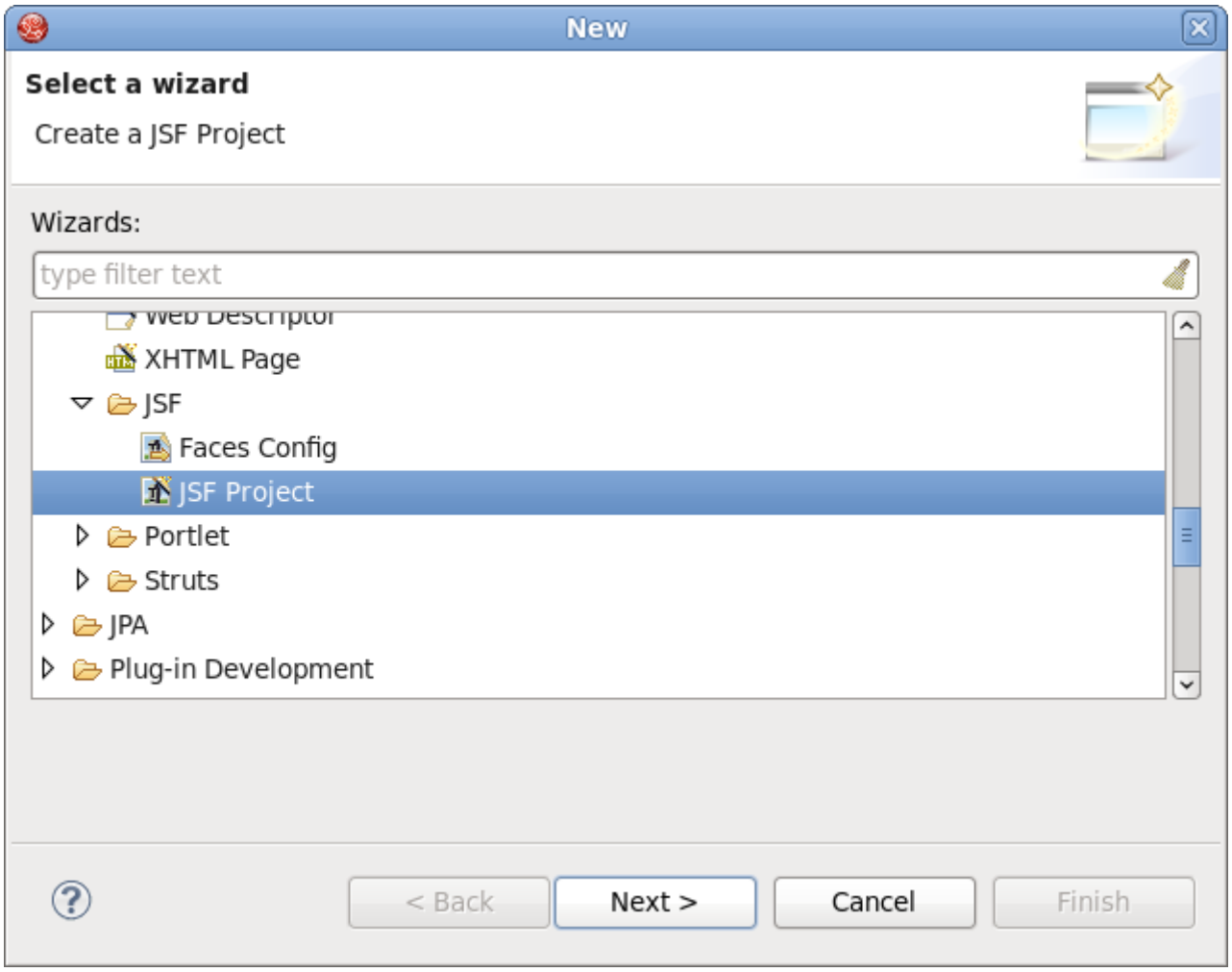

Figure 3.1. Choosing a JSF Project

On the next page you will be prompted to enter the **Project Name** and select a location for the project (or just leave a default path).

The **JSF Version** option also allows you to specify the JSF implementation to use.

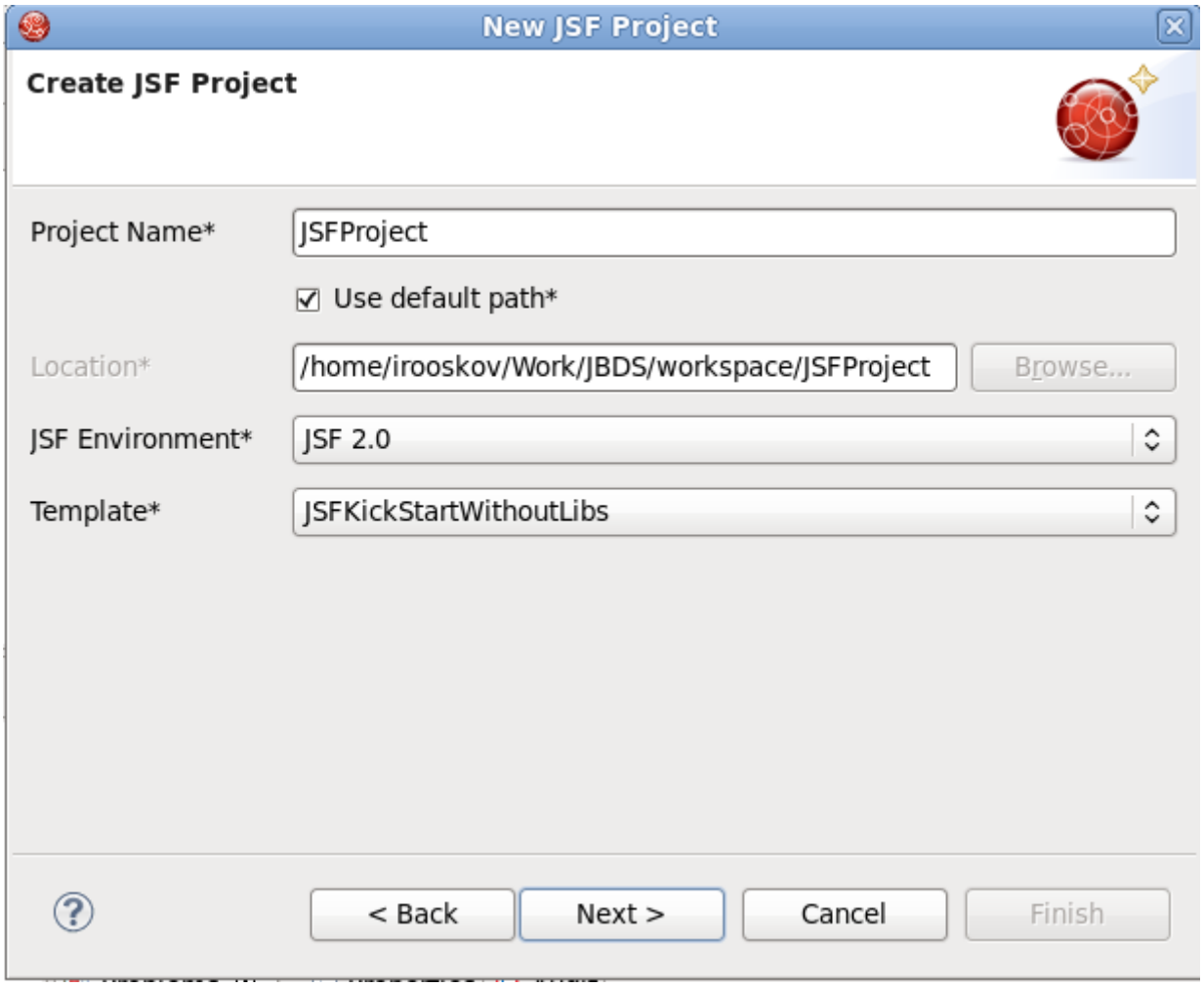

Figure 3.2. Creating a New JSF Project

There are a number of predefined project templates that are both flexible and easily customizable. You can pick a different template on which the projects Importing Existing should be based on. Almost all templates come in two variations: with and without JSF libraries.

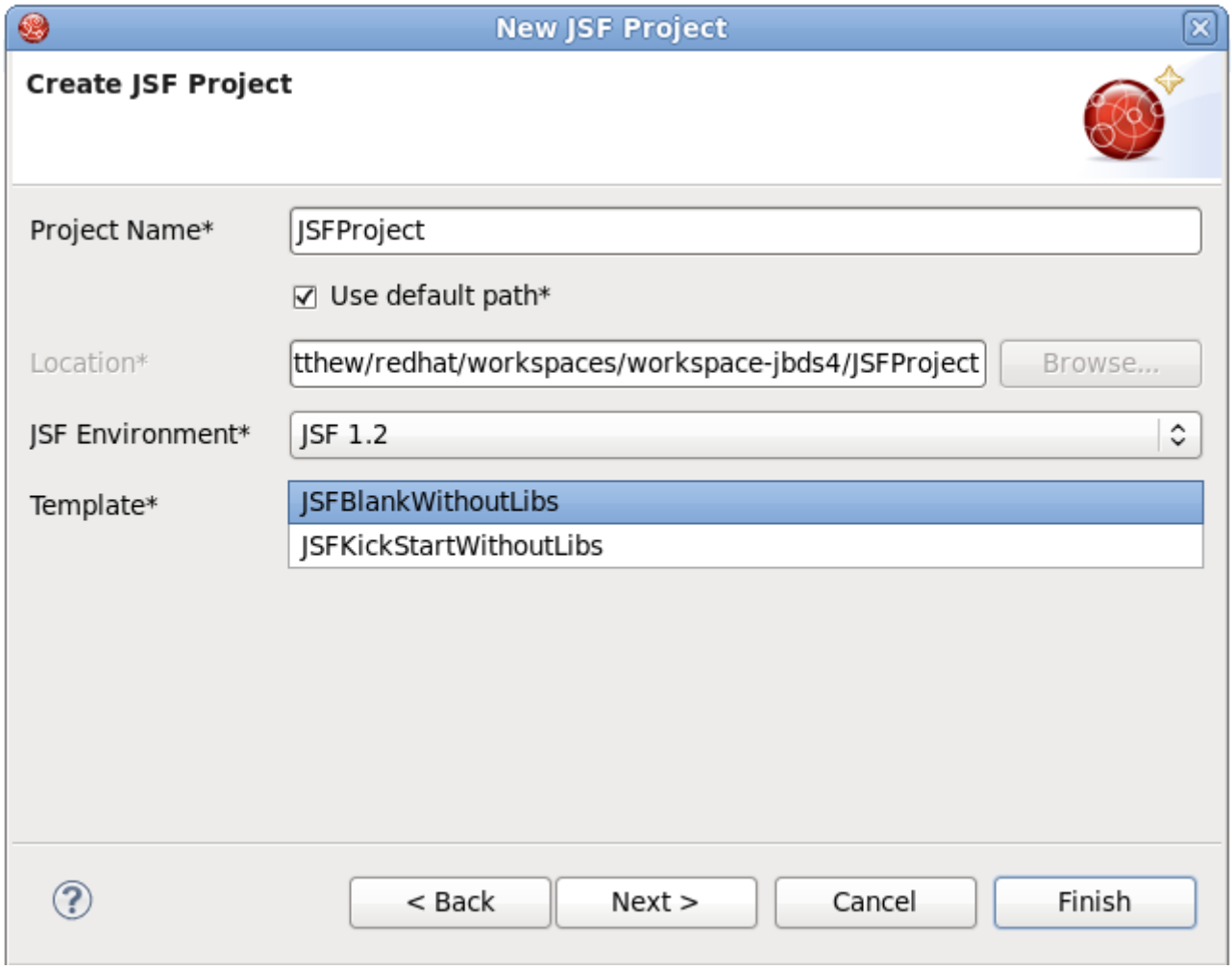

Figure 3.3. Choosing JSF Templates

The table below provides description for each possible JSF template.

Table 3.1. JSF Project Templates

Template	Description
JSFBlankWithoutLibs	This template will create a standard Web project structure with all the JSF capabilities. Use a template without libs to avoid library conflicts when your server already has JSF libraries installed.
JSFKickStartWithoutLibs	This template will create a standard Web project structure, and also include a sample application that is ready to run. Use a template without libs to avoid library conflicts when your server already has JSF libraries installed.

On the next page you need to select which **Servlet version** to use, and specify whether or not to register this application with JBoss AS (or other server) in order to run and test your application.

The **Context Path** option defines the name under which the application will be deployed.

The **Runtime** value tells Eclipse where to find the Web libraries necessary to build (compile) the project. It is not possible to finish the project creation without selecting a Runtime. If you do not have any values, click the **New...** button to add new Runtime.

The **Target Server** option allows you specifying whether or not to deploy the application. The Target Server corresponds to the Runtime value selected above. If you do not want to deploy the application, uncheck this option.

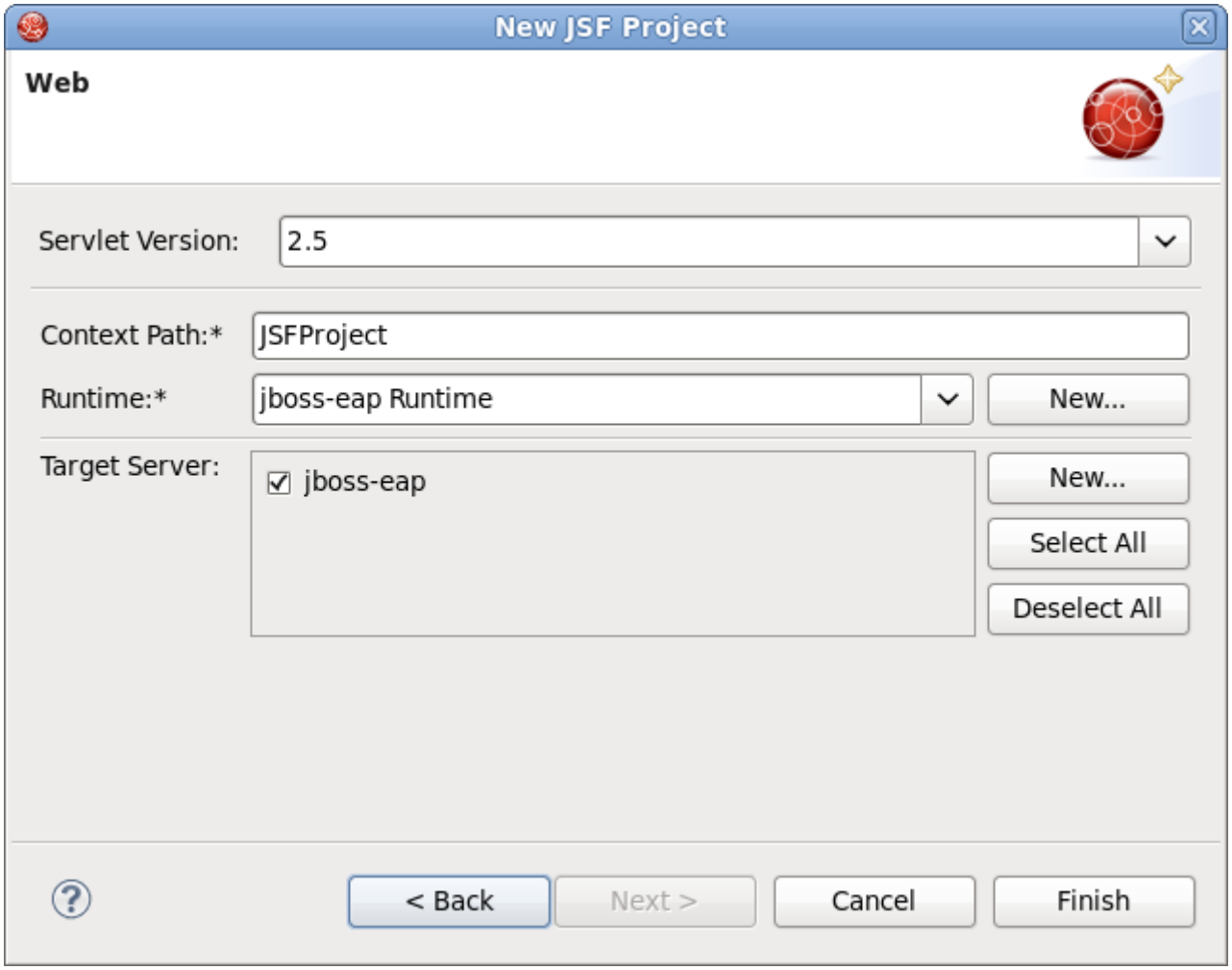

Figure 3.4. Registering the Project on Server

When you are all done, you should see that the project has appeared in the Package Explorer view:

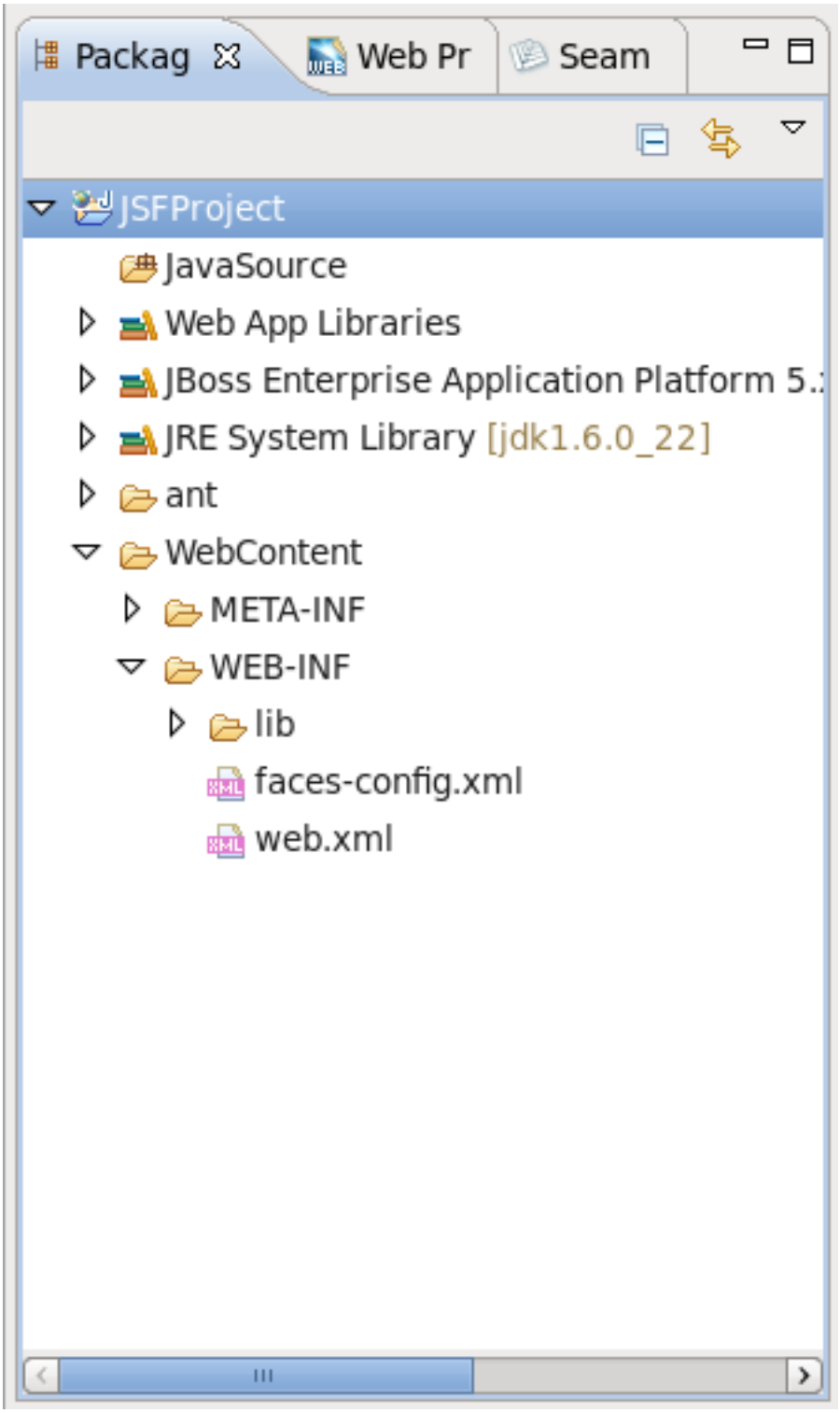

Figure 3.5. A New Project in the Package Explorer

At this point you can open the `faces-config.xml` file and start working on your application. There are a lot of features available when developing JSF applications. These features will be discussed in more detail later in this document.

3.2. Importing Existing JSF Projects with Any Structure

For detailed information on migration of JSF projects into a workspace see the Migration Guide.

3.3. Adding JSF Capability to Any Existing Eclipse Project

It is also possible to add JSF™ capabilities (JSF libraries, tag libraries) to any existing Eclipse project in your workspace. After that you will be able to make use of features such as the JSF configuration editor, JBoss Tools JSP editor and any others.

Right click the project and select **JBoss Tools** → **Add JSF Capabilities** from the context menu. This will start the process of adding the necessary libraries and files to convert the project into a Web JSF project.

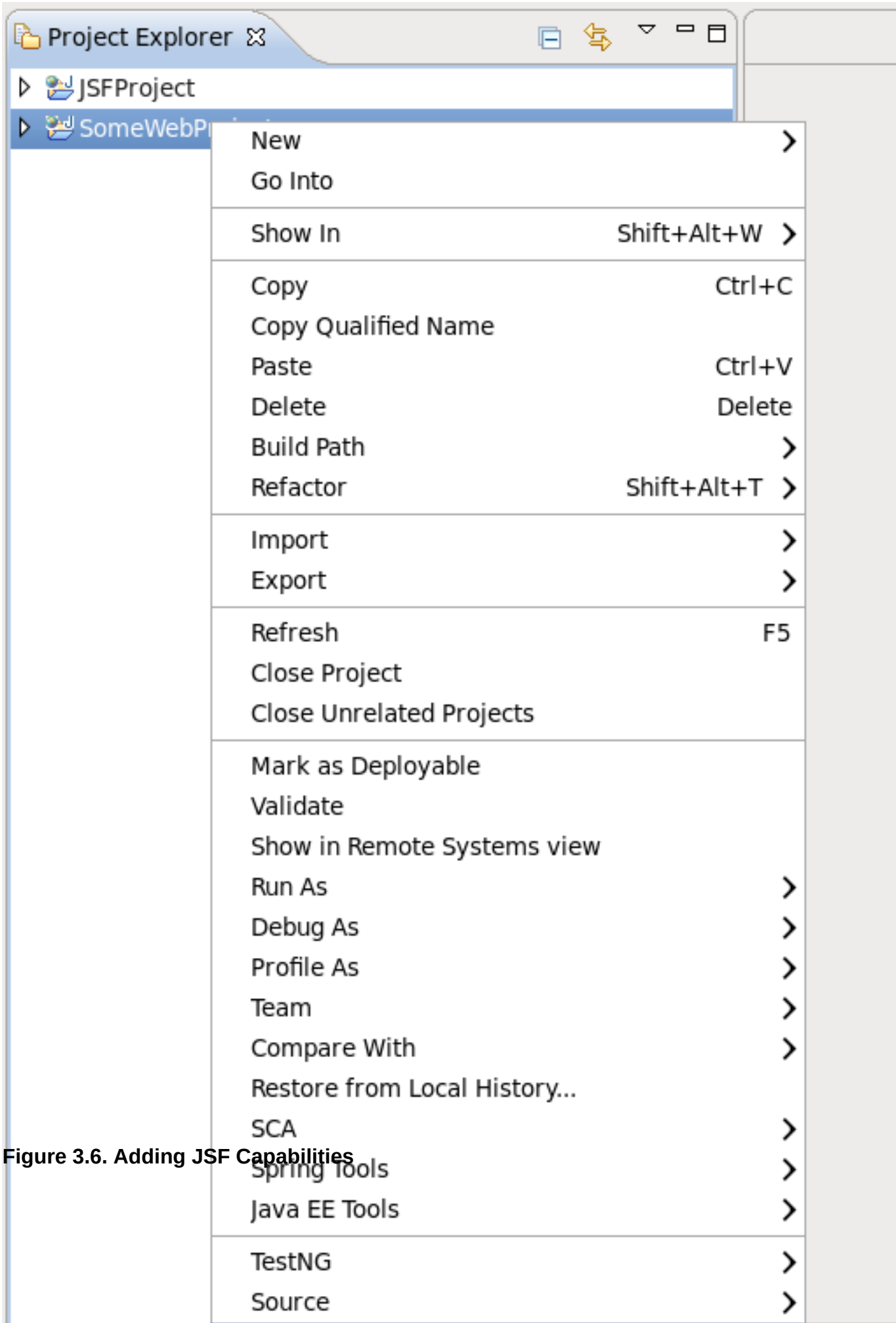

Figure 3.6. Adding JSF Capabilities

The wizard will first ask you to define location of the `web.xml` file and the project name.

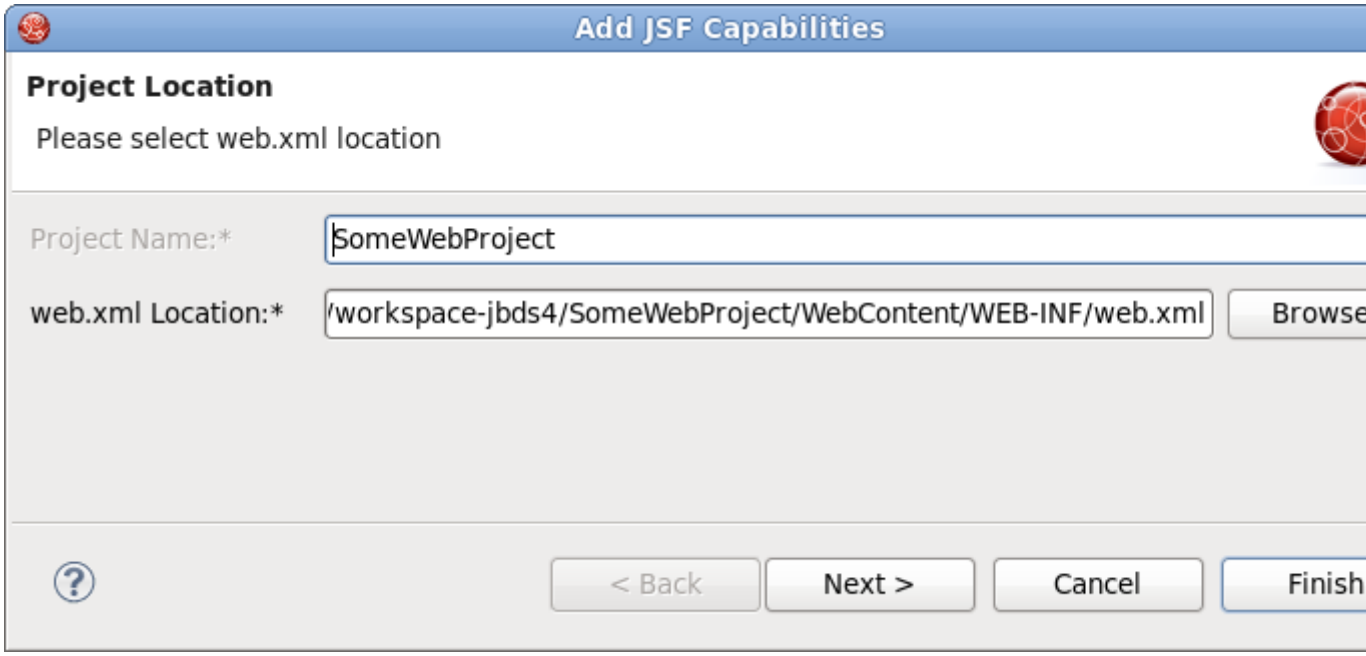

Figure 3.7. Project Location

On the last page you can specify the different folders for your project as well as register this application with a servlet container.

Make sure to select the **Add Libraries** option to add all required JSF related libraries into your project.

The **Context Path** option defines the name under which the application will be deployed.

The **Runtime** value tells Eclipse where to find Web libraries necessary to build (compile) the project. It is not possible to finish project import without selecting the Runtime. If you don not have any values, click the **New...** button to add a new Runtime.

The **Target Server** option allows you to specify whether or not to deploy the application. The Target Server corresponds to the Runtime value selected above. If you do not want to deploy the application, uncheck this value.

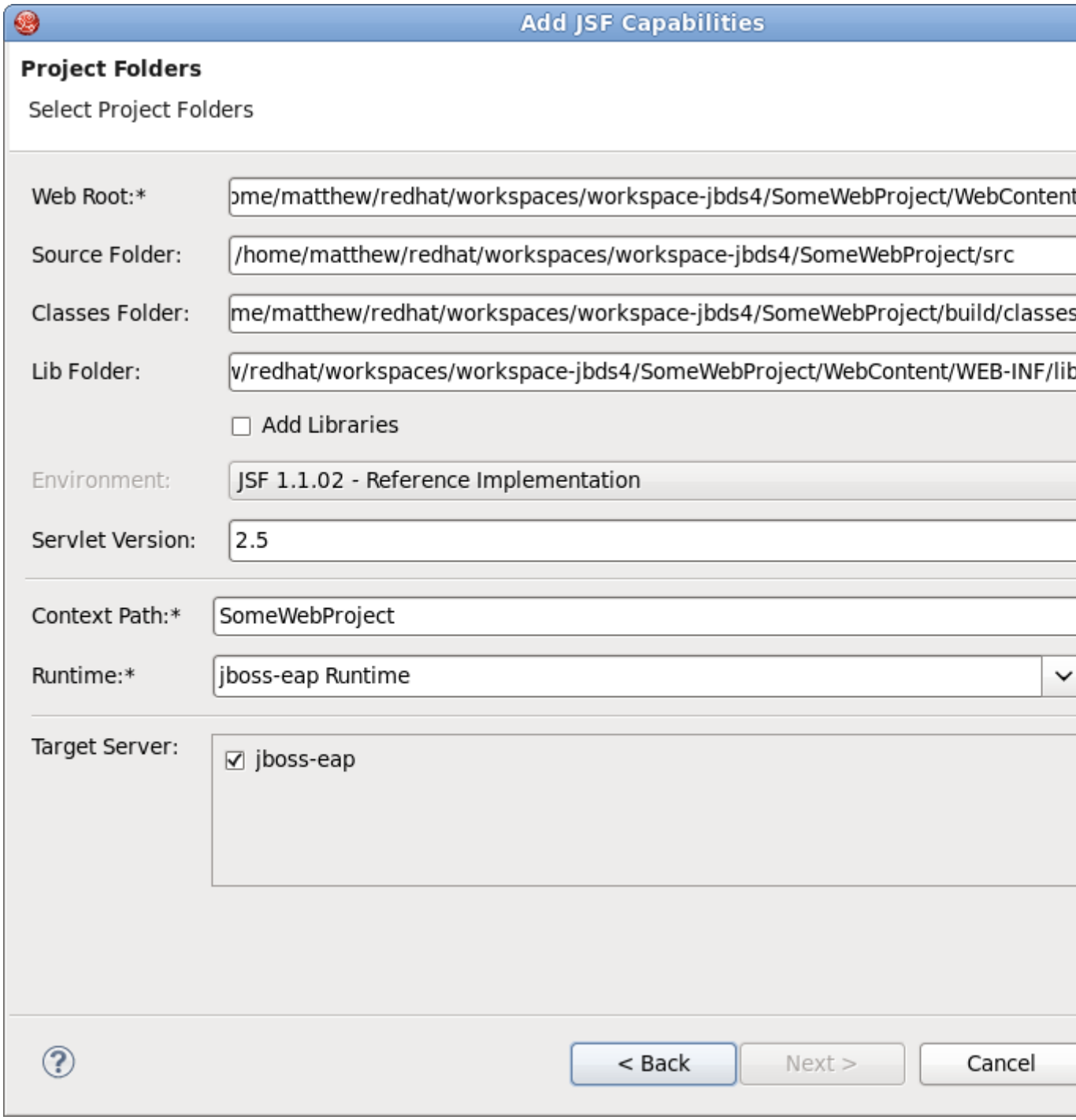

Figure 3.8. Project Folders

Once your project is imported you can see that JSF related libraries `jsf-api.jar` and `jsf-impl.jar` have been added to your project.

Note:

Some application servers provide their own JSF implementation libraries. To avoid conflicts you should not add JSF libraries while adding JSF capabilities.

You are now ready to work with JSF by creating a new JSF configuration file:

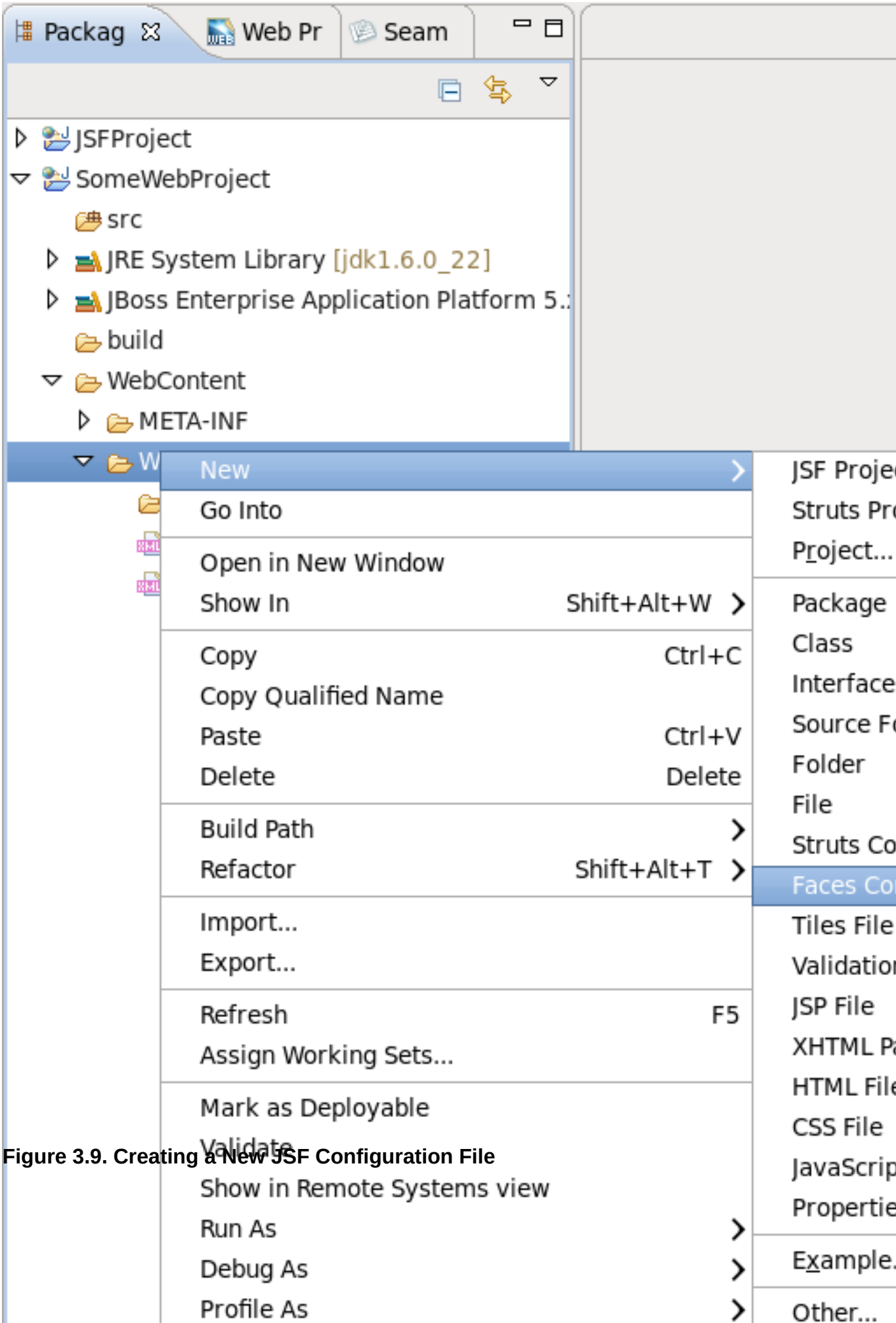

Figure 3.9. Creating a New JSF Configuration File

Once the file has been created, it should be opened in a special editor (see *Chapter 5, JSF Configuration File Editor*).

3.4. Adding Your Own Project Templates

A template is a set of files that is provided as a basis when creating a new project. Project templates provide content and structure for a project.

JSF Tools provides powerful template capabilities which allow you to create new templates and import existing Struts and JSF projects. This templating facility has a variety of aspects to consider. Let's start with the most straightforward case and consider the process of creating a template from your existing JSF project.

Let's say you have a project that you want to use as the basis for a new template. The following steps will show you how to achieve this:

- In the Web Projects view, right-click the project and select **JBoss Tools JS** → **Save As Template**

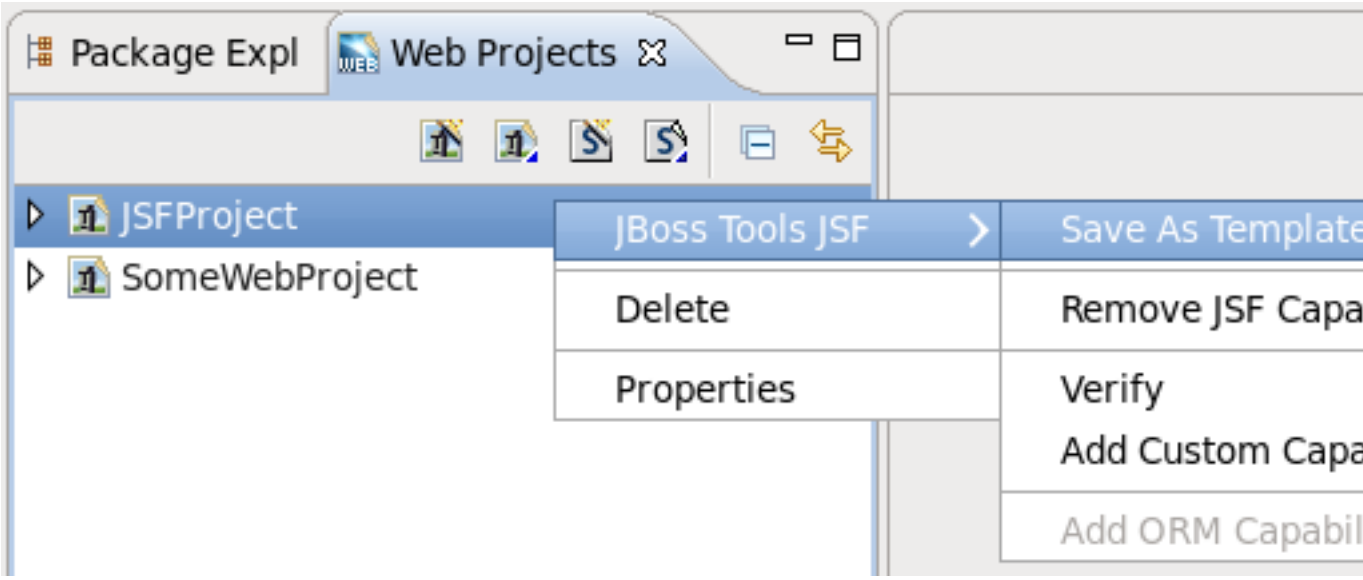

Figure 3.10. Saving Your Project as Template

- In the first dialog box, you can specify a name for the template (it will default to the project name) and confirm what run-time implementation of the project technology will be used.

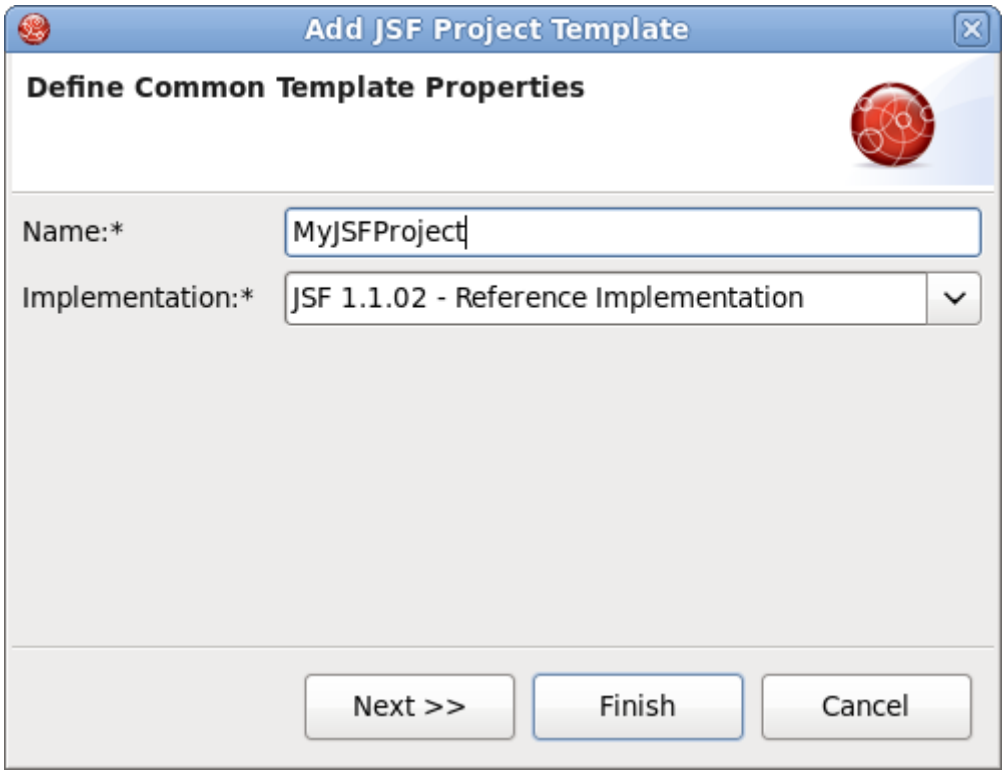

Figure 3.11. Define Template Properties

- When you click the **Next** button a dialog box will be presented with your project structure displayed, along with a number of check boxes. Here you can select only those parts and files in your project directory that should be part of the template.

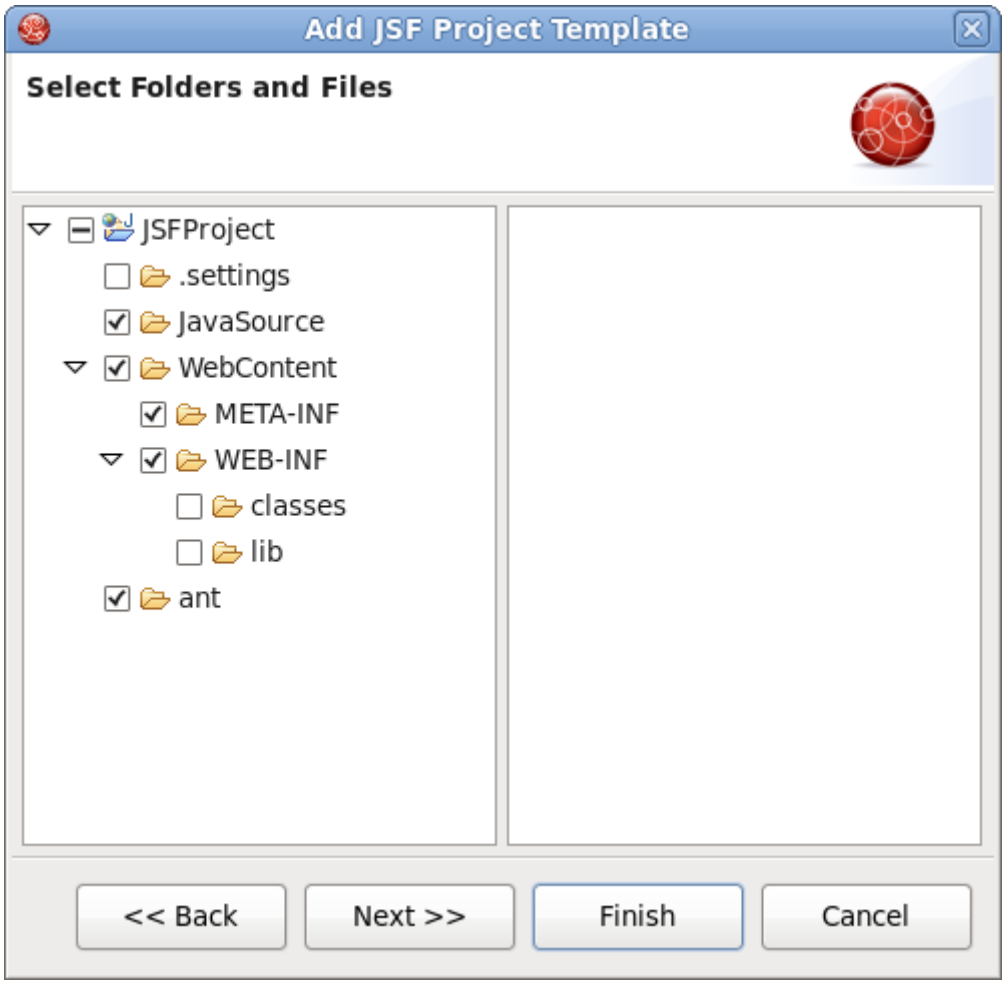

Figure 3.12. Define Template Properties

- At this point, unless you want to designate some extra files as having Velocity template coding inside them, you should click the **Finish** button.

That's it. This template can be used with any new or imported project that uses the same run-time implementation as the project you turned into a template.

At this point you have a fully configured project. Now you can add some additional logic to it starting with the JSF configuration file.

3.5. Relevant Resources Links

You can find a more in-depth explanation on how to work with the special wizards, editors and views that can be used while developing JSF applications in our Visual Web Tools Guide.

Web.xml Editor

The `web.xml` file inside the `WEB-INF` folder is a deployment descriptor file for a Web Application. It describes the servlets and other components and deployment properties that make up your application.

JBoss Tools add the `web.xml` file to created JSF project automatically and provides a special editor for its editing. See the Visual Web Tools guide for more information on the `web.xml` editor.

JSF Configuration File Editor

First, we should mention that JSF configuration file (`faces-config.xml`) is intended for registering JSF application resources such as Converters, Validators, Managed Beans and page-to-page navigation rules.

Now, let's look at how you can easily configure this file by means of a special graphical editor for the JSF configuration file. The editor has three main views:

- Diagram
- Tree
- Source

They can be selected via the tabs at the bottom of the editor.

5.1. Diagram view

Here, we will show you how to work with JSF configuration file through the Diagram view of the editor.

As you can see on the figure below, the Diagram view displays the navigation rules container in the `faces-config.xml` file:

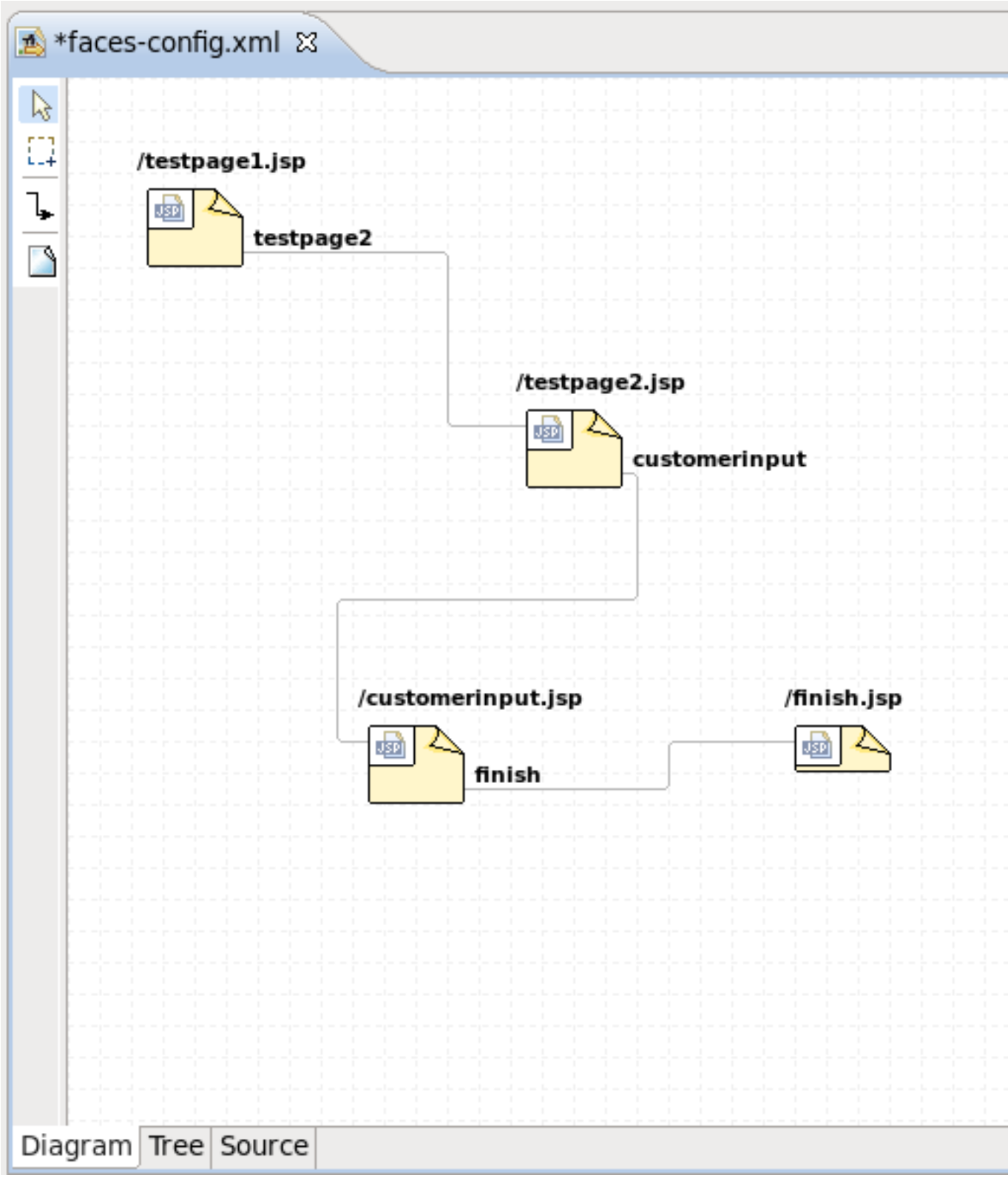

Figure 5.1. Diagram View

If you have a large diagram, make use of the Outline view. Within it you can switch to a **Diagram Navigator** mode by selecting the middle icon at the top of the view window. This allows you to

easily move around the diagram. Just move the blue area in any direction, and the diagram on the left will also move:

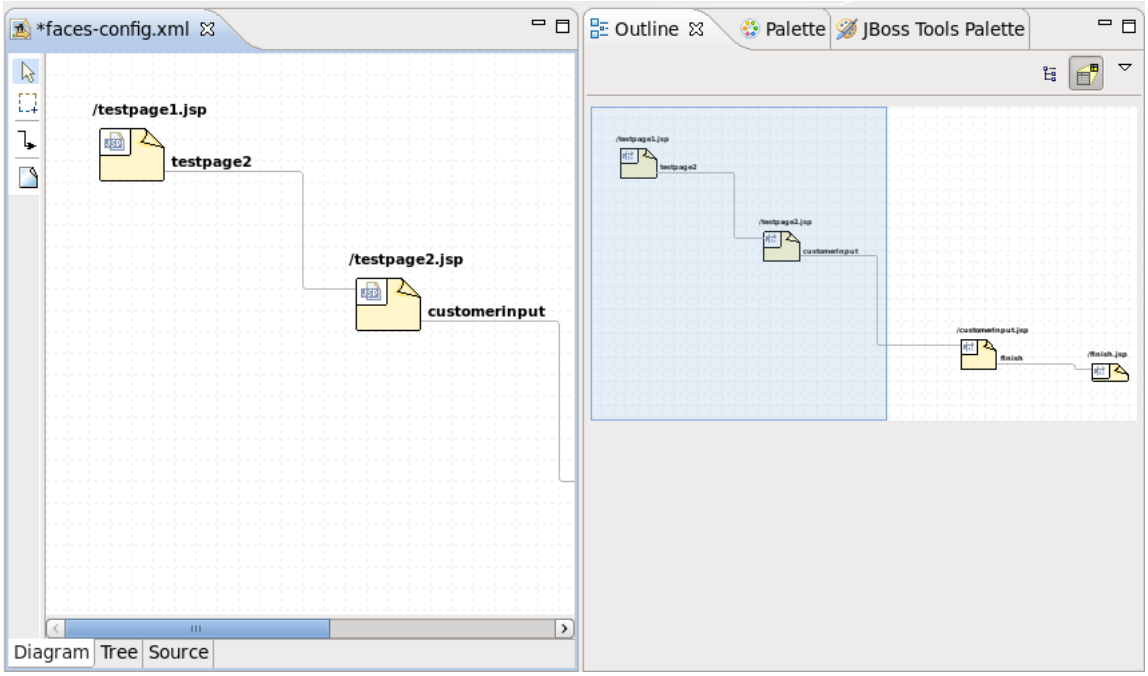

Figure 5.2. Outline View for Diagram

To create a new page here, you should click the page icon (View Template) on the toolbar from the left and then click anywhere on the diagram. A New Page Wizard will appear.

To create a transition for connecting pages:

- Select the transition icon from the toolbar (New Connection).
- Click the source page.
- Click the target page.

A transition will appear between the two pages:

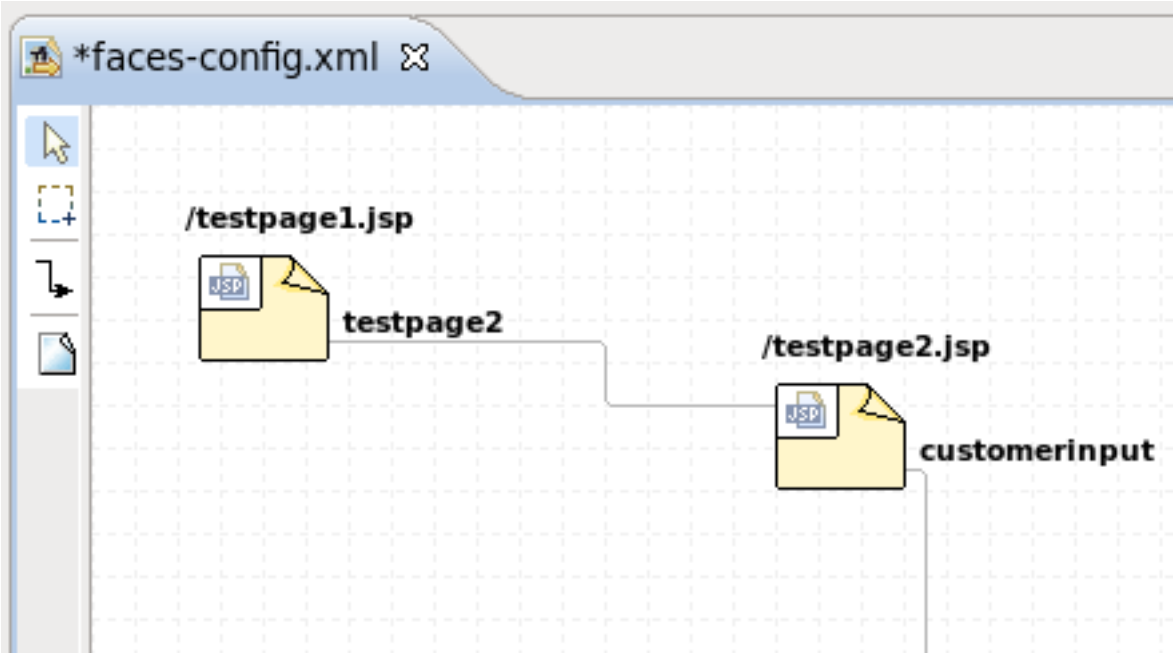

Figure 5.3. Transition between JSP Pages

It is also possible to create a new page with context menu by right-clicking anywhere on the diagram and selecting the **New View...** option.

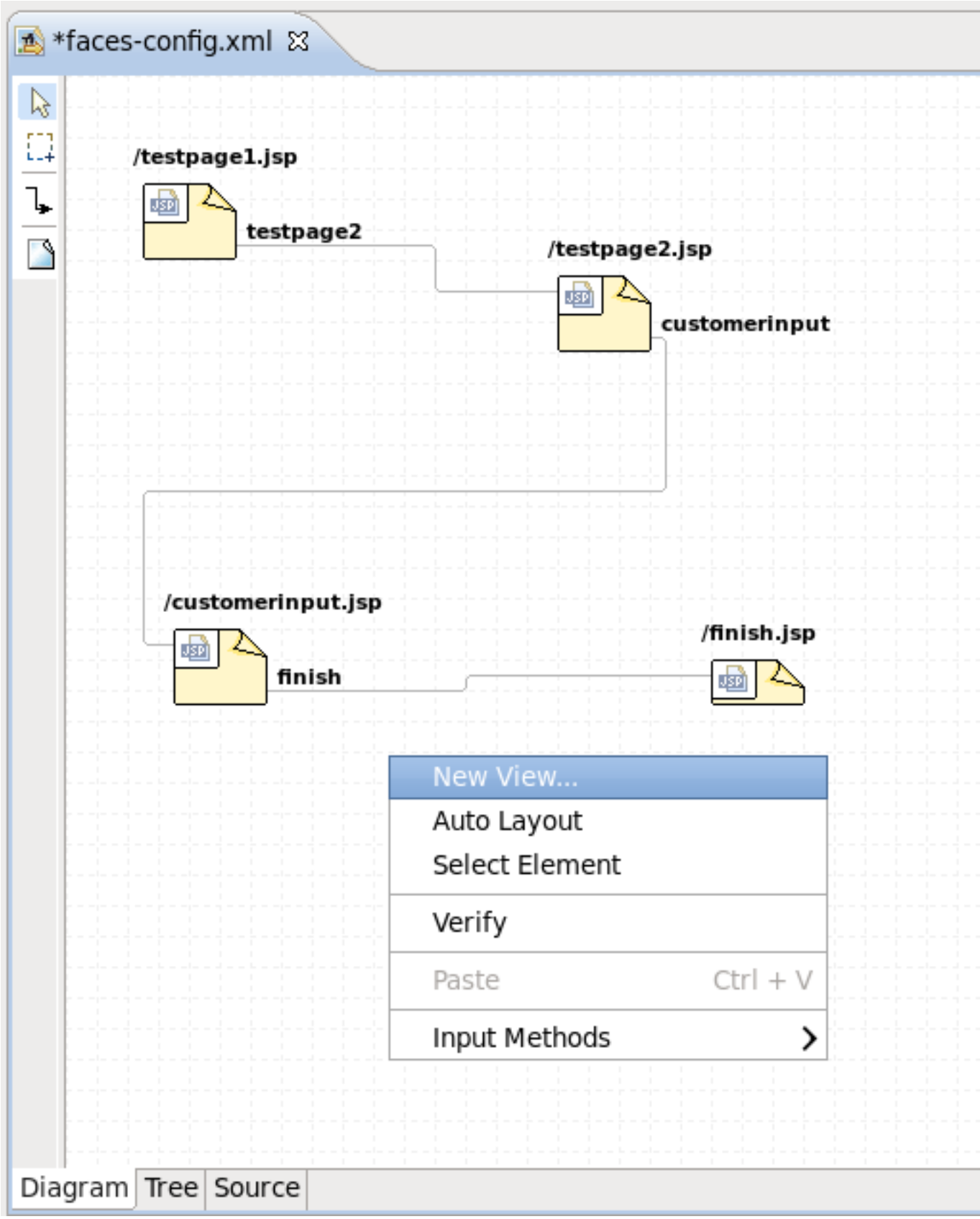

Figure 5.4. Creating a New View

To edit an existing transition, first select the transition line. Then, place the mouse cursor over the last black dot (on the target page). The mouse cursor will change to a big +. At this point, drag the line to a new target page:

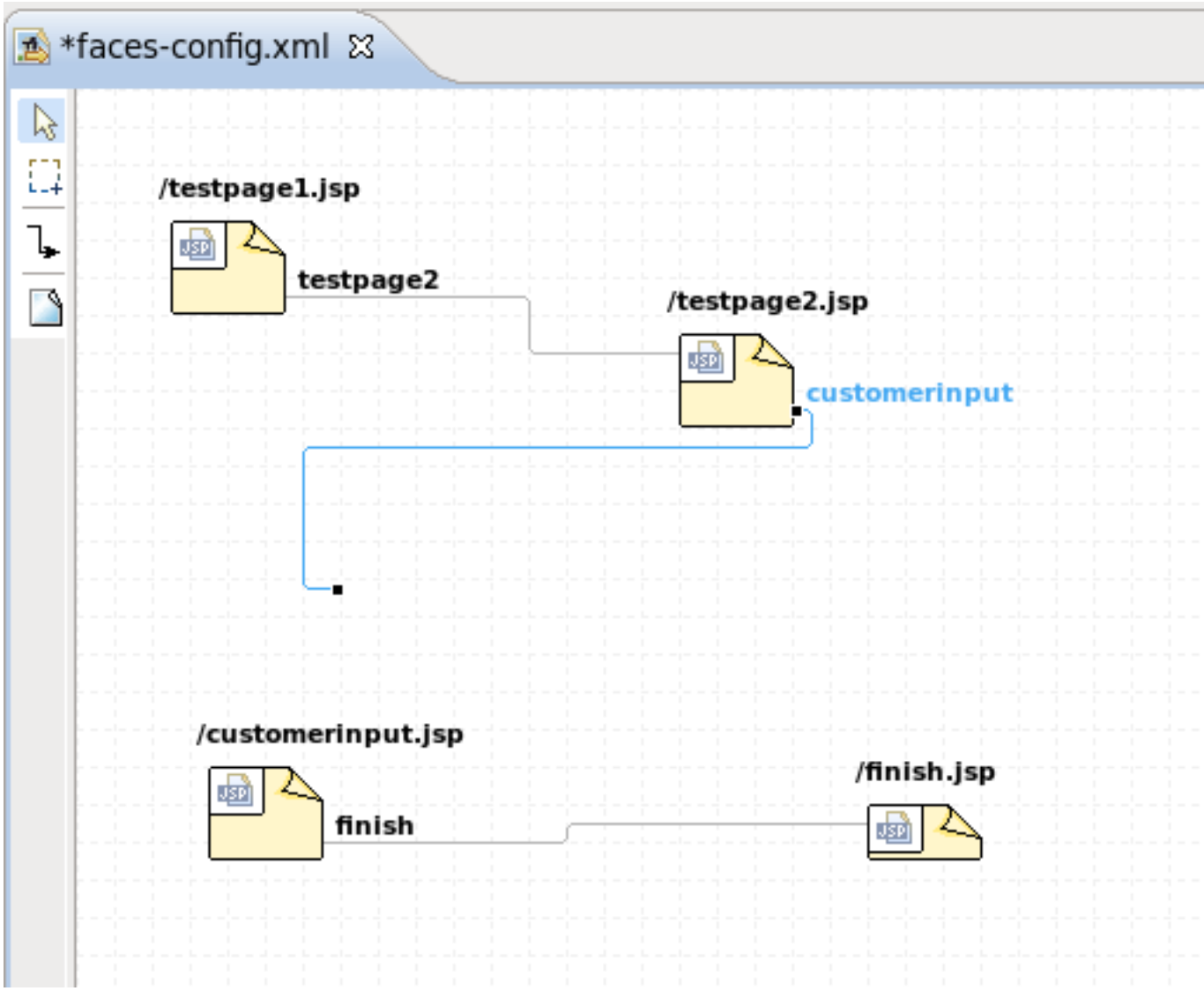

Figure 5.5. Editing Transition between Views

5.2. Tree View

You can find it more convenient to edit your JSF Configuration file in the Tree view of the **VPE**.

The view displays all JSF application artifacts referenced in the configuration file in a tree format. By selecting any node on the left, you can view and edit its properties which will appear in the right-hand area. Let's look at the structure of this tree more closely.

- Under the **Application** node you can adjust JSF application specific settings such as internationalization, extensions, adding property and variable resolvers, etc.

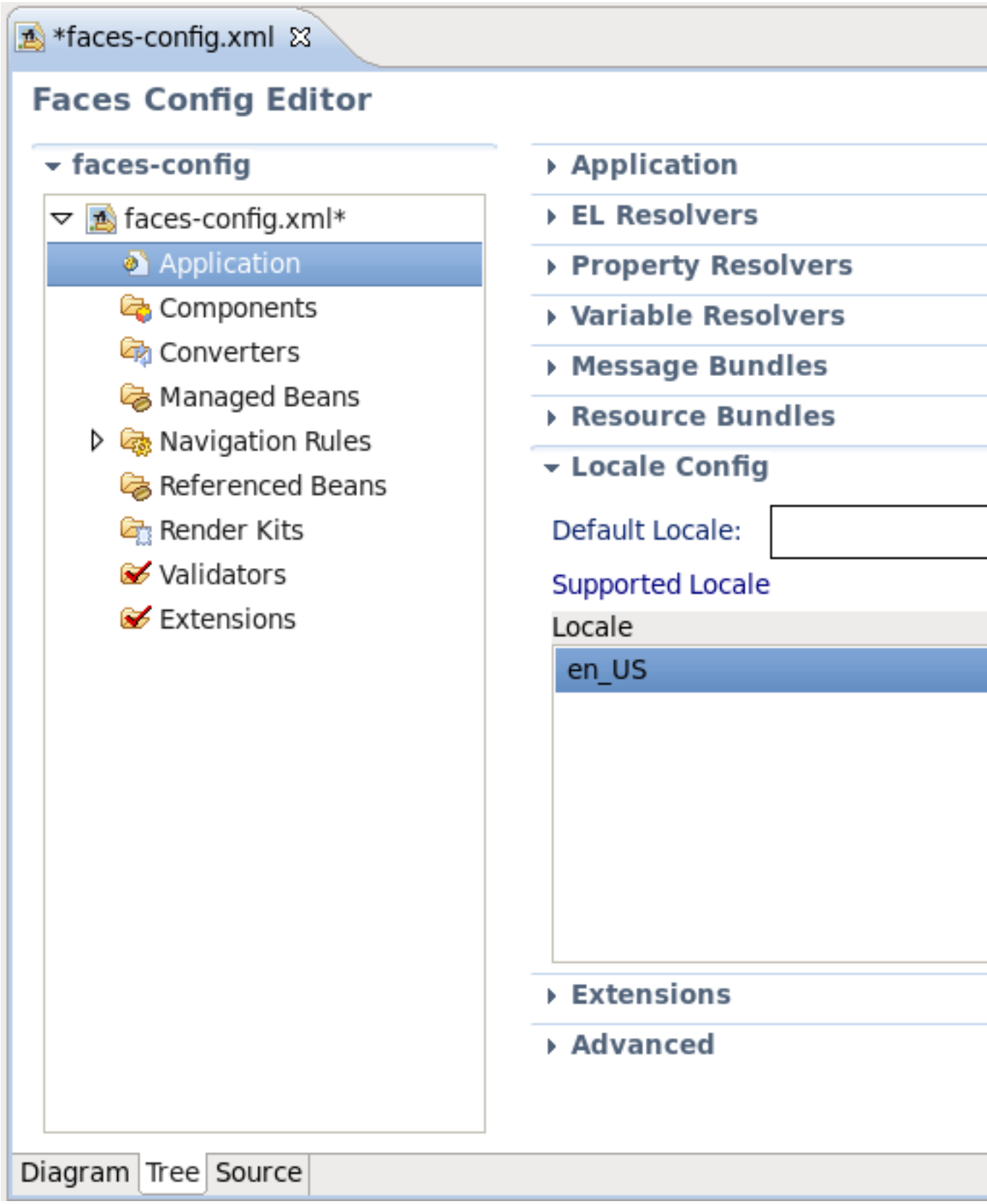

Figure 5.6. JSF Application Specific Settings

- The **Components** node is for registering custom JSF components. Right-click and select **New** → **Component** or just click the **Add** button in the right-hand area to add a new component to the JSF Configuration file.

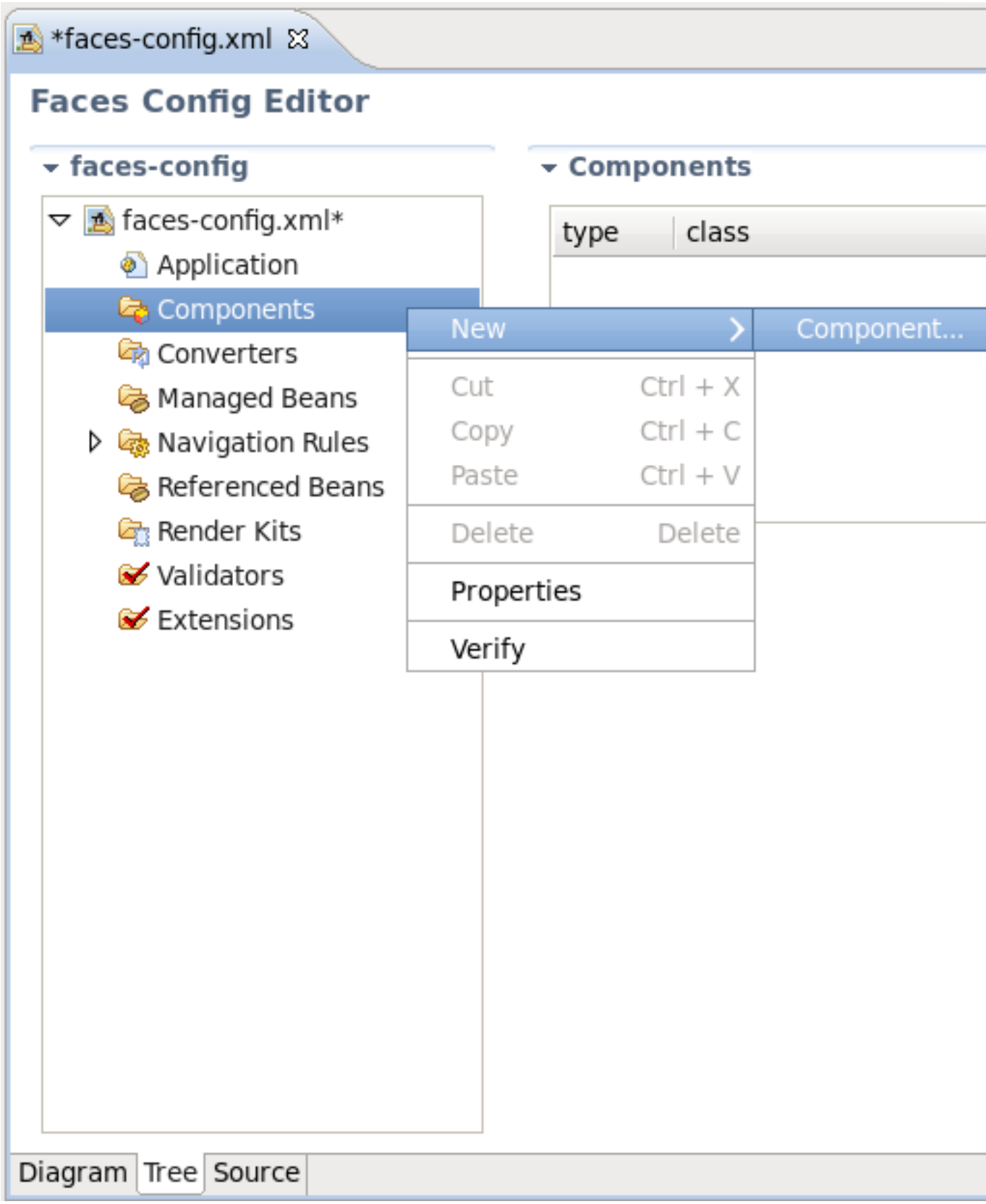

Figure 5.7. Registering a New JSF Component

In the **Add Component** wizard you should set a component type and point to a component class by using the **Browse** button or create a new class for this component by using the **Component-Class** link.

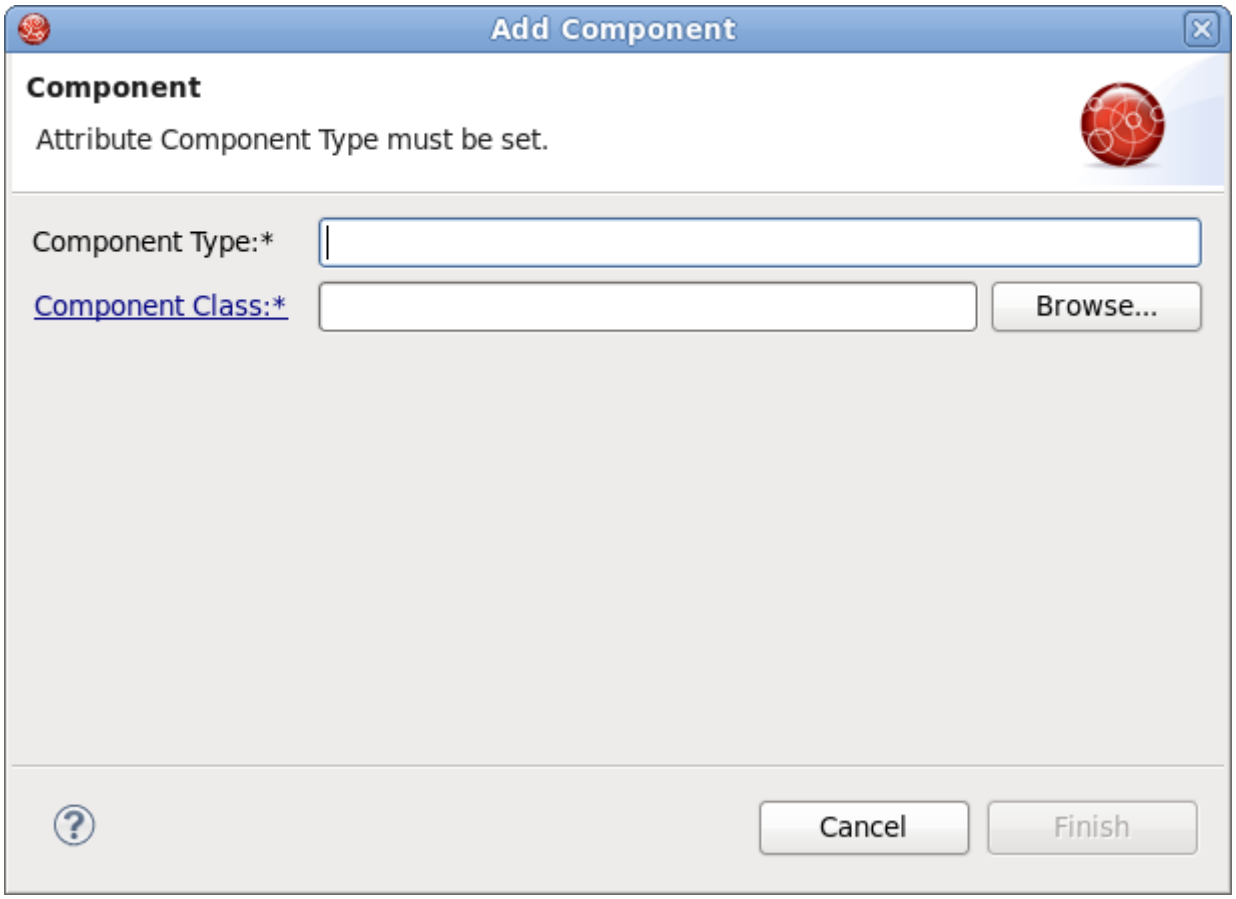

Figure 5.8. Adding a New JSF Component to the JSF Configuration File

- Use the **Render Kit** node to create and register a set of related renderers for custom JSF components.

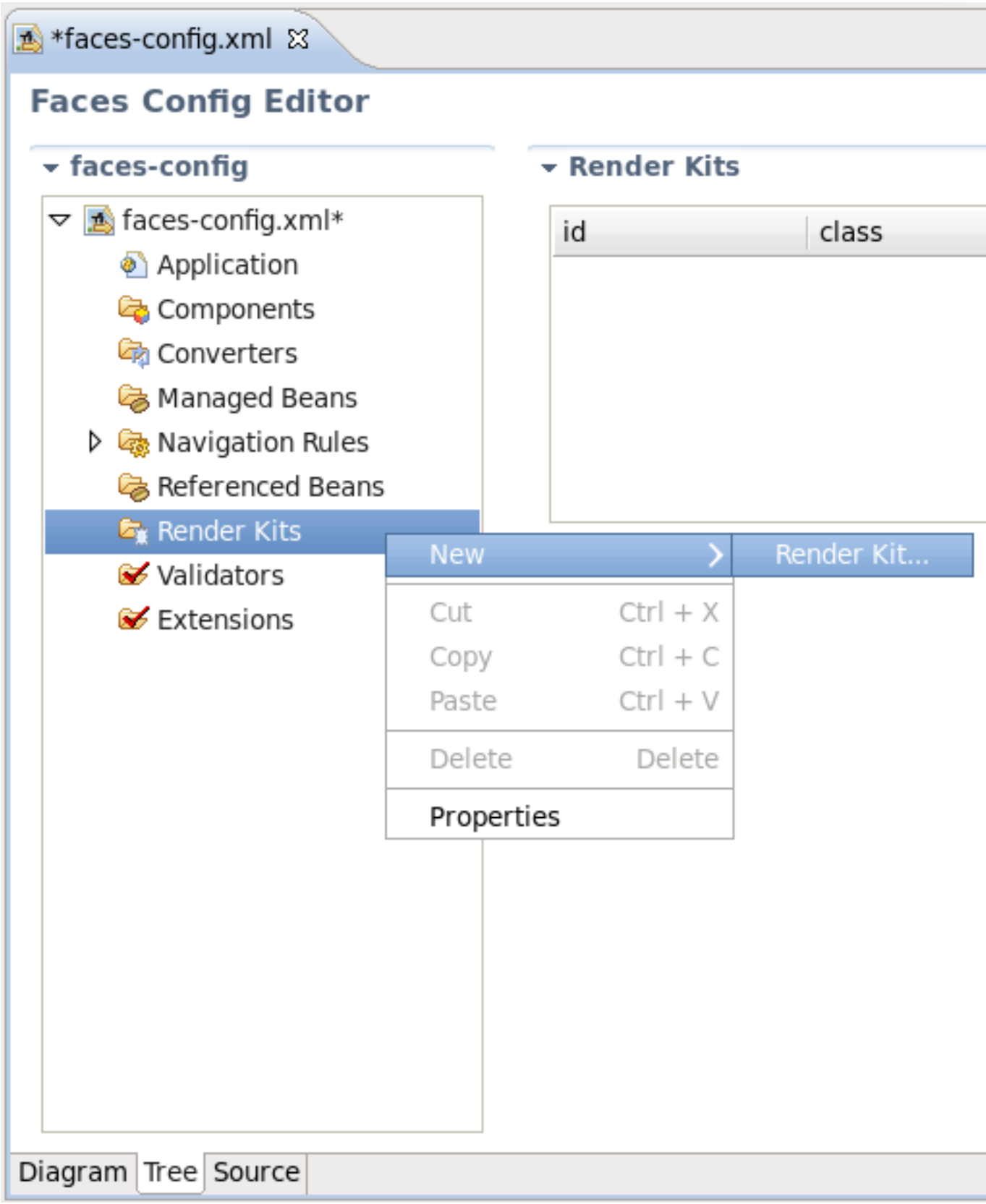

Figure 5.9. Adding a New JSF Renderer Kit to the JSF Configuration File

- Under the **Converters** node you can create a converter class for your JSF application either with an id or for a proper class. For more information on this procedure see *Section 7.1, “Create and Register a Custom Converter”*.

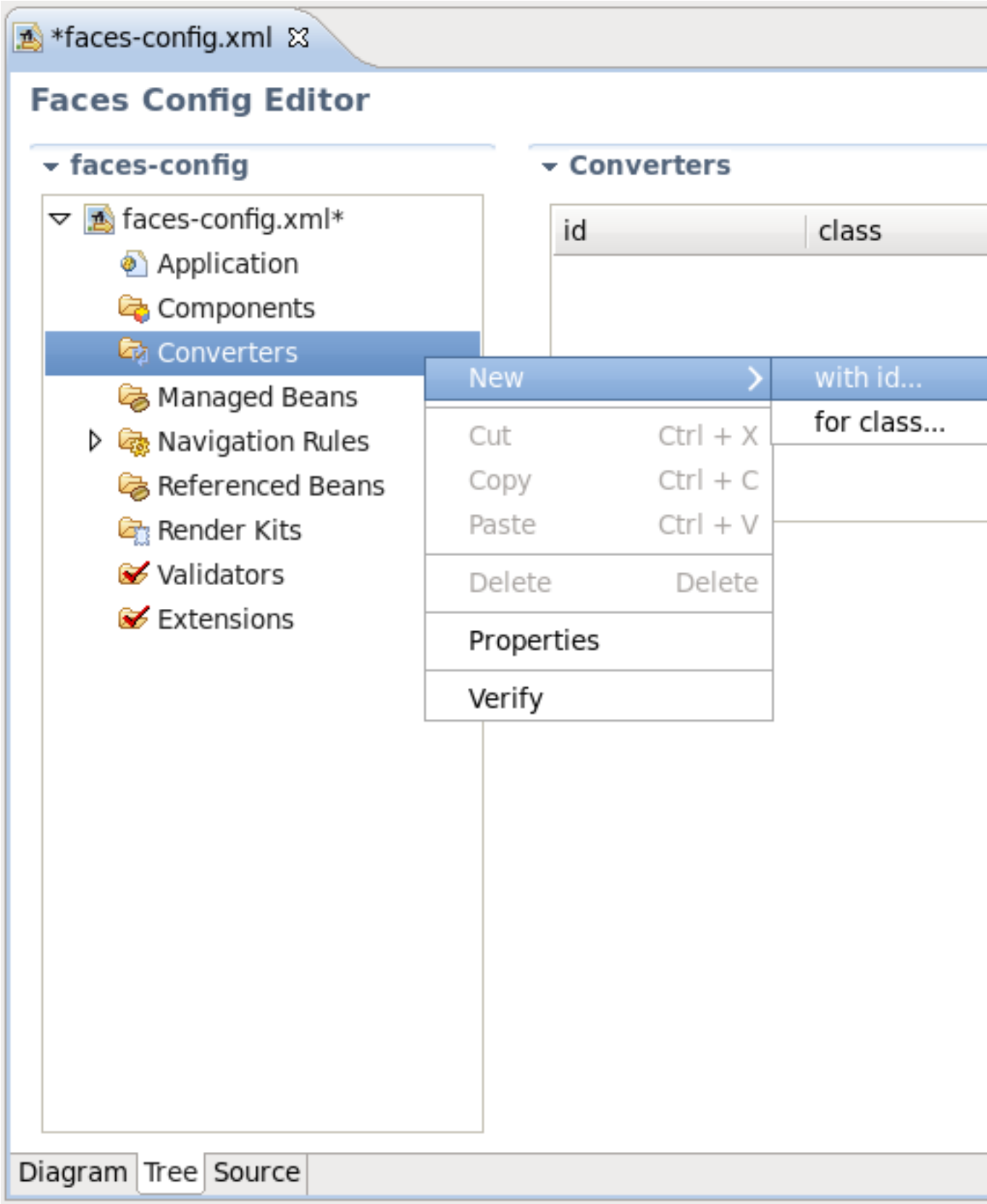

Figure 5.10. Creating a New Custom Converter

- The **Managed Bean** node allows you to create and register Bean classes in your JSF application. Read more on the topic in *Chapter 6, Managed Beans*.

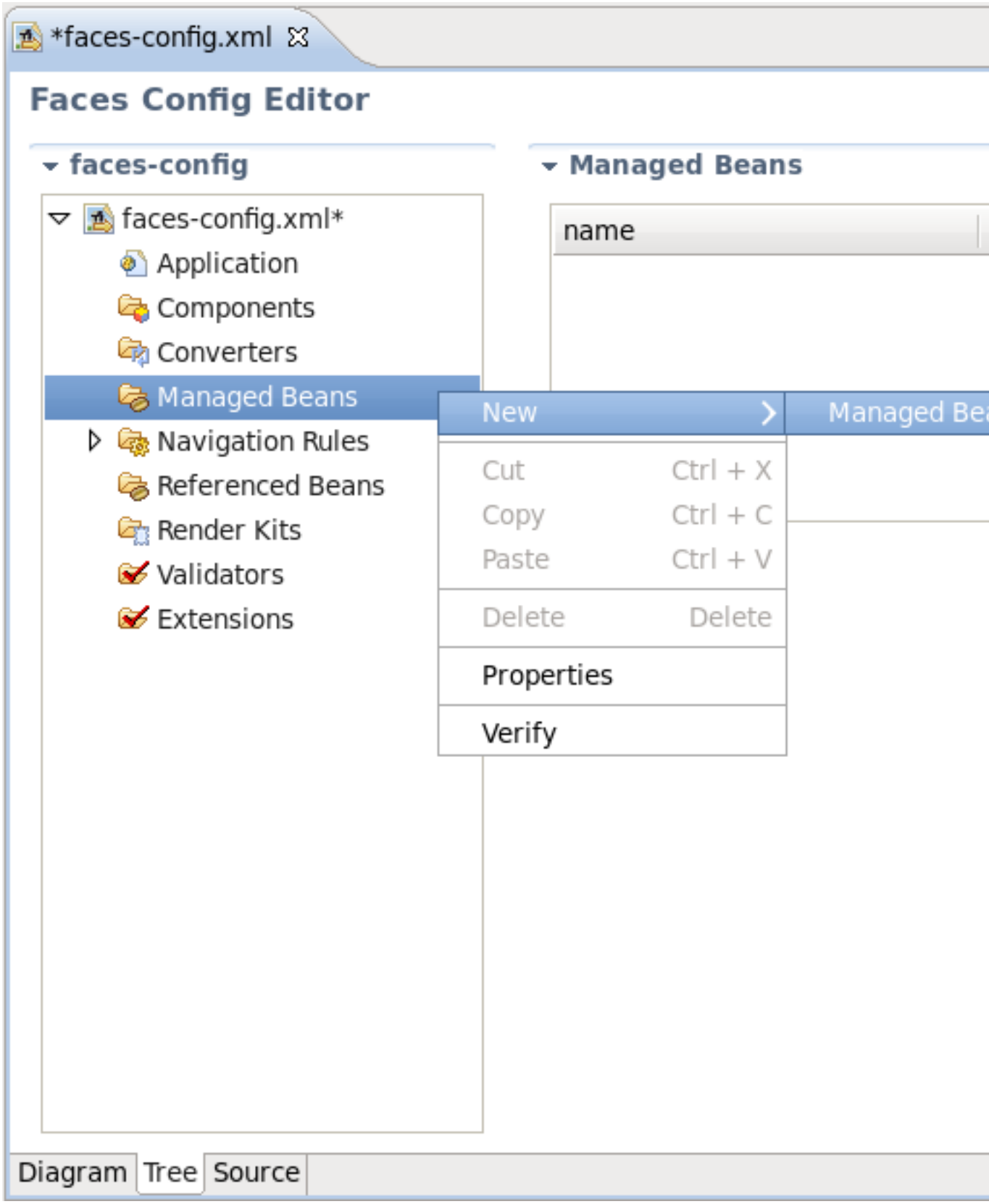

Figure 5.11. Managed Beans

- Use the **Navigation Rules** node to configure a navigation between the pages in your application. Here you can create a new navigation rule and adjust necessary properties for it in the right-hand area.

Tip:

The same you can do in the Diagram view of the JSF Configuration file editor (see *Section 5.1, "Diagram view"*).

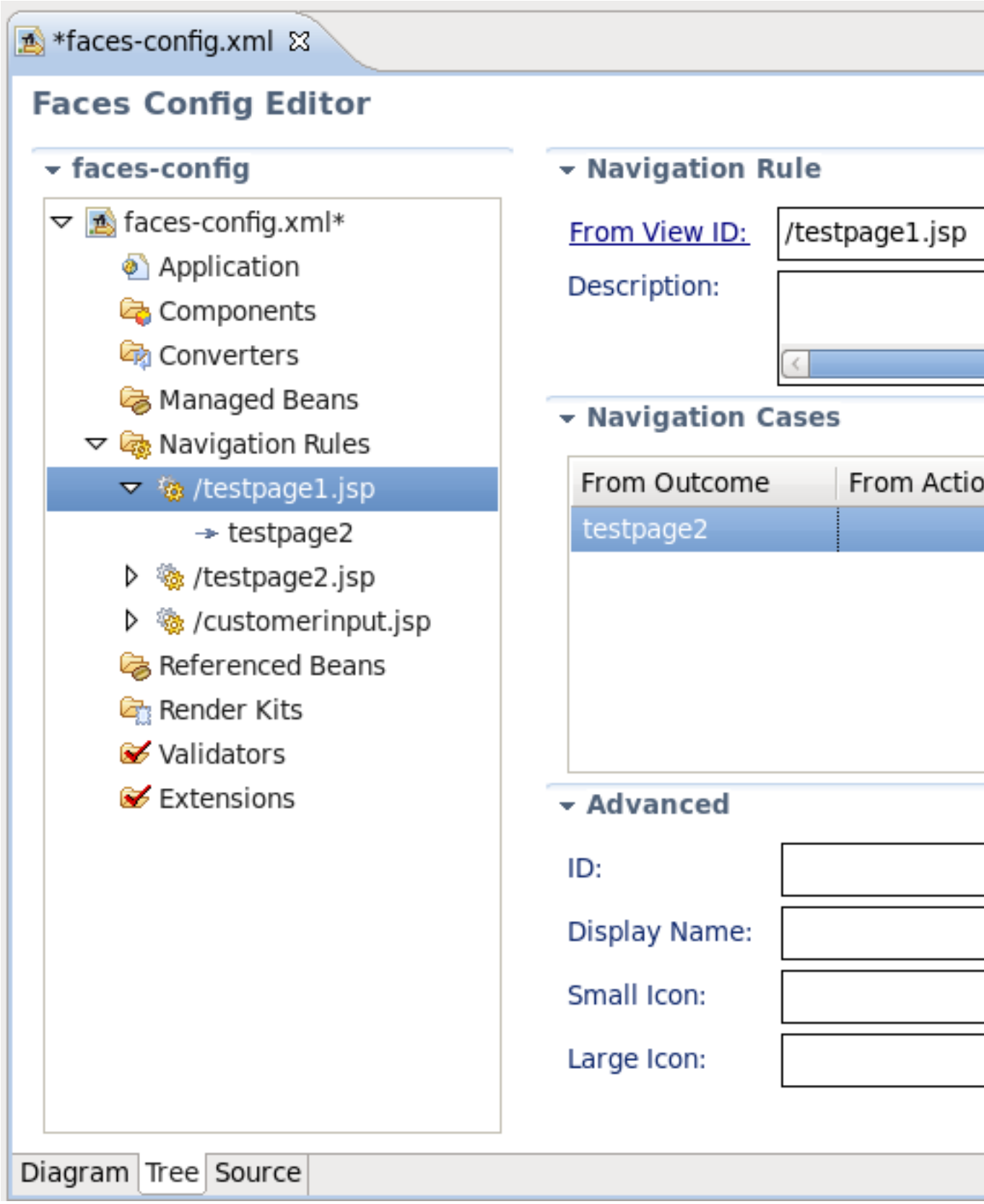

Figure 5.12. Configuring Navigation Rules

- Under the **Referenced Beans** node you can add a new Referenced Bean and configure various properties for it. To learn more on this refer to *Section 7.3, “Create and Register Referenced Beans”*.

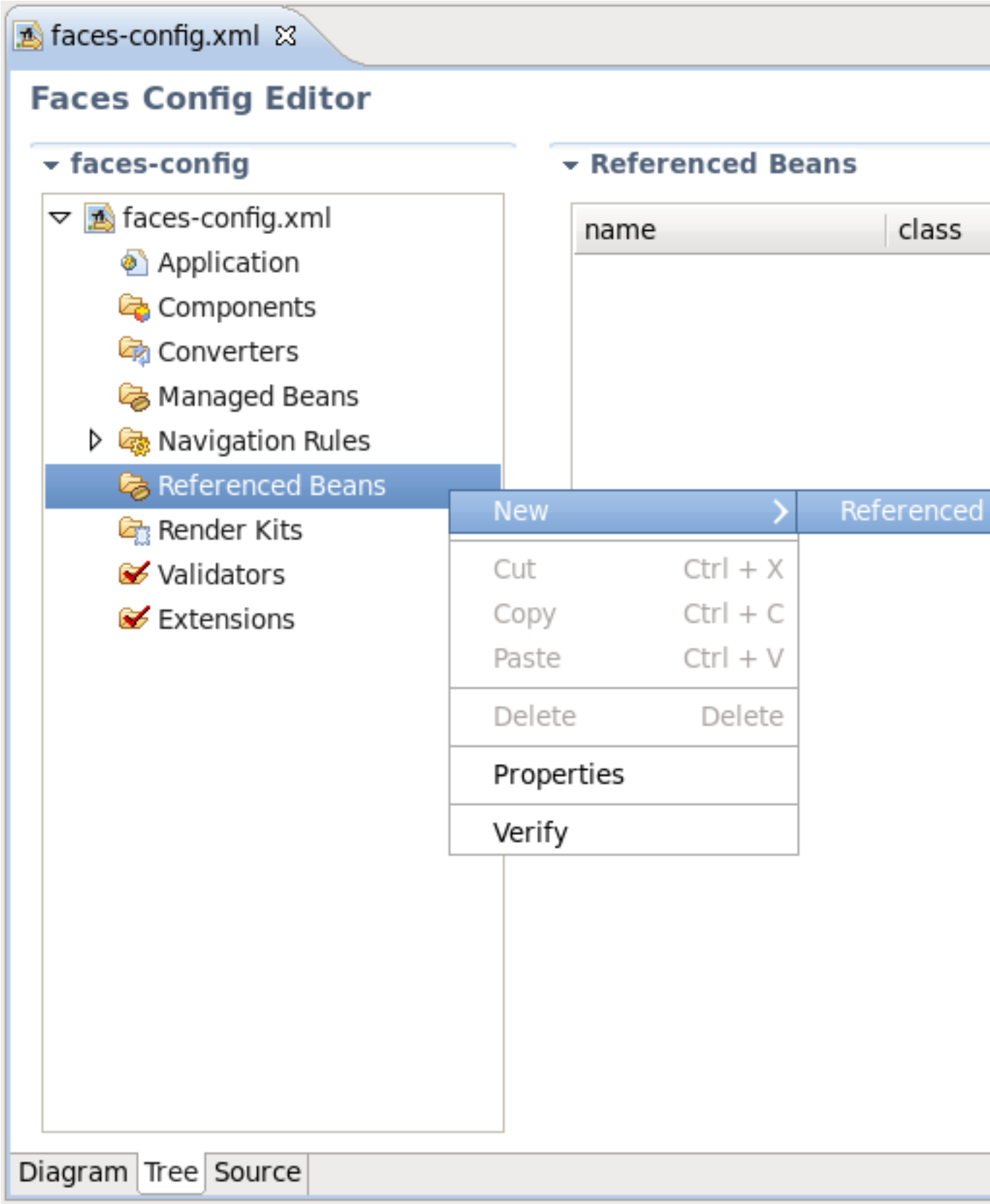

Figure 5.13. Referenced Beans

- The **Validators** node is needed to create validator classes for organizing the validation of your application data. You can read more on the topic in *Section 7.2, “Create and Register a Custom Validator”*.

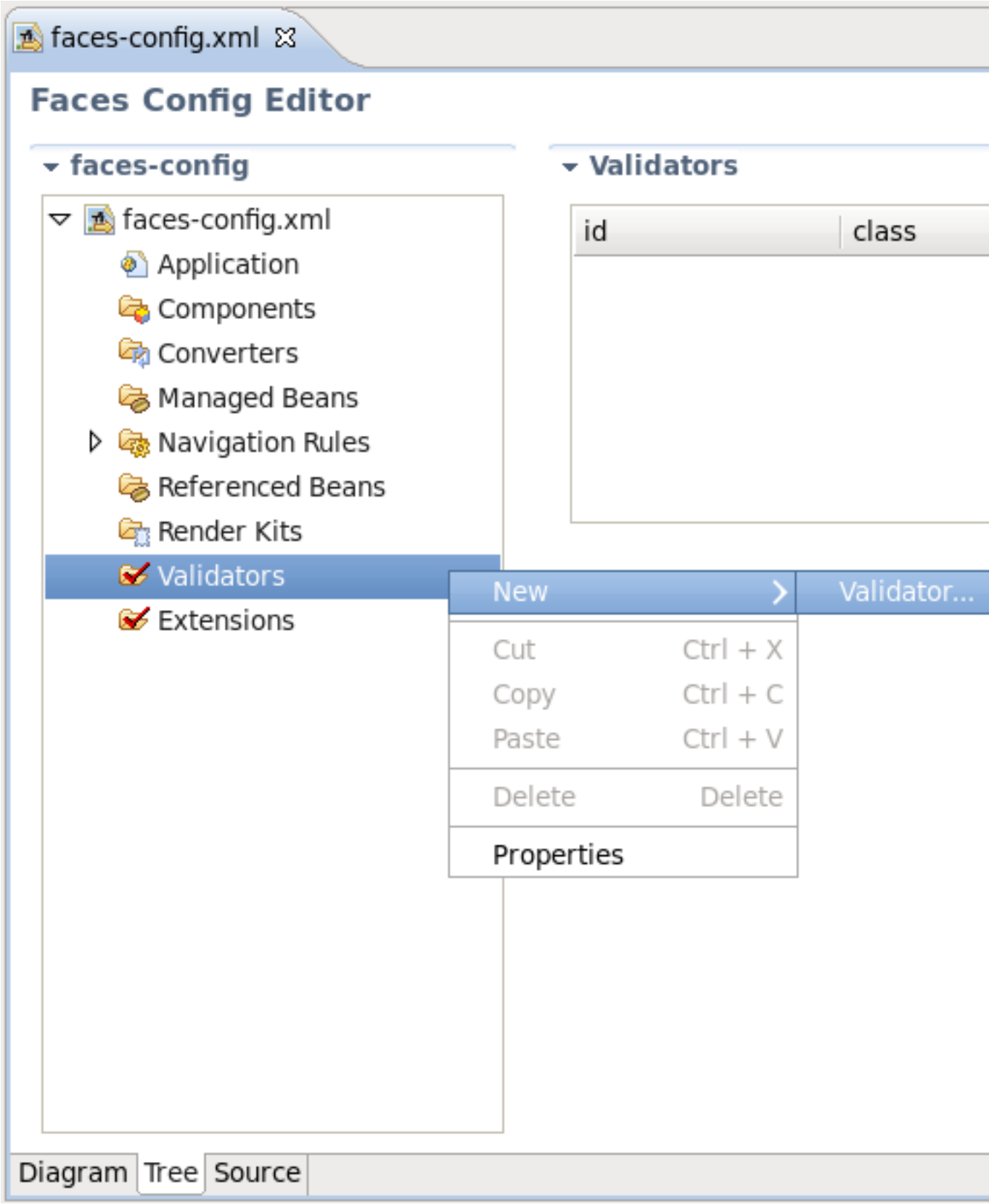

Figure 5.14. Validators

- The **Extensions** node is for setting extensions in your `faces-config.xml` file.

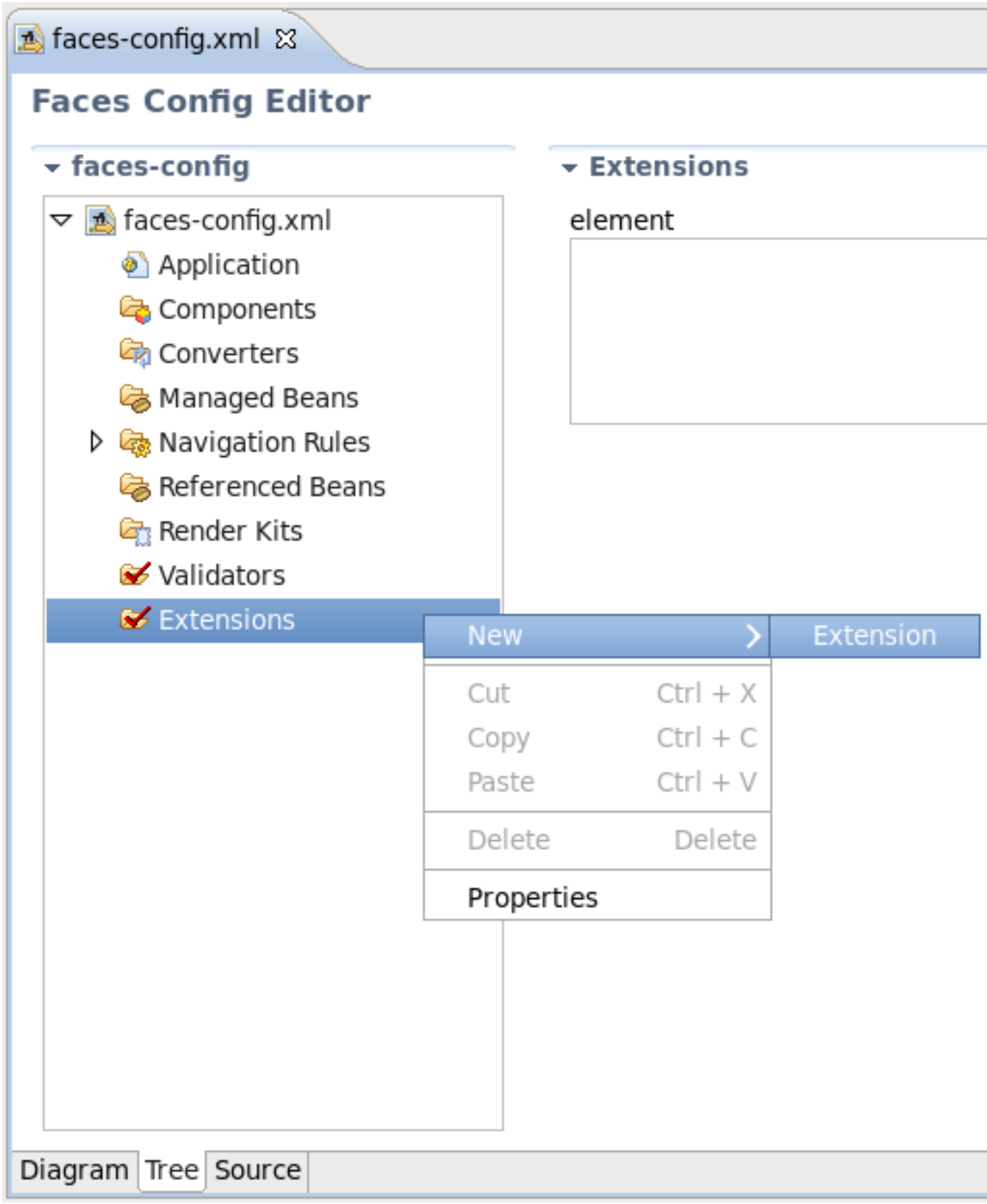

Figure 5.15. Adding Extensions

In the **Tree view** you can also edit the properties of the selected element with the help of the **Properties view** as shown below:

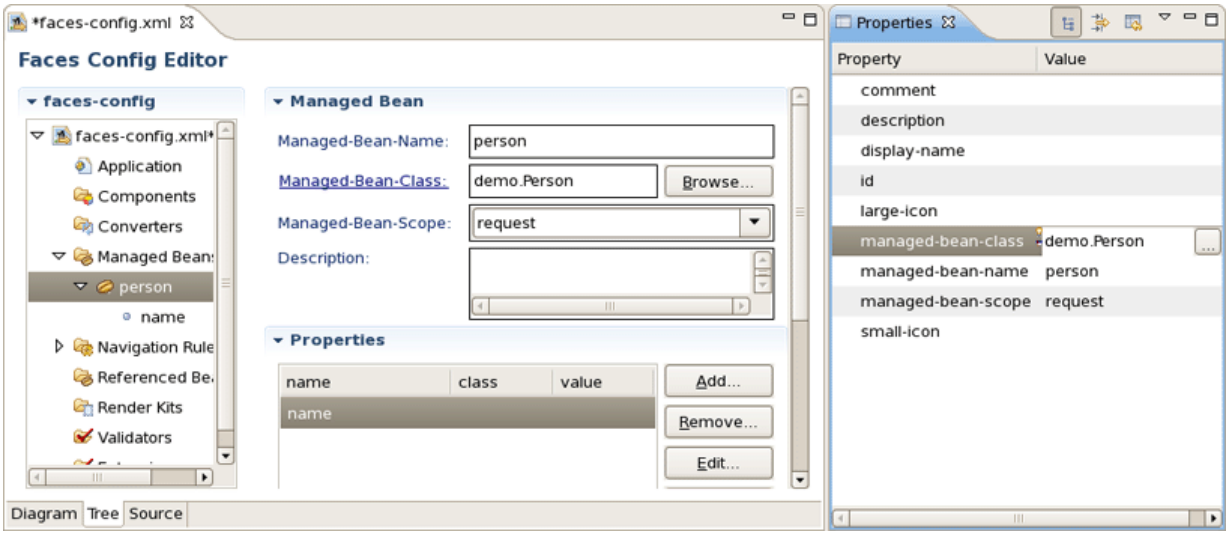

Figure 5.16. Properties View

5.3. Source View

Here, we'll discuss how you can configure your `faces-config.xml` file with the help of the **Source** View.

The **Source** View for the editor displays the text content of the JSF configuration file. It is always synchronized with other two views, so any changes made in one of the views will immediately appear in the other:

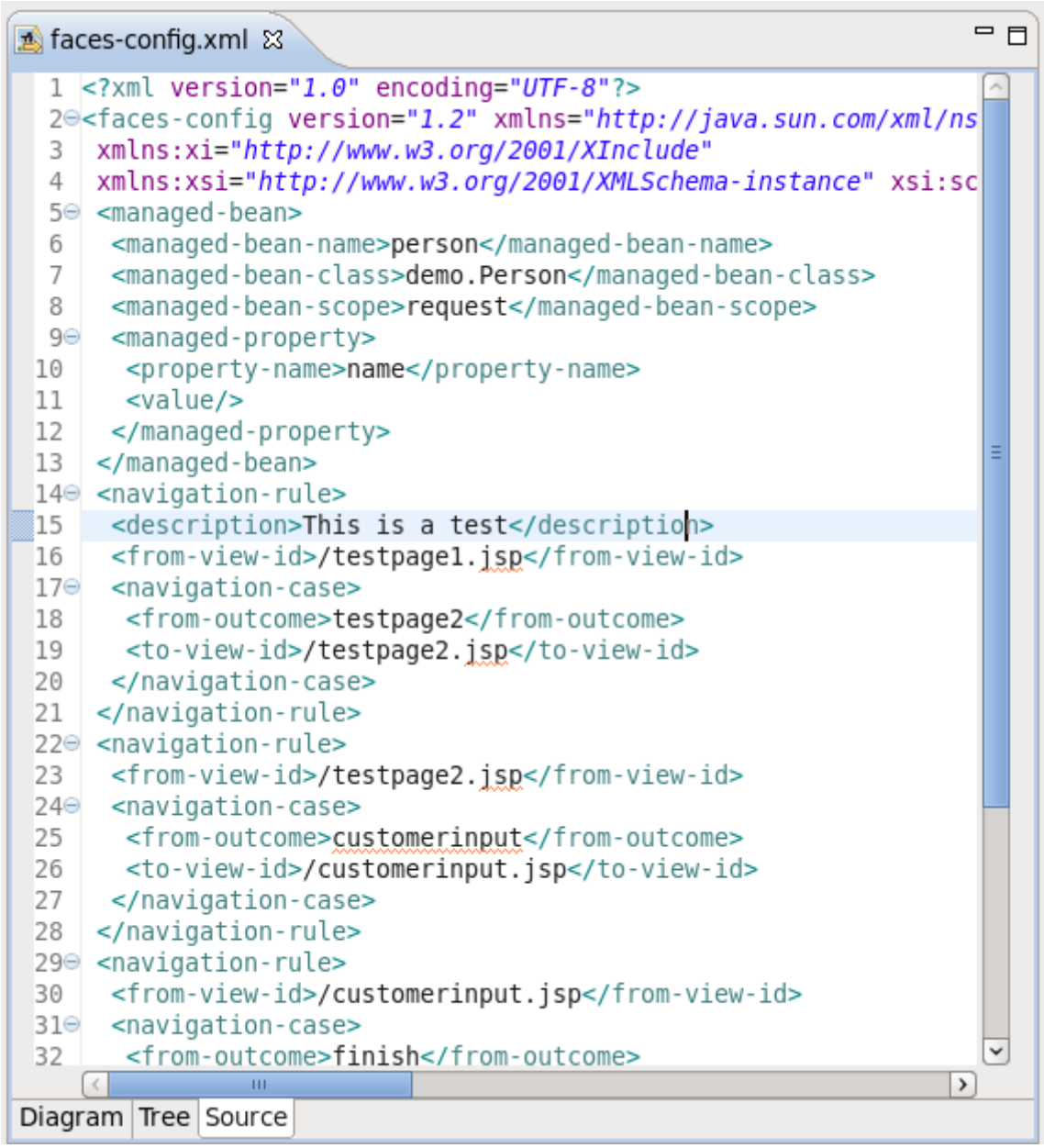

Figure 5.17. Source View

You can also work in the **Source** View with the help of the **Outline** View. The **Outline view** shows a tree structure of the JSF configuration file. Simply select any element in the **Outline** View, and it will jump to the same place in the Source editor, so you can navigate through the source code with **Outline** View.

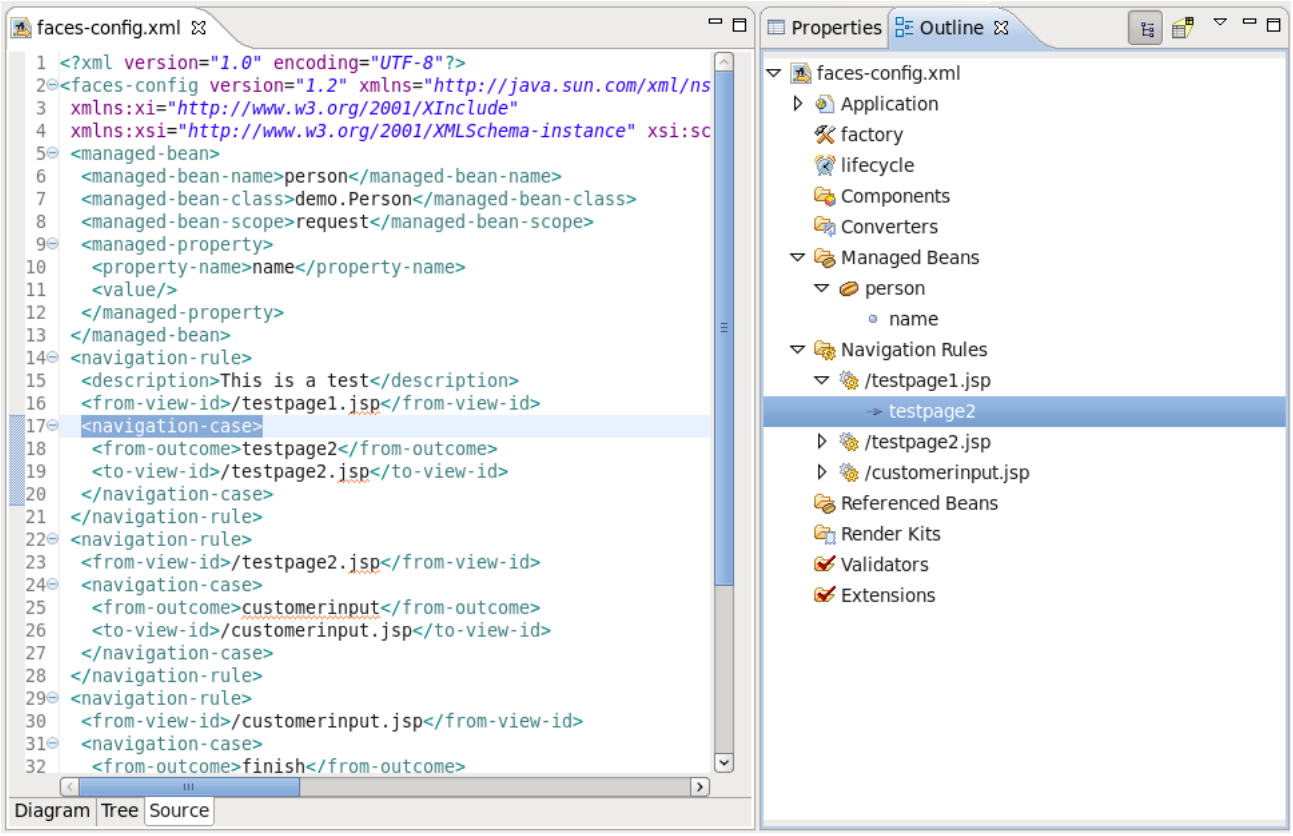

Figure 5.18. Outline View

5.4. Editor Features

Here we'll discuss a very important features that JSF configuration file editor provides when working with JSF resources.

5.4.1. Open On

The JSF configuration file editor comes with the very useful OpenOn navigation feature. You can find more information on this feature in the Visual Web Tools Guide.

5.4.2. Code Assist

Code Assist provides a pop-up tip to help you complete your code statements. It allows you to write your code faster and with more accuracy.

Code assist is always available in the Source mode:

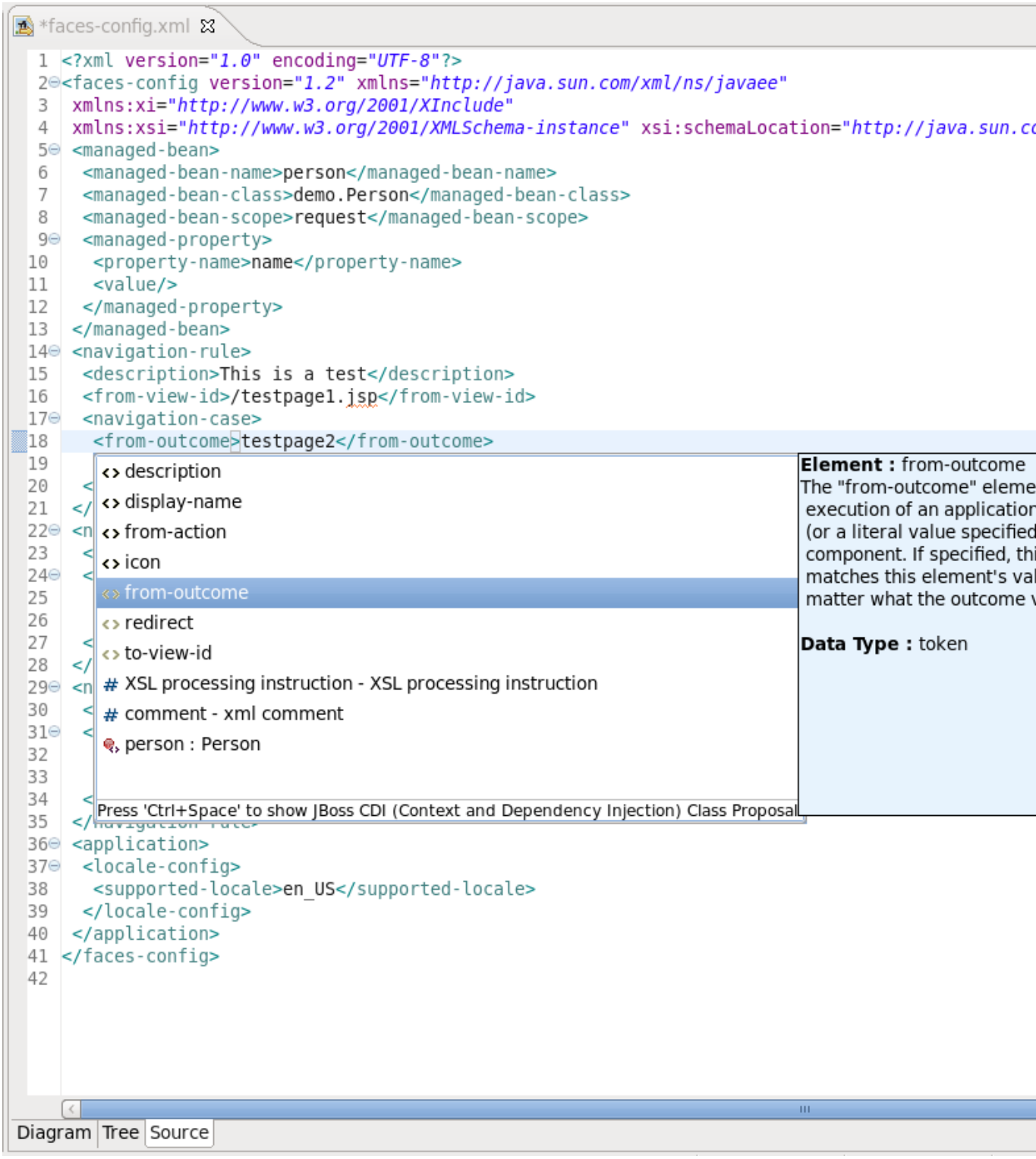

Figure 5.19. Code Assist in Source View

5.4.3. Error Reporting

Constant error checking is provided while you are developing your project. This greatly reduces your development time as it allows you to catch many errors during the development process.

Errors will be reported by *Chapter 8, JSF Project Verification* facility:

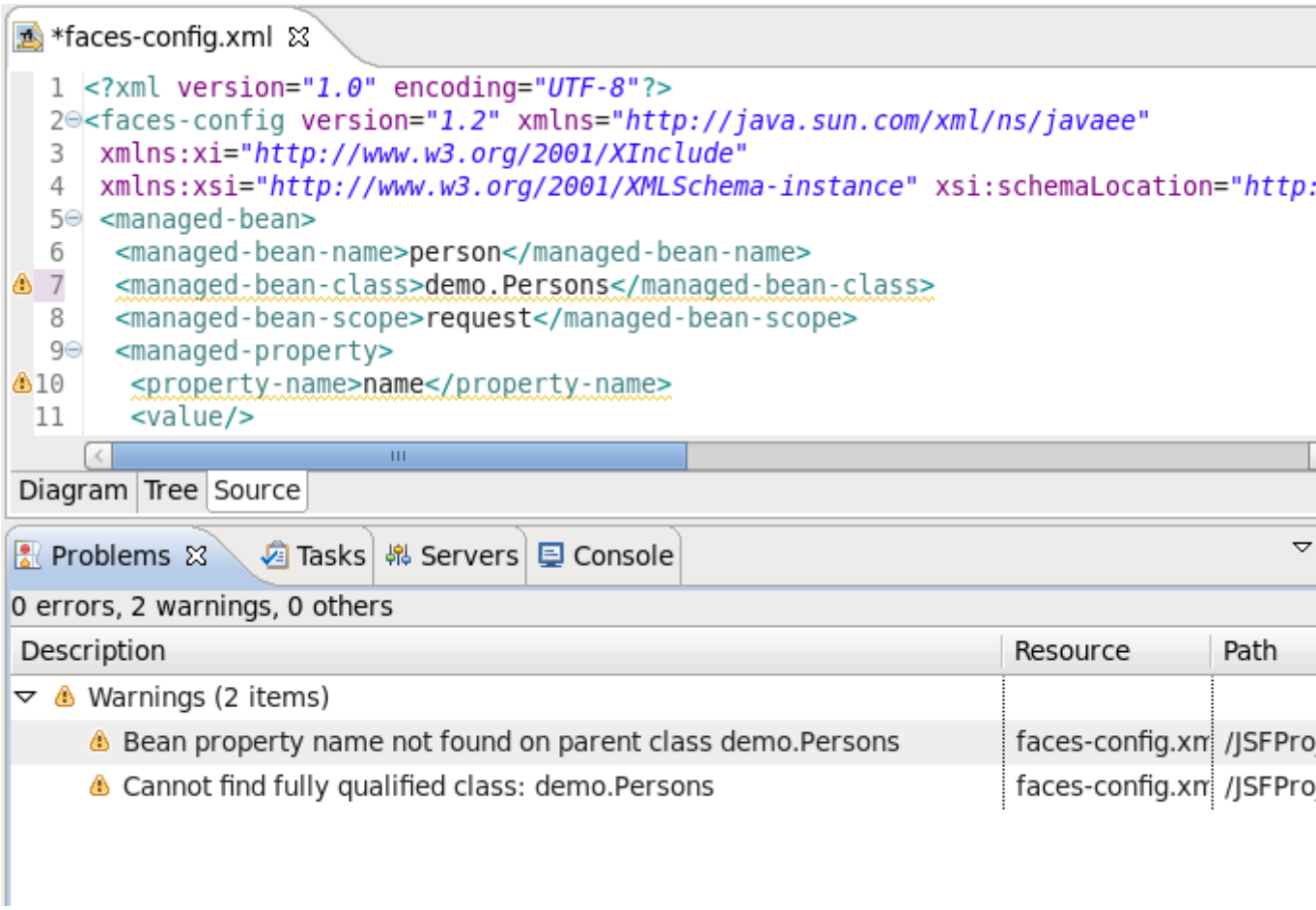

Figure 5.20. Error Reporting in Source View

Other errors are also reported.

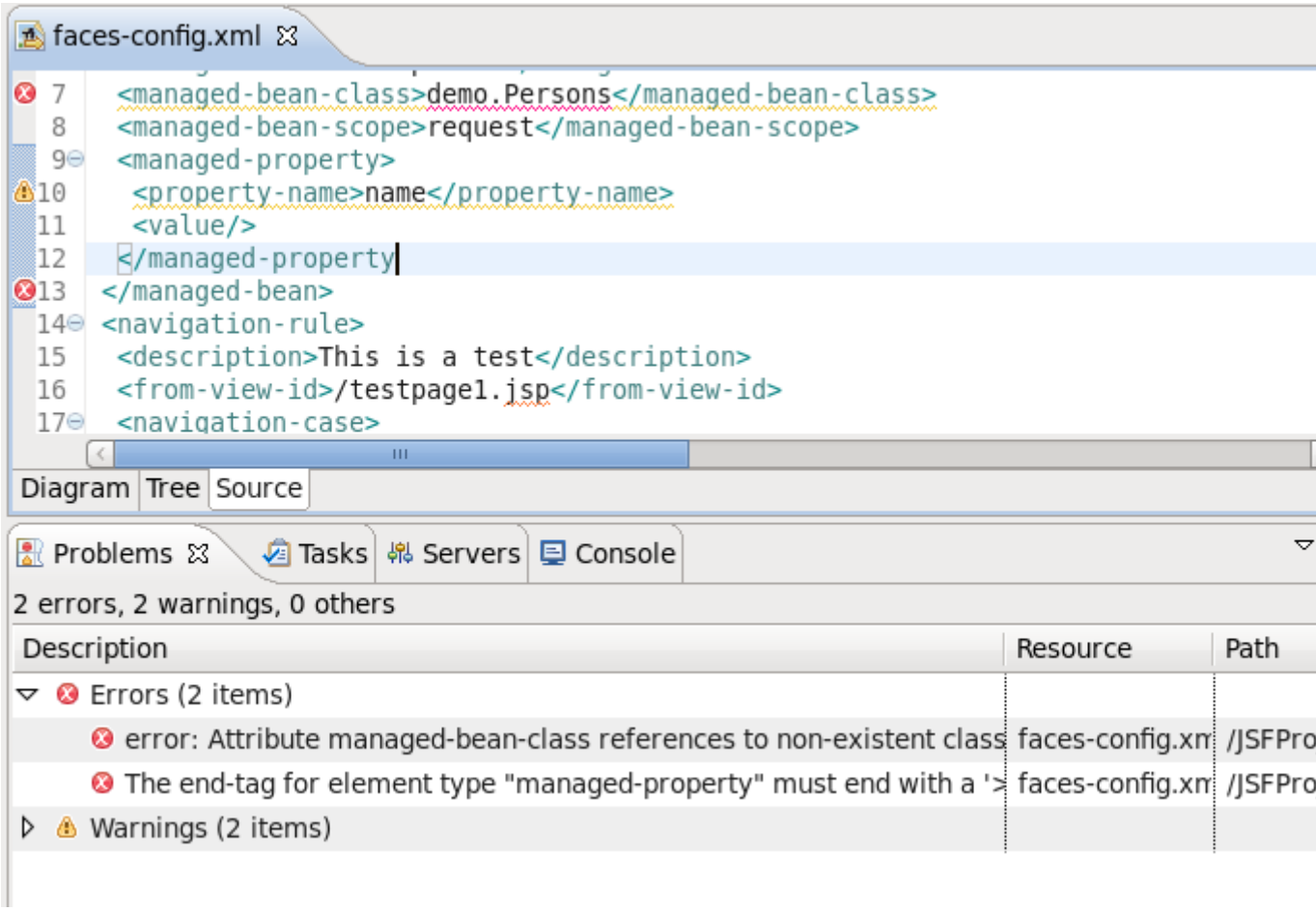

Figure 5.21. Other Errors Reporting

Managed Beans

JSF Tools provides a number of useful features when working with managed beans, such as:

- Adding and generating code for new managed beans
  - Generating code for attributes and getter/setter methods
- Adding existing managed beans to a JSF configuration file

This guide will look at each of these features in more detail.

6.1. Code Generation for Managed Beans

To begin, create a new managed bean in JSF configuration file editor using the **Tree view**.

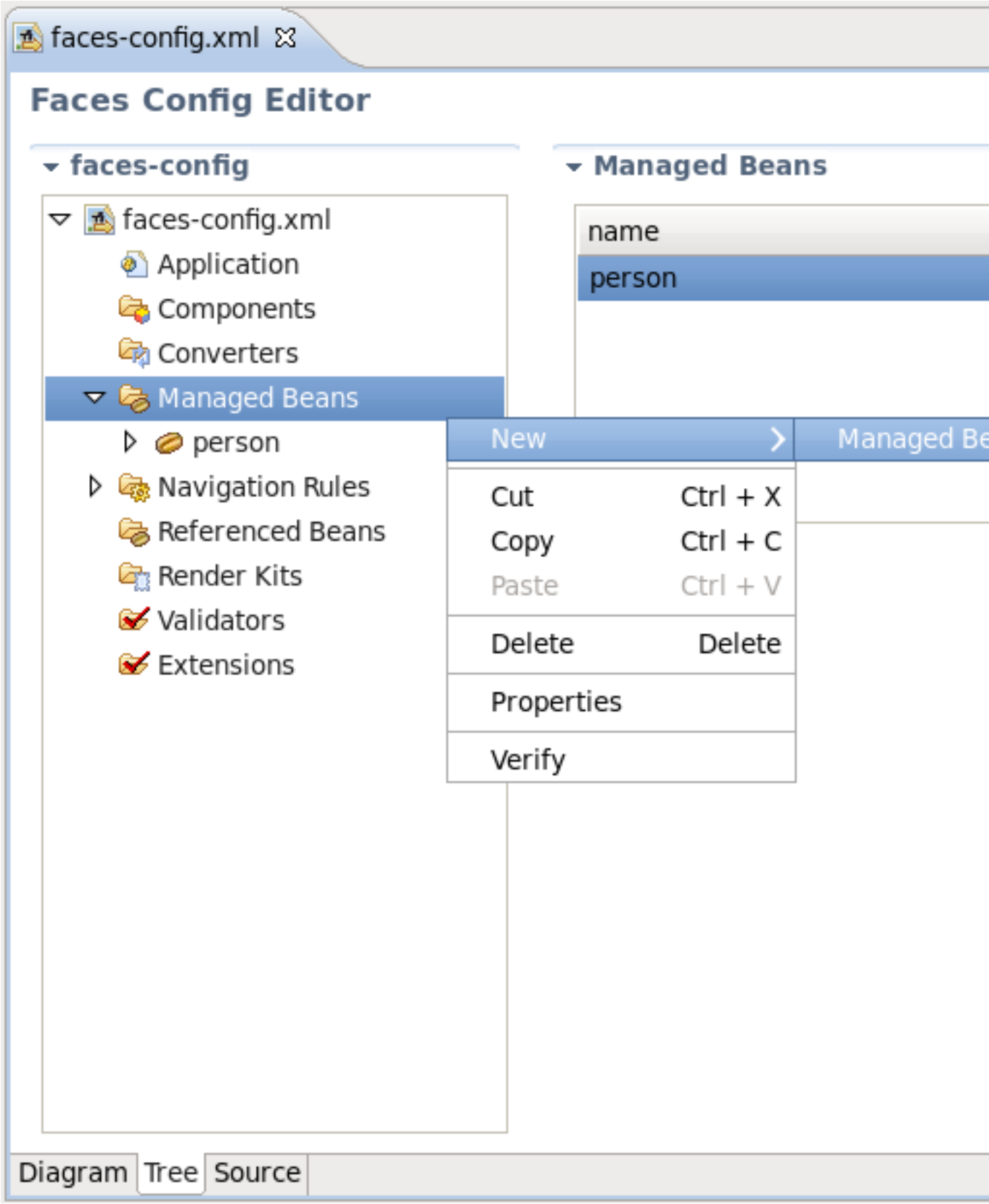

Figure 6.1. Creation of New Managed Bean

Note:

When you define a new managed bean, make sure that **Generate Source Code** option is checked as shown in the figure below.

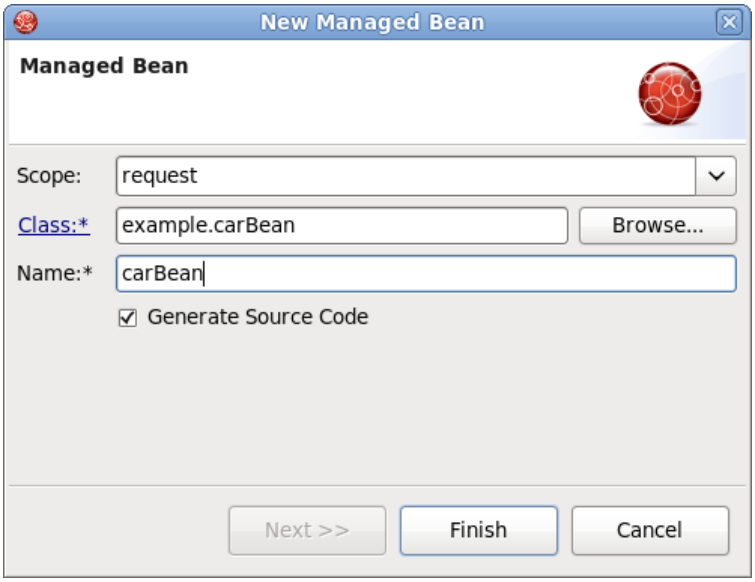

Figure 6.2. New Managed Bean

After the Java class has been generated you can open it for additional editing. There are two ways to open a Java class:

- Click on the **Managed-Bean-Class** link in the editor.

- Right click the *managed bean* and select the **Open Declaration** option.

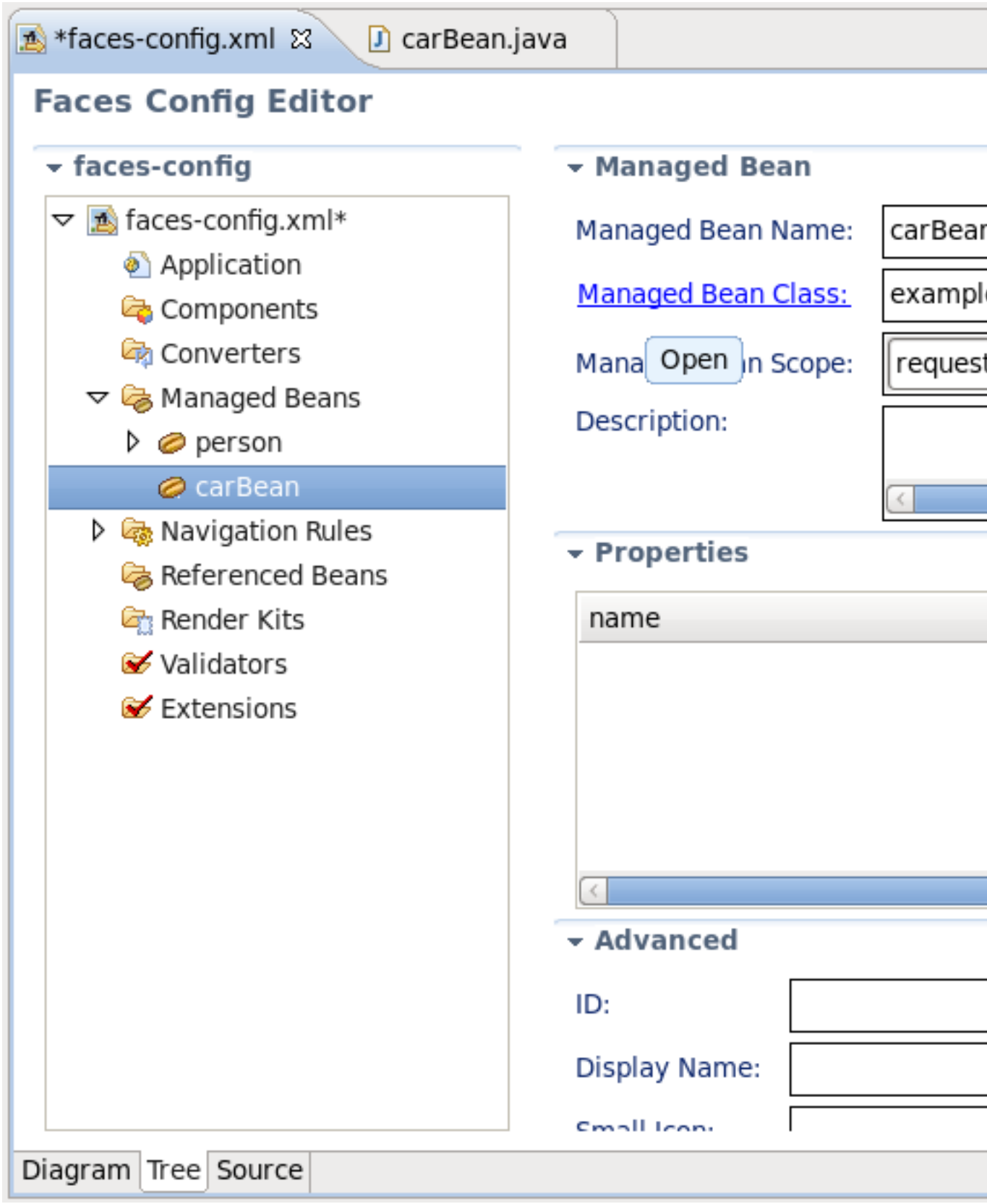

Figure 6.3. Opening of Created Managed Bean

The generated Java source should look as follows:

*faces-config.xml carBean.java

```
1  /**..
4  package example;
5
6  /**
7   * @author matthew
8   *
9   */
10 public class carBean {
11     public carBean() {
12     }
13 }
14
```

Figure 6.4. Java Source Code

You can also generate source code for properties, also includes getter and setter methods. Right click on the bean and select **New** → **Property**. You will then see the **Add Property** dialog.

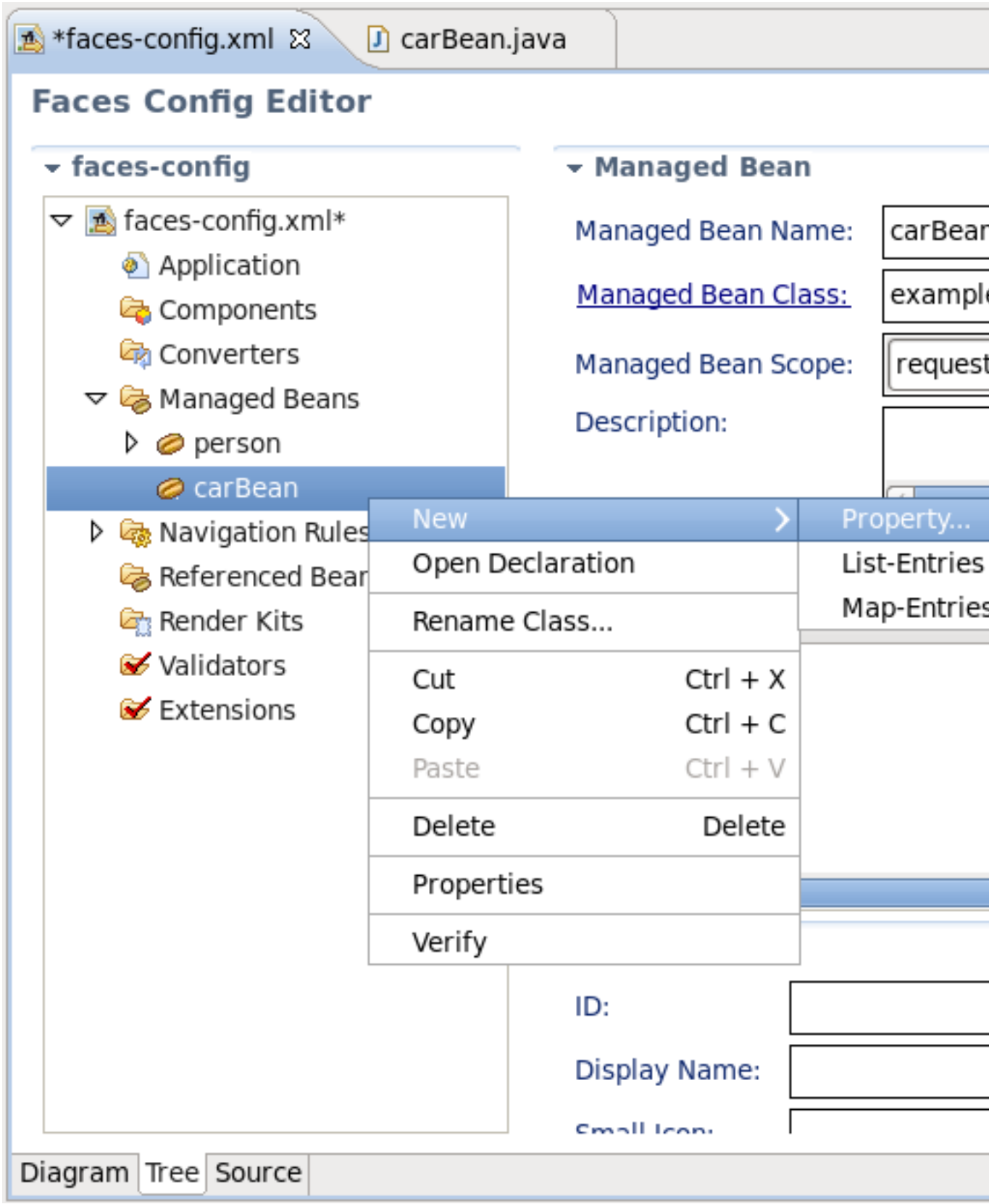

Figure 6.5. Generation of Source Code for Properties

When the form is open make sure that all the check boxes are selected:

- Add Java property
- Generate Getter
- Generate Setter

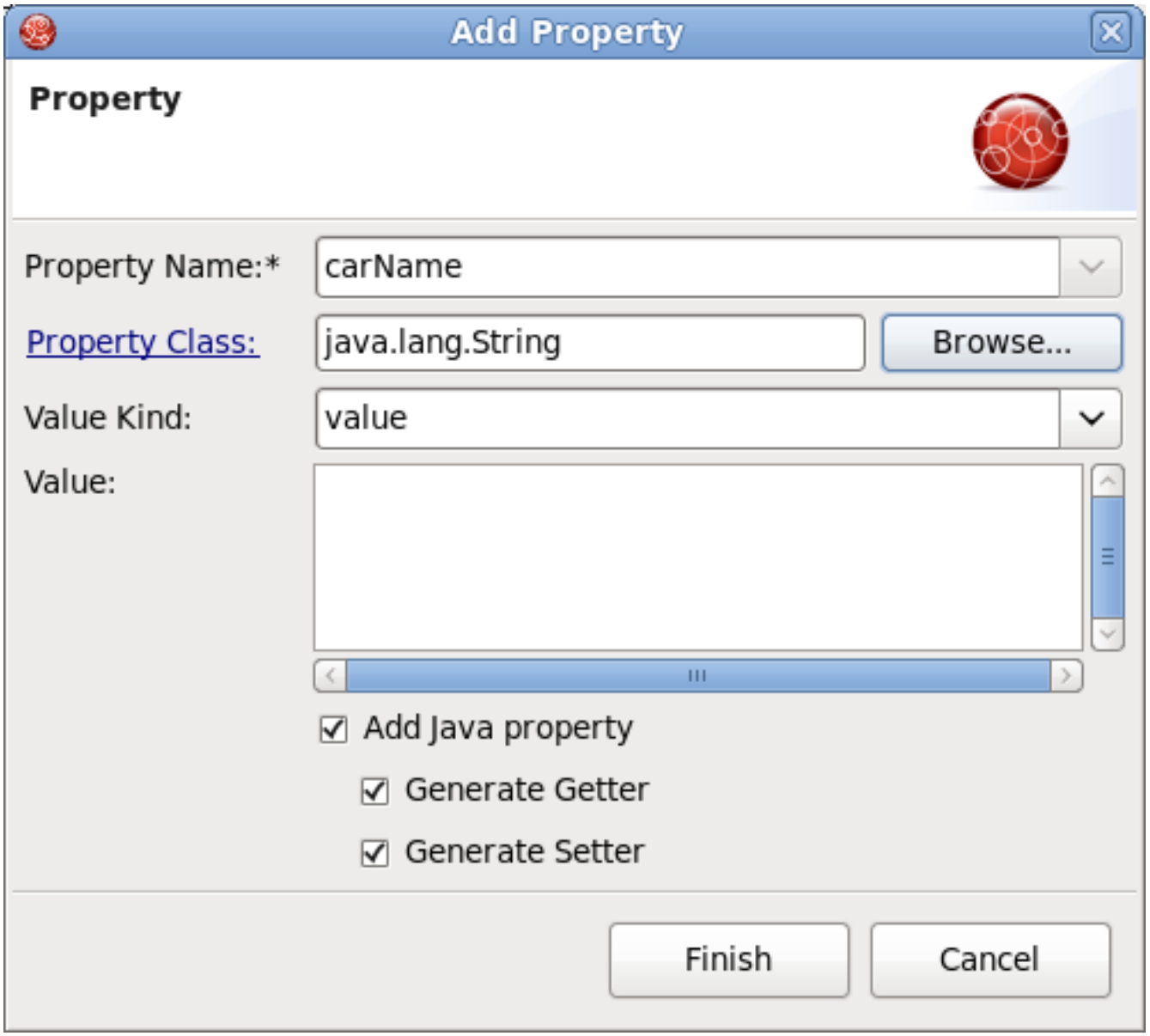

Figure 6.6. "Add Property" Form

Once the generation is complete, you can open the file and see the newly added property with accompanying "get" and "set" methods:

*faces-config.xml carBean.java

```
1  /**
2   *
3   */
4  package example;
5
6  /**
7   * @author matthew
8   *
9   */
10 public class carBean {
11     private java.lang.String carName;
12
13     public carBean() {
14     }
15
16     public java.lang.String getCarName() {
17         return carName;
18     }
19
20     public void setCarName(java.lang.String carName) {
21         this.carName = carName;
22     }
23 }
24
```

Figure 6.7. Generated Java Source Code for Property

This covers the options available when creating a new Managed Bean. The next section will show you how to add an existing Bean into a JSF configuration file.

6.2. Add Existing Java Beans to a JSF Configuration File

If you already have a Java bean you can easily add it to a JSF configuration file.

You should start the same way you create a new managed bean. Use the **Browse...** button to add your existing Java class.

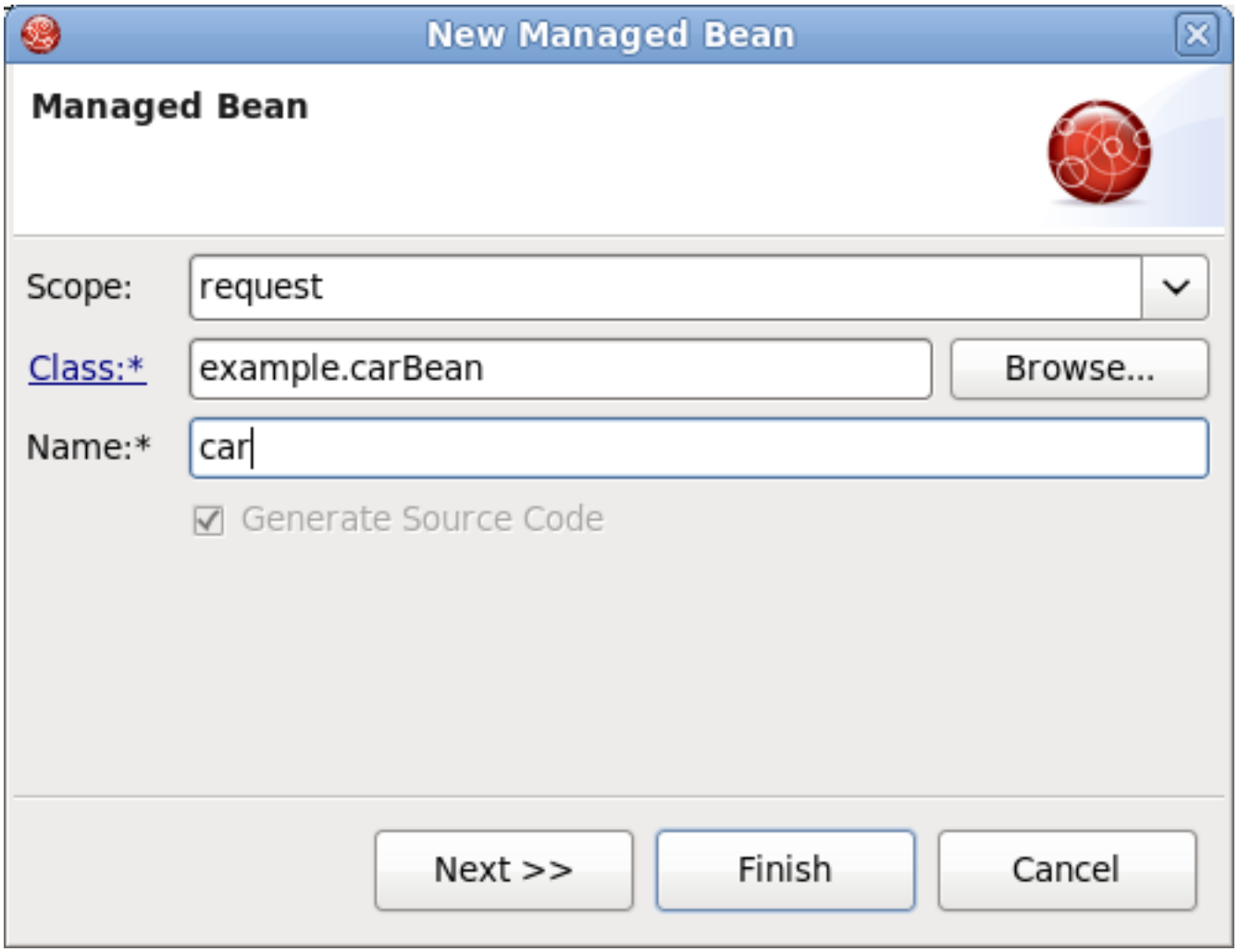

Figure 6.8. New Managed Bean Form

Once the class is set, its **Name** will be set as well. But you can easily substitute it for the other one. Notice that **Generate Source Code** option is not available as the Java class already exists.

After adding your class the **Next** button will be activated. When you click it you will be presented with the **Managed Properties** dialog where all corresponding properties are displayed. Checking the appropriate ones will add them into your JSF Configuration File.

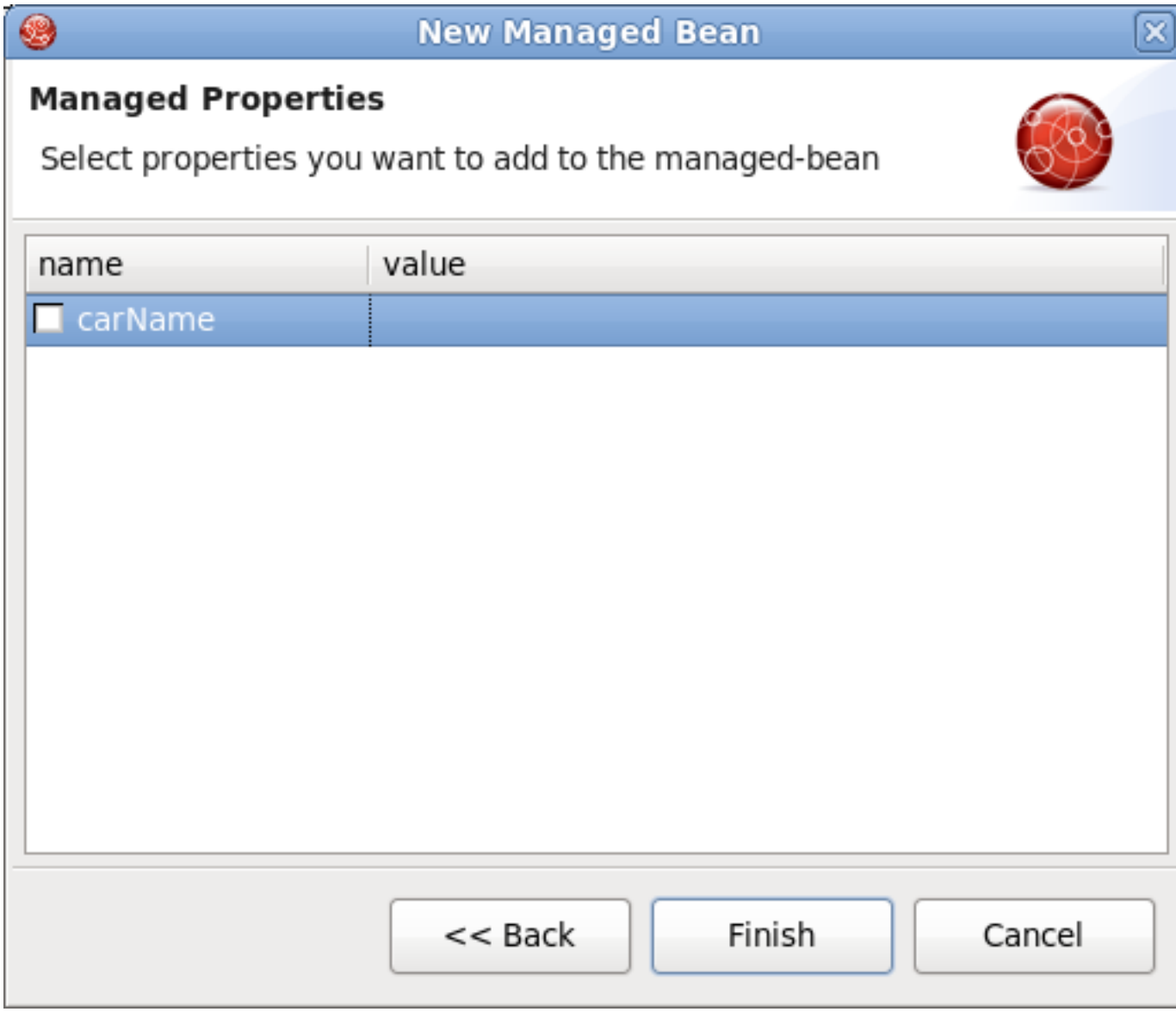

Figure 6.9. Selection of Bean's Properties.

If you don't want to add any, just click the **Finish** button.

The steps above have demonstrated how you can add an existing Bean to the JSF configuration file, i.e. `faces-config.xml`. The next chapter will demonstrate how to organize and register other kinds of artifacts.

Creation and Registration

7.1. Create and Register a Custom Converter

It's also possible to create a custom Converter in order to specify your own converting rules. Let's look at how you can do this.

To create and register a custom converter it is necessary perform the following steps:

- In the Project Explorer view open the `faces-config.xml` file and select **Tree** tab.

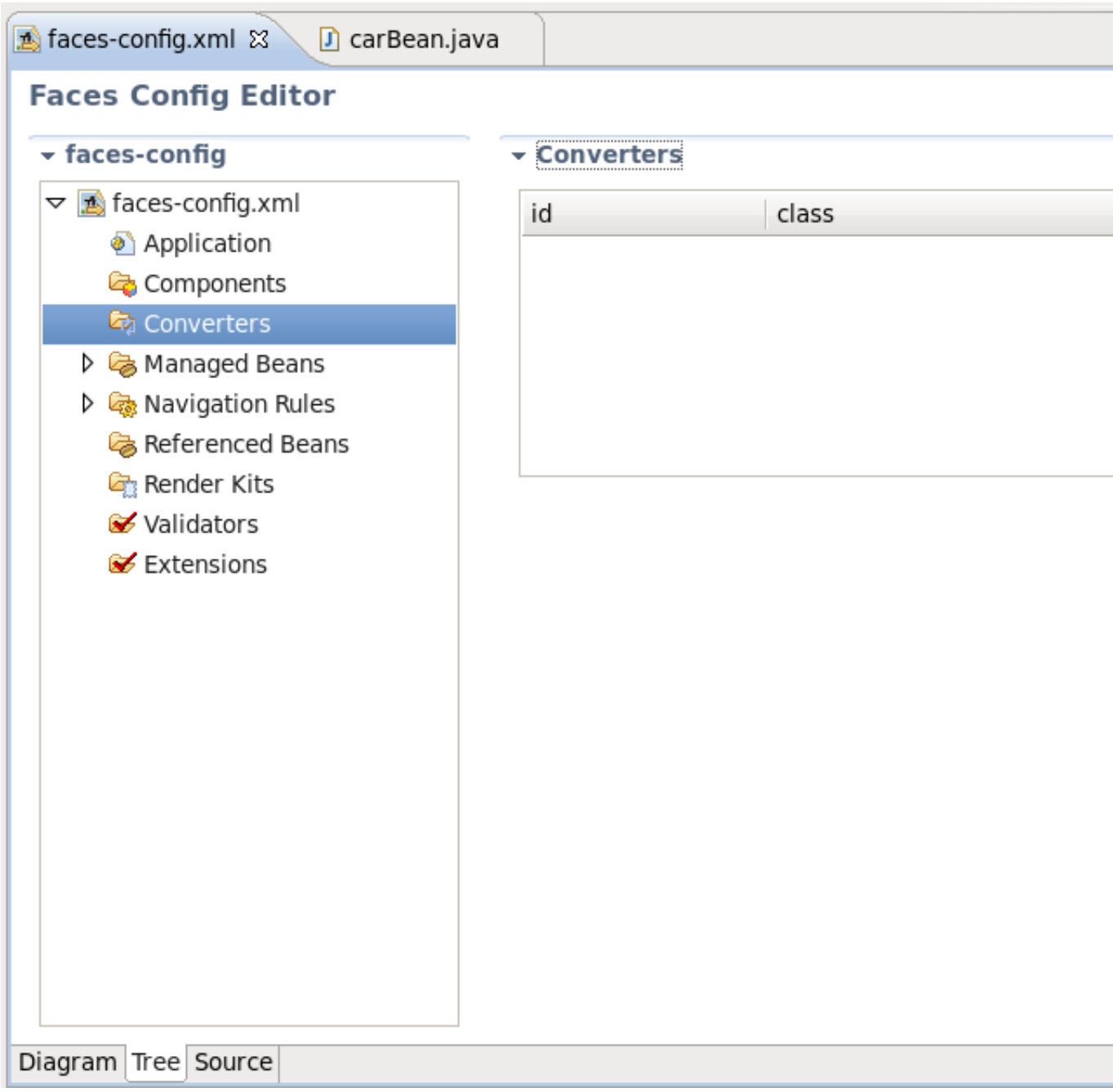

Figure 7.1. Converters

- Select **Converters** and click the **Add** button.
- On the form type the name of your converter in the *Converter-id* field and name of the class for converters. After clicking **Finish** button your custom converter is registered under the entered name.

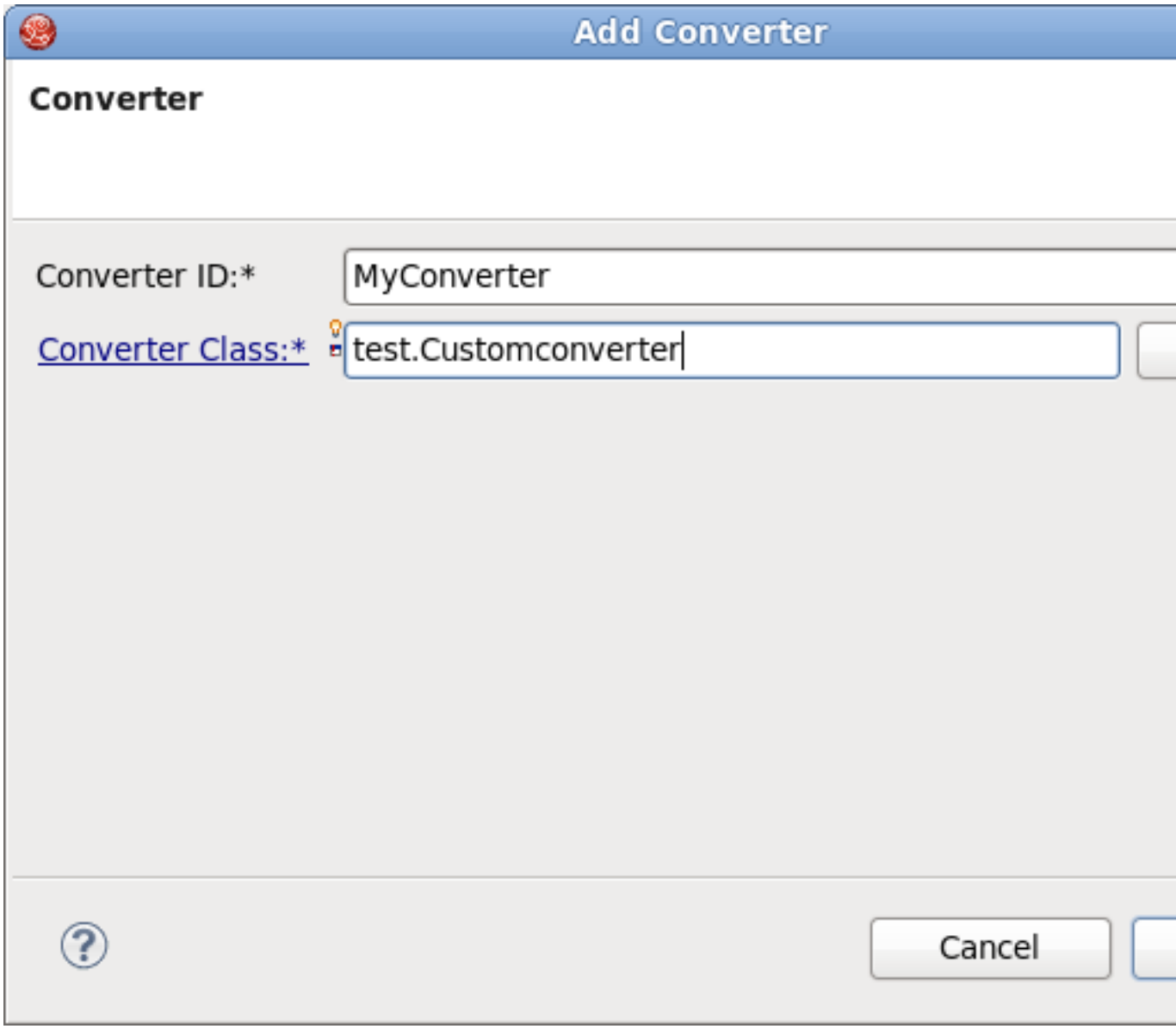

Figure 7.2. Add Converter Form

- Now you can create a *"converter"* class. In the Converter section you should see your **Converter-id** and **Converter-class**. Click on the **Converter-Class** link to generate the source code.

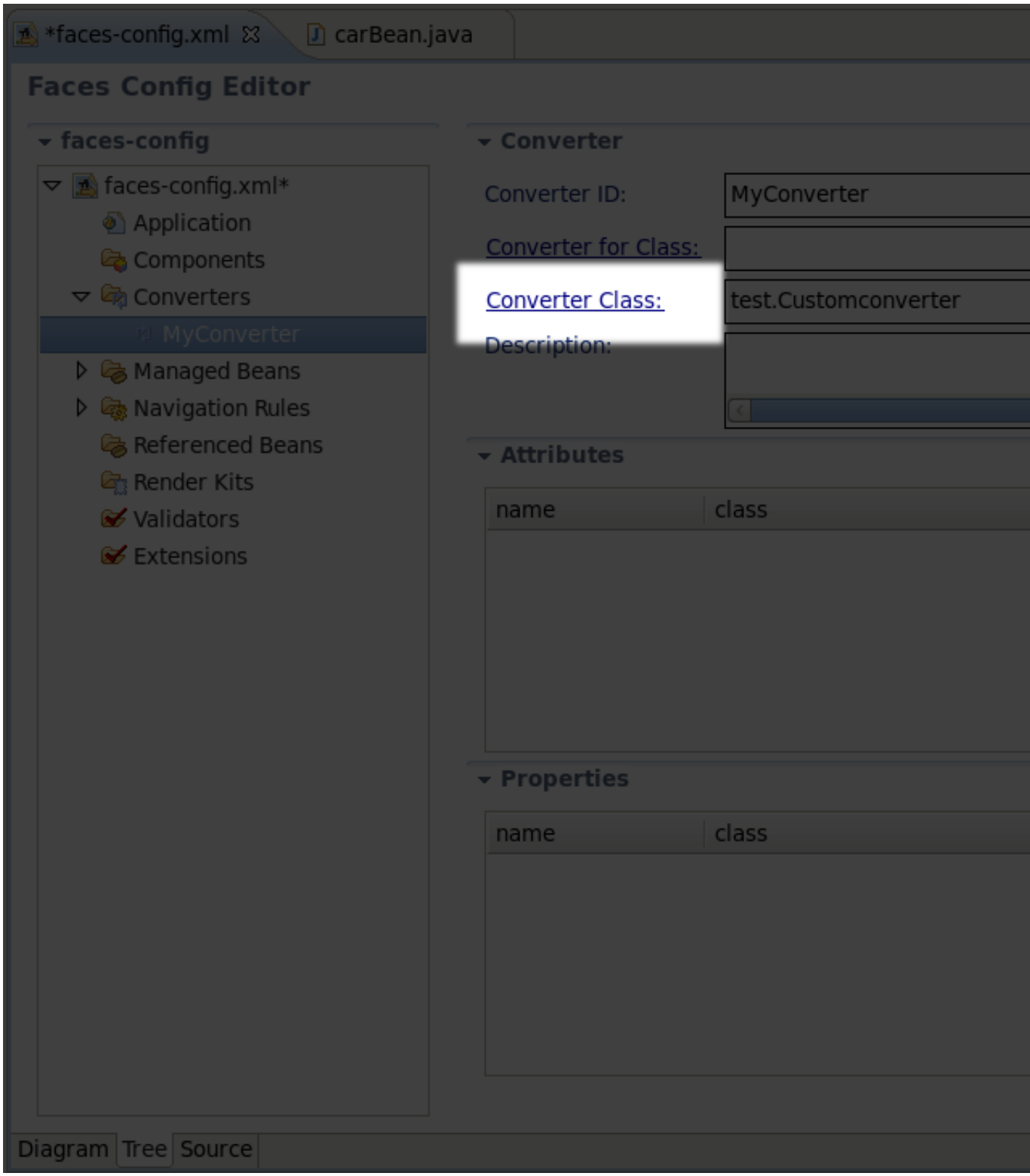

Figure 7.3. Generation of Source Code for Converter Class

- A usual wizard for creating a Java class will appear. All needed fields here will be adjusted automatically. Just leave everything without changes and click the **Finish** button.

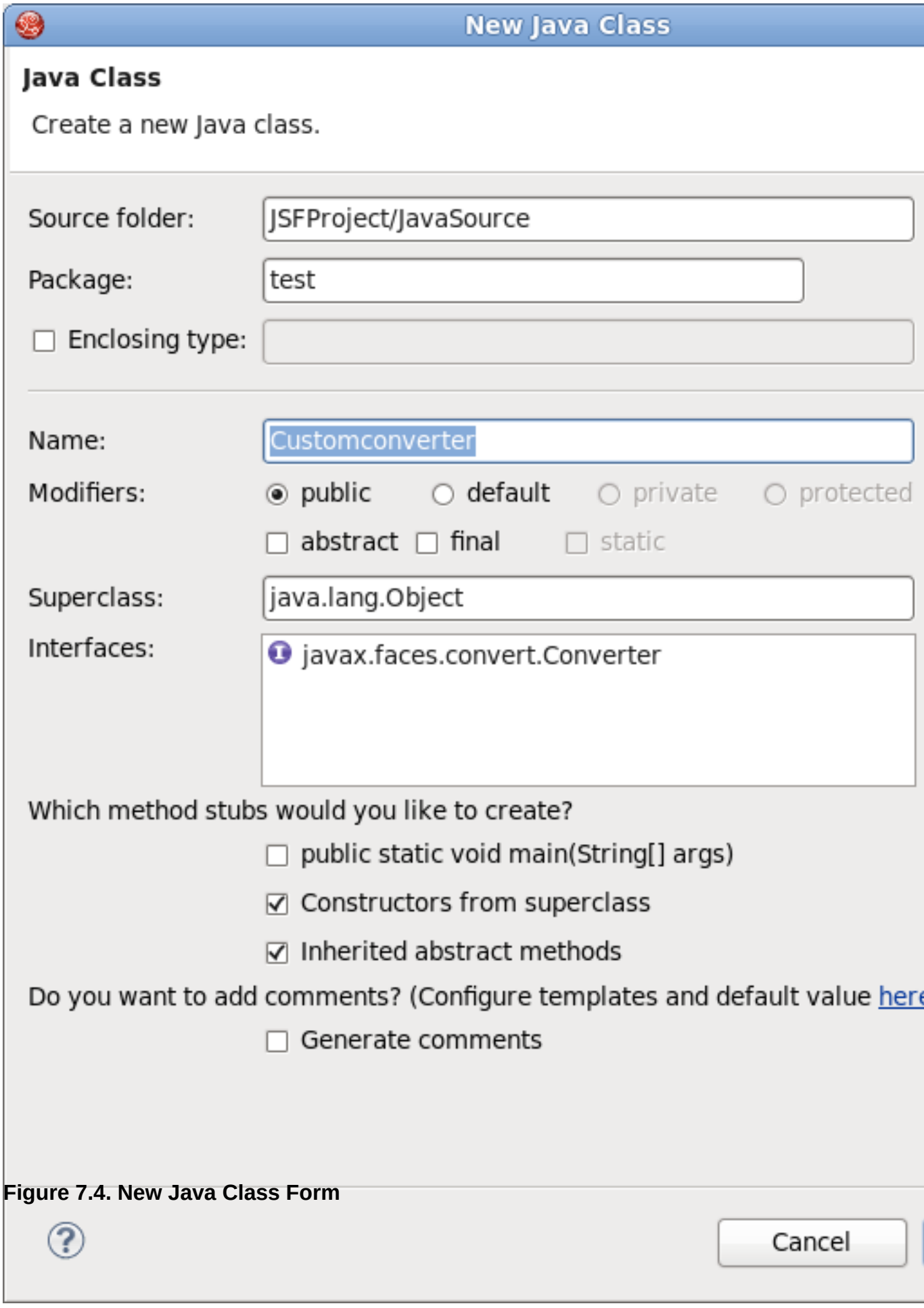

Figure 7.4. New Java Class Form

- To open a converter class click again on the **Converter-Class** link in the Converter section.

*faces-config.xml | carBean.java | Customconverter.java

```
1  package test;
2
3  import javax.faces.component.UIComponent;
6
7  public class Customconverter implements Converter {
8
9      public Customconverter() {
10         // TODO Auto-generated constructor stub
11     }
12
13     @Override
14     public Object getAsObject(FacesContext arg0, UICompone
15         // TODO Auto-generated method stub
16         return null;
17     }
18
19     @Override
20     public String getAsString(FacesContext arg0, UICompone
21         // TODO Auto-generated method stub
22         return null;
23     }
24
25 }
26
```

Figure 7.5. Converter Class

Now you are able to add a business logic of converter in the Java editor.

7.2. Create and Register a Custom Validator

It is also quite easy to develop your own custom Validators. The required steps are similar to those shown previously:

- In the Project Explorer view open the `faces-config.xml` and select the **Tree** tab.

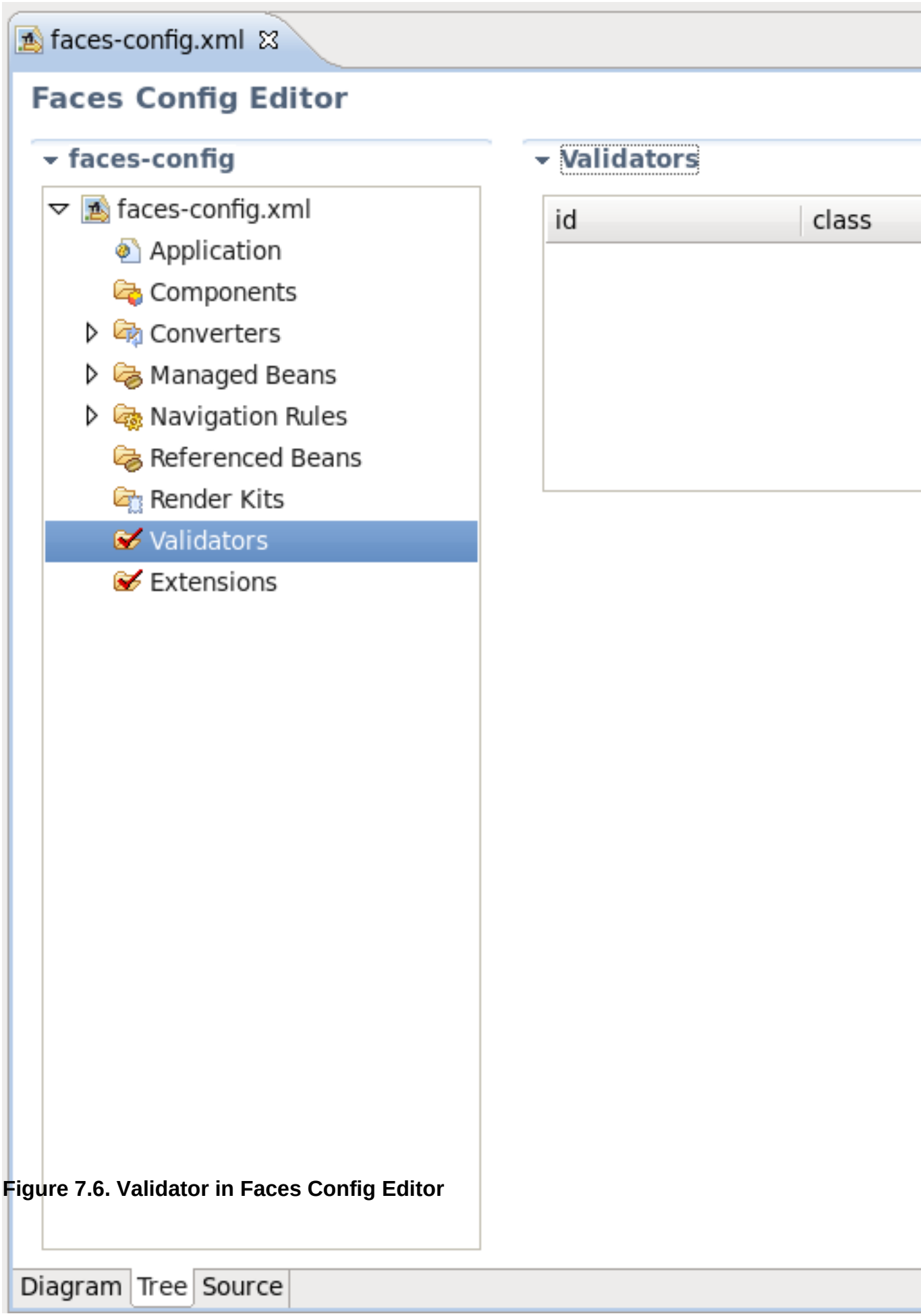

Figure 7.6. Validator in Faces Config Editor

- Select the **Validators** option and click the **Add**button.
- Type the name of your validator in the **Validator-id** field and name of the class for validators. After clicking the **Finish** button your custom validator is registered under the entered name.

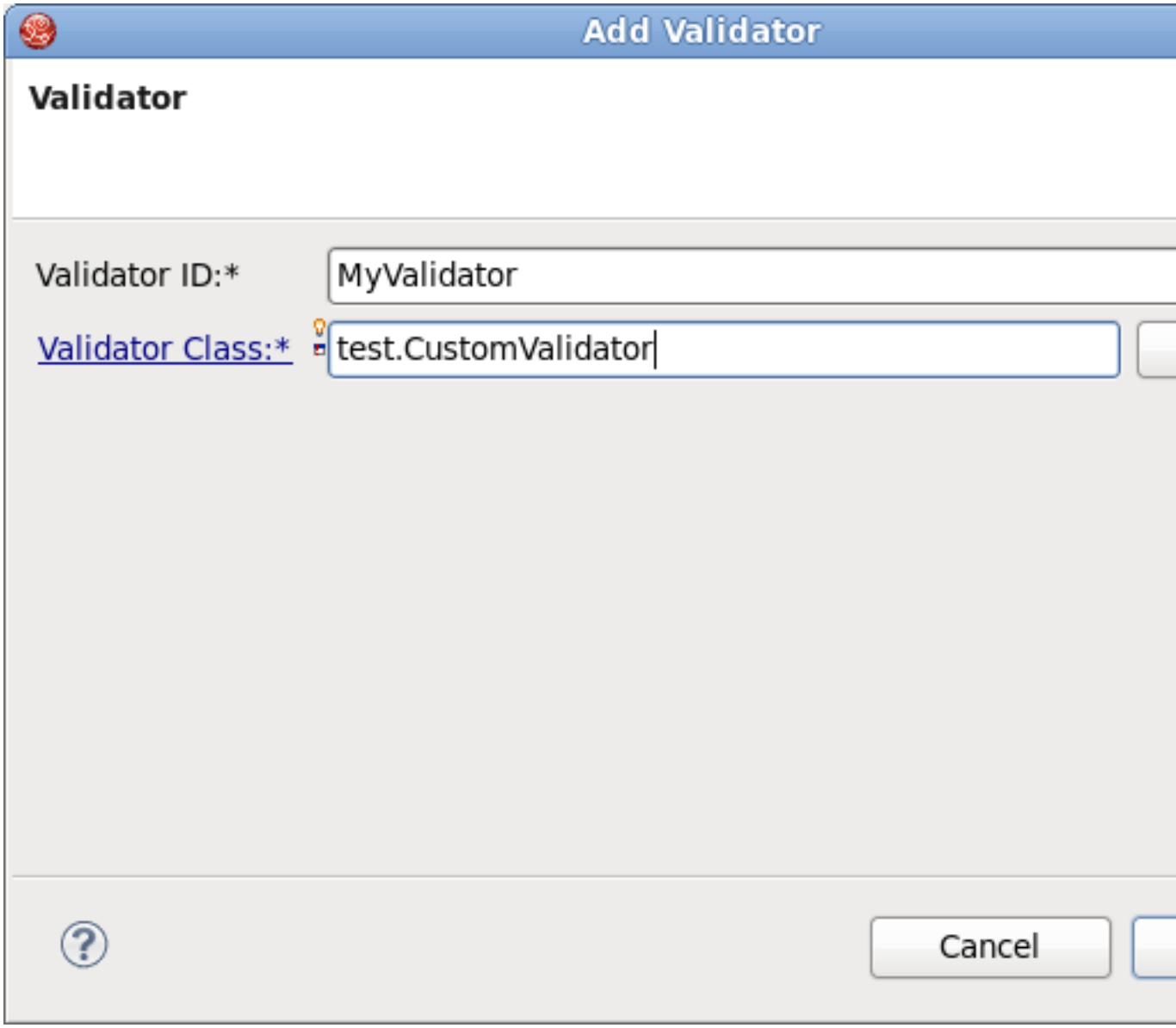

Figure 7.7. Adding Validator

Now you can create the "validator" class.

- In the Validator section you can see your **Validator-id** and **Validator-class**. To generate the source code click on **Validator-class**.

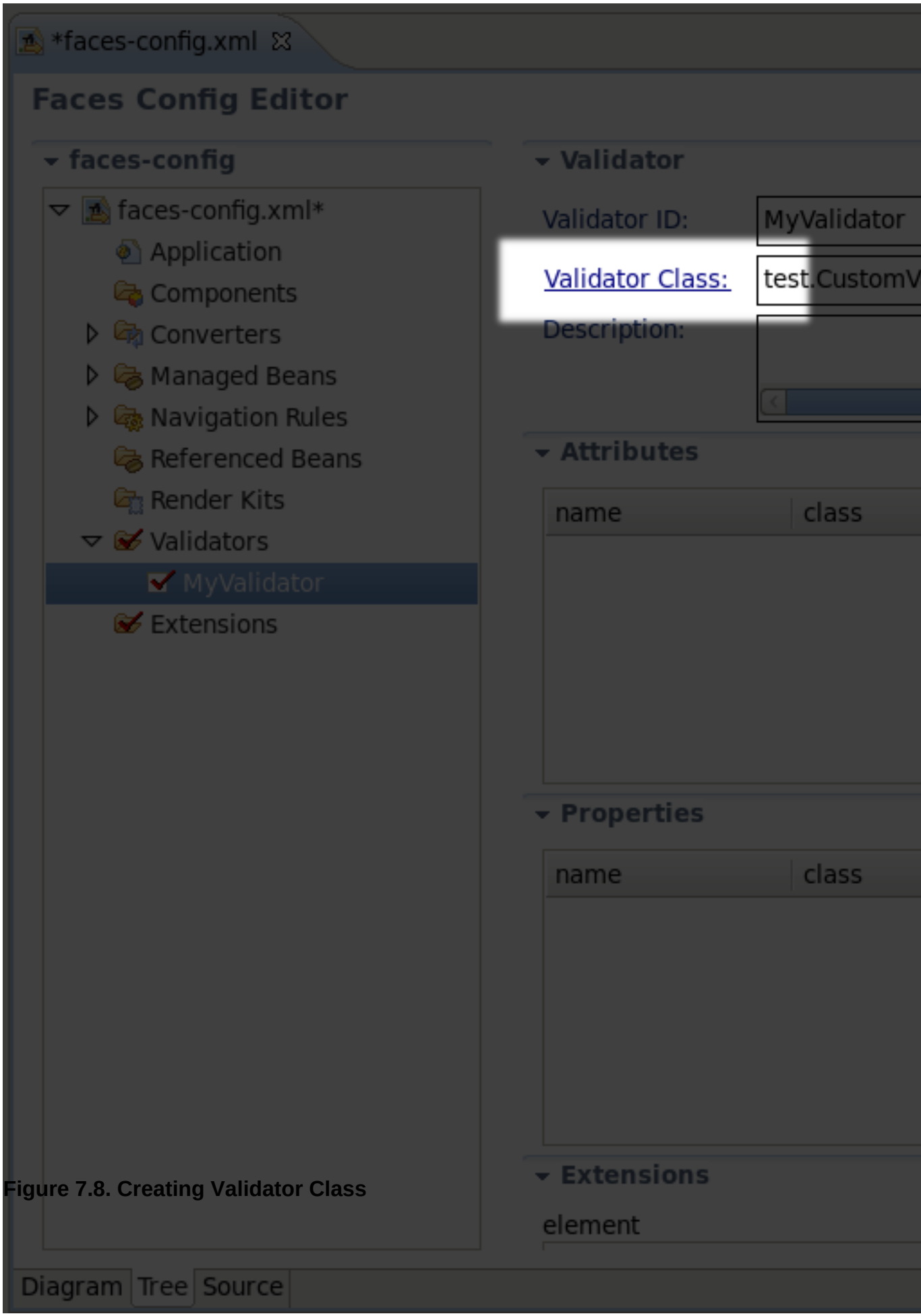

Figure 7.8. Creating Validator Class

- Java class will be created automatically. Leave everything without changes and click the **Finish**.

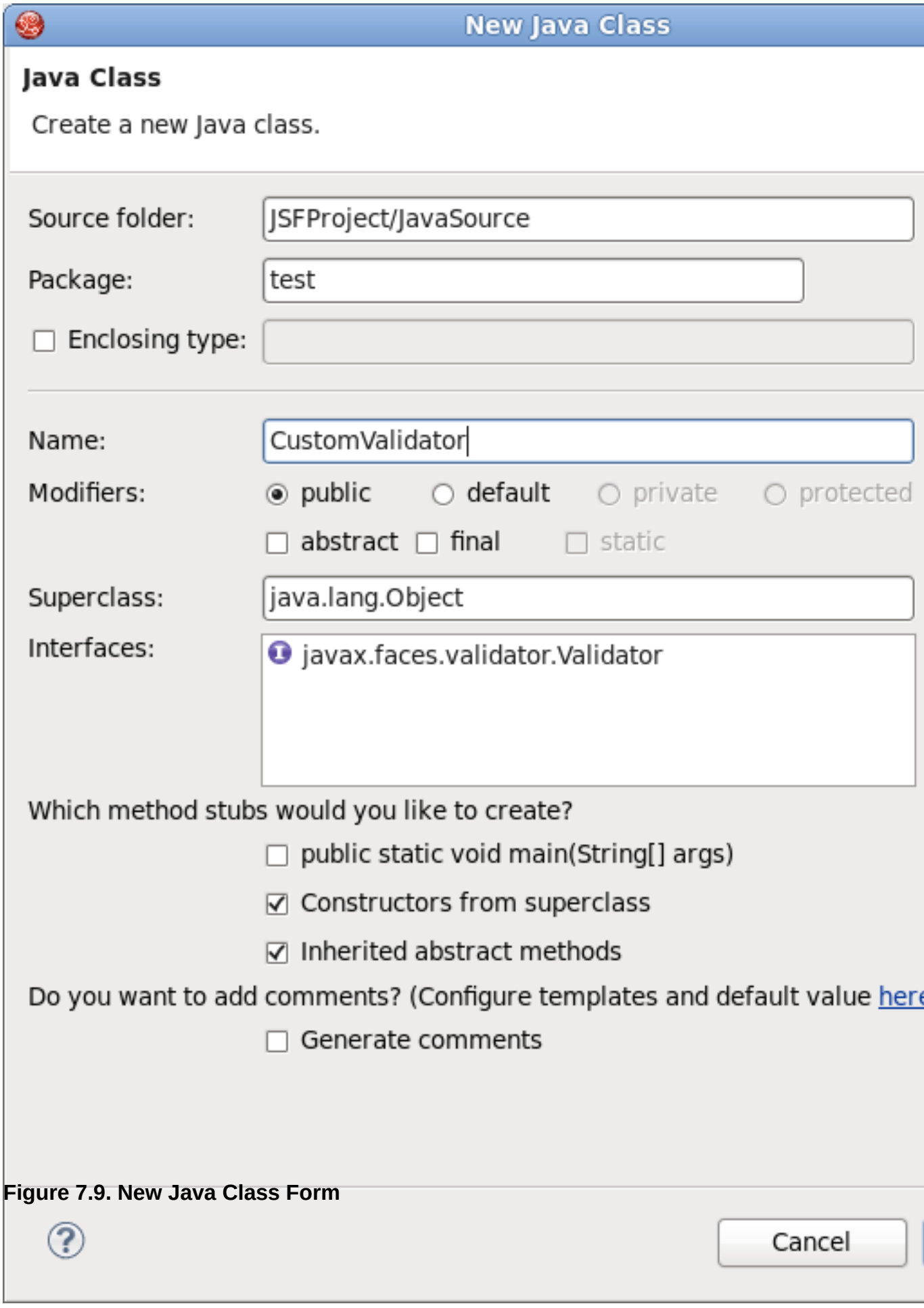

Figure 7.9. New Java Class Form

- To open the validator class click on the **Validator-Class** link in the Validator section. Now you are able to write a business logic of validator in the Java editor.

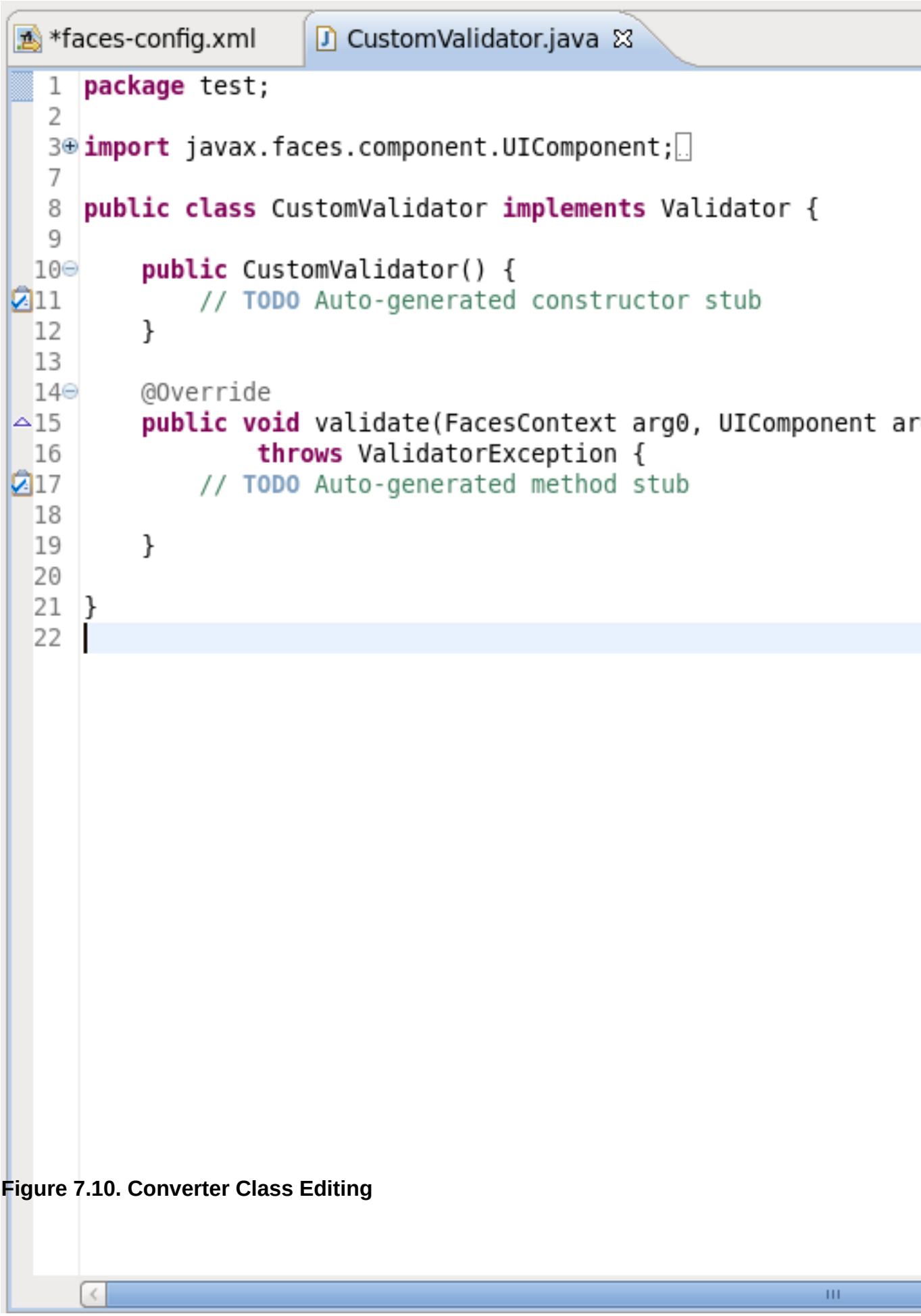

Figure 7.10. Converter Class Editing

7.3. Create and Register Referenced Beans

The creation of Referenced Beans is similar to the creation of Custom Validators. The steps below show you the steps required to create Referenced Beans.

- In the Project Explorer view open the `faces-config.xml` and select the **Tree** tab.

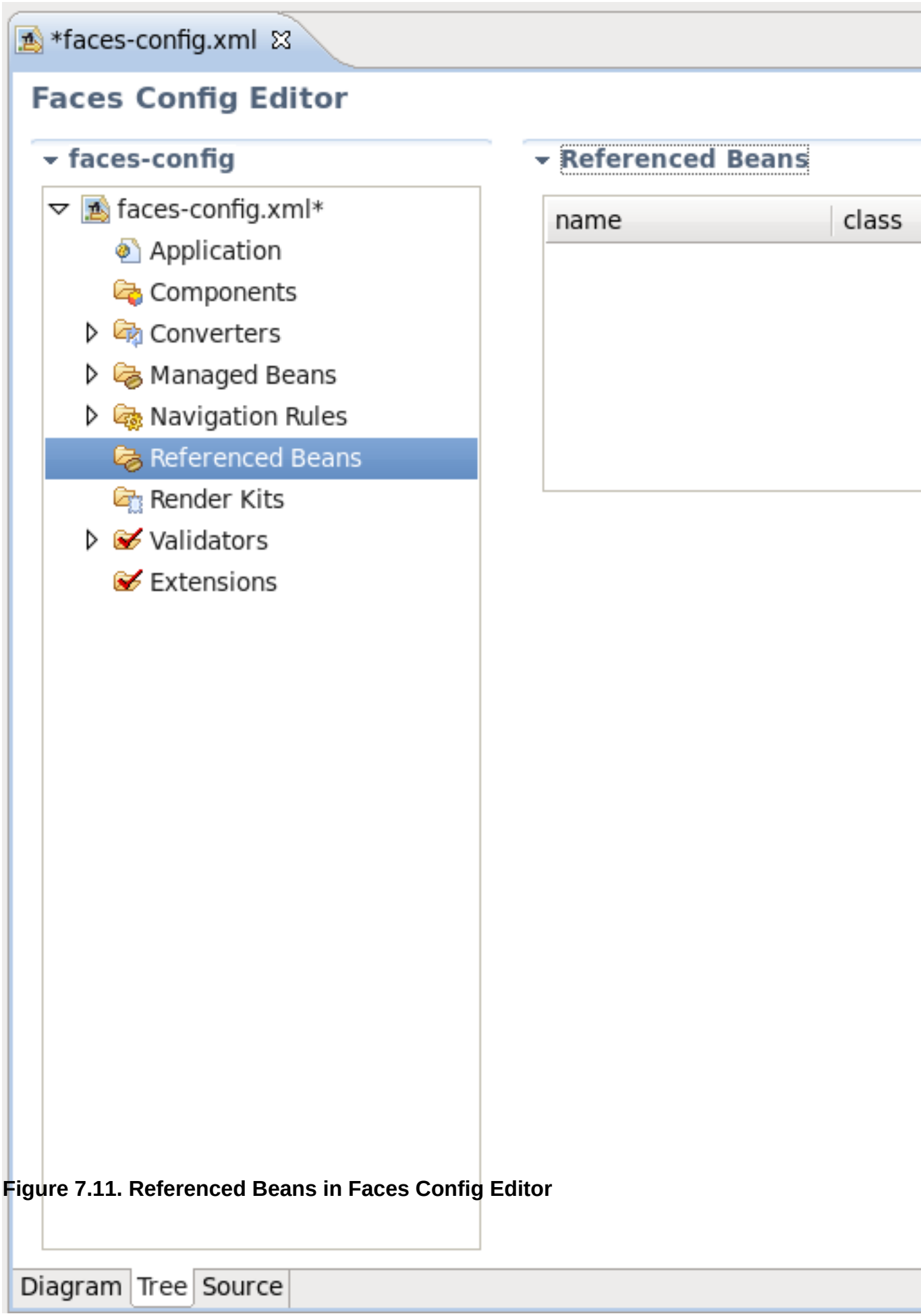

Figure 7.11. Referenced Beans in Faces Config Editor

- Select the **Referenced Beans** option and click on the **Add** button.
- Type in the name of your Referenced Bean and type in or select the **Referenced-Bean-Class** value by clicking the **Browse** button.

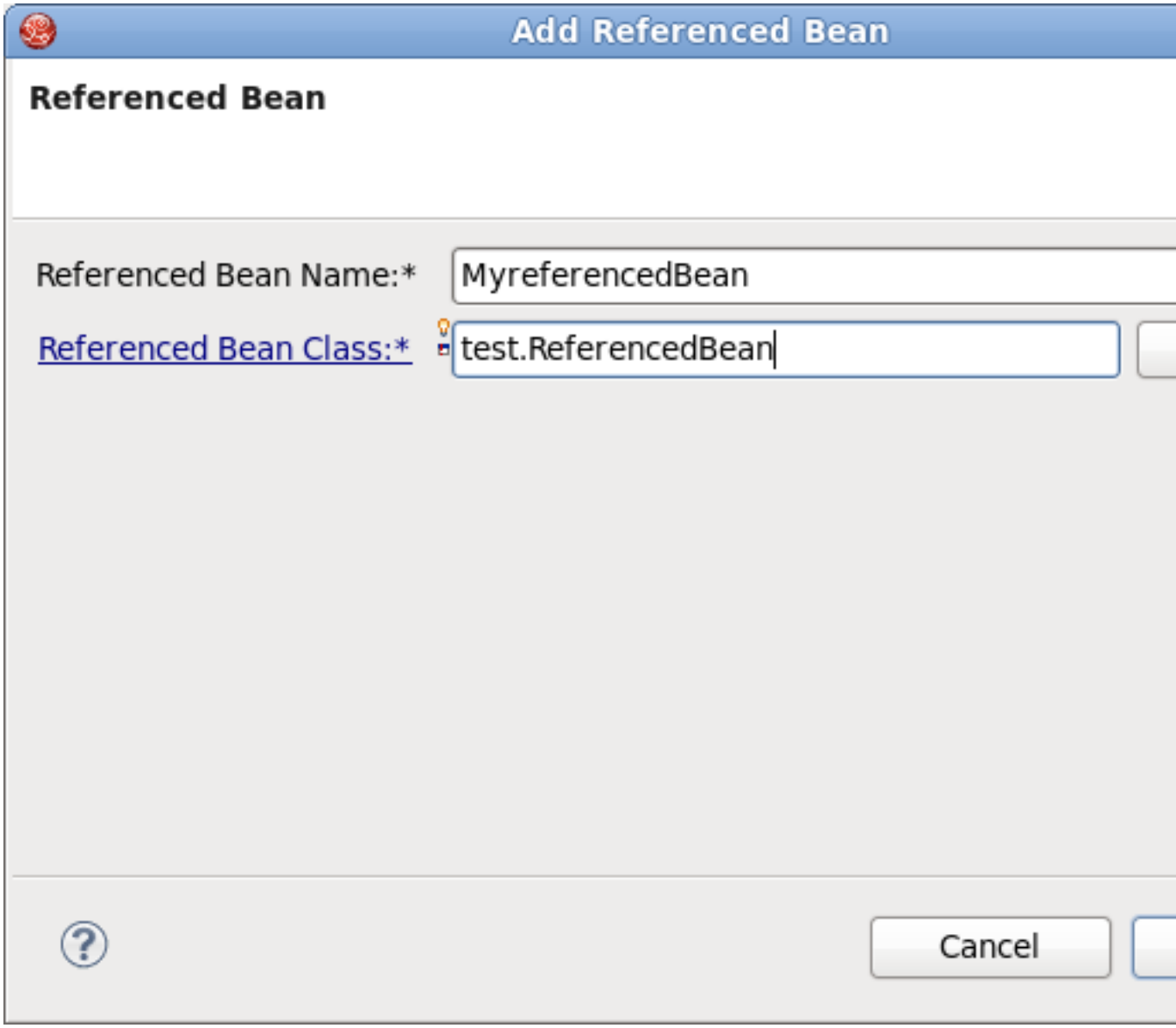

Figure 7.12. Add Referenced Bean

- In the Referenced Bean section you should see your **Referenced-Bean-Name** and **Referenced-Bean-Class**. Click on the link to open the Java creation wizard.

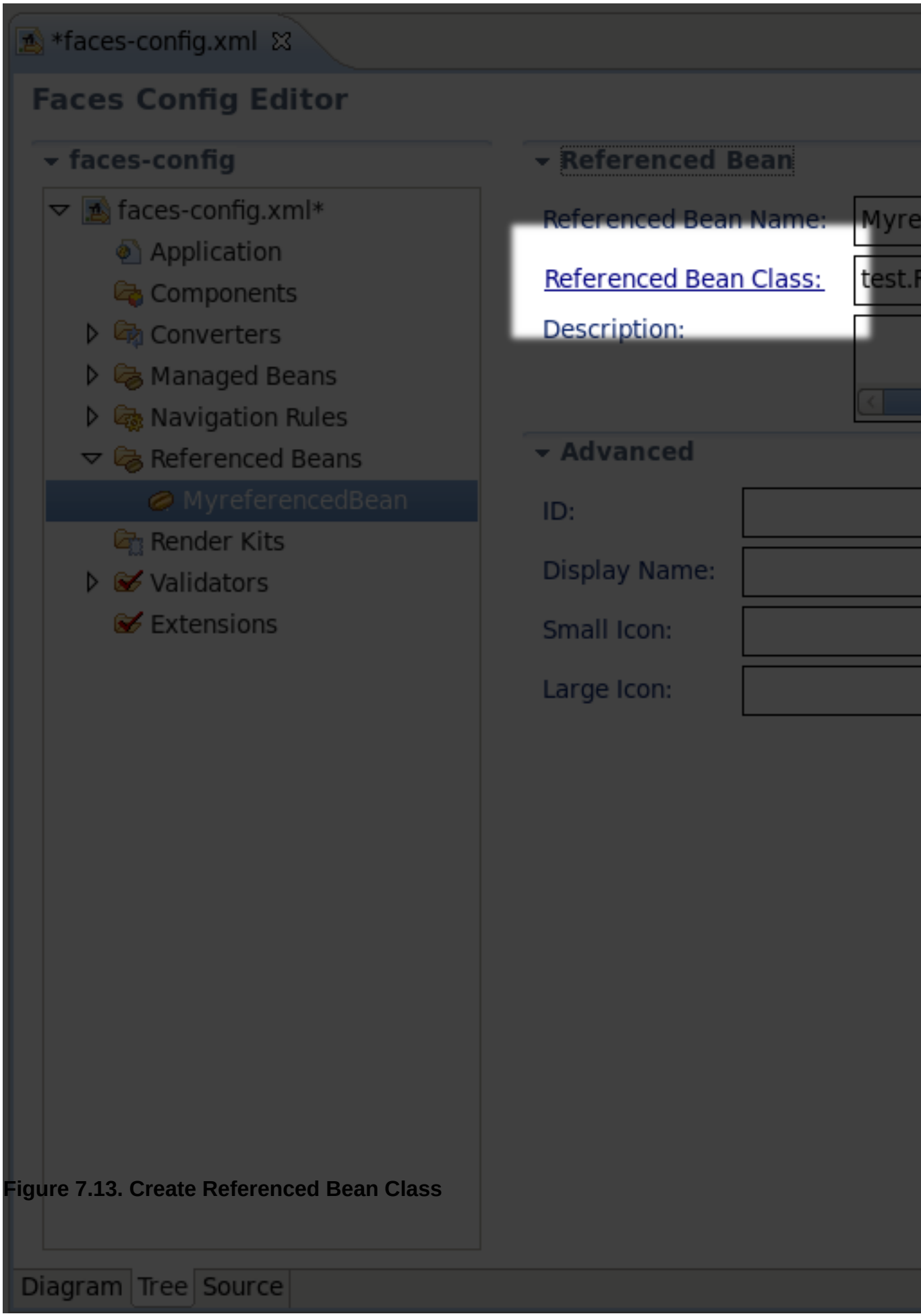

Figure 7.13. Create Referenced Bean Class

- The Java class will be created automatically. Leave everything with their default values and click the **Finish** button.

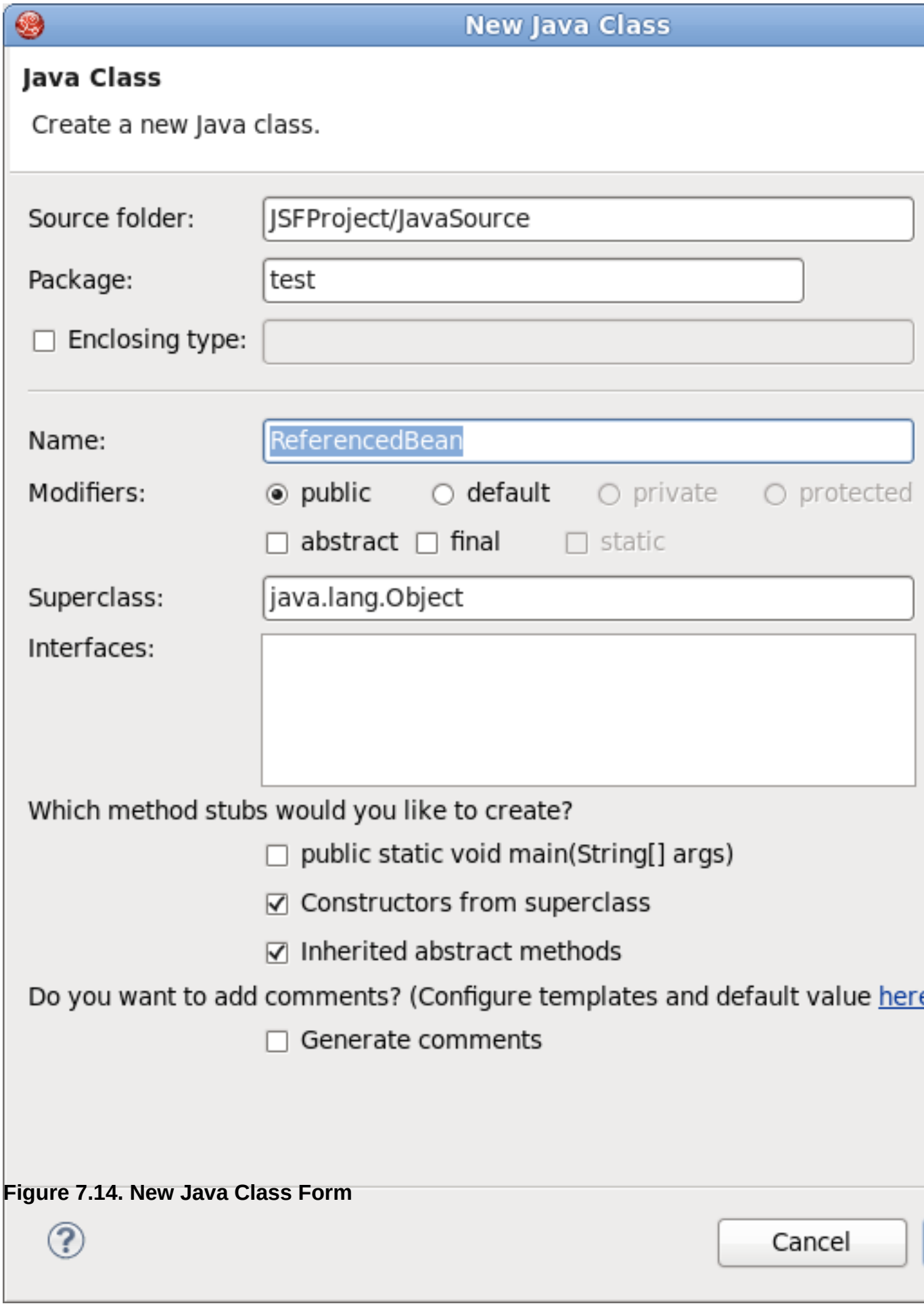

Figure 7.14. New Java Class Form

- To open a Referenced Bean class click the **Referenced-Bean-Class** in the Referenced Bean section. Now you are able to write business logic of Referenced Bean in the Java editor.

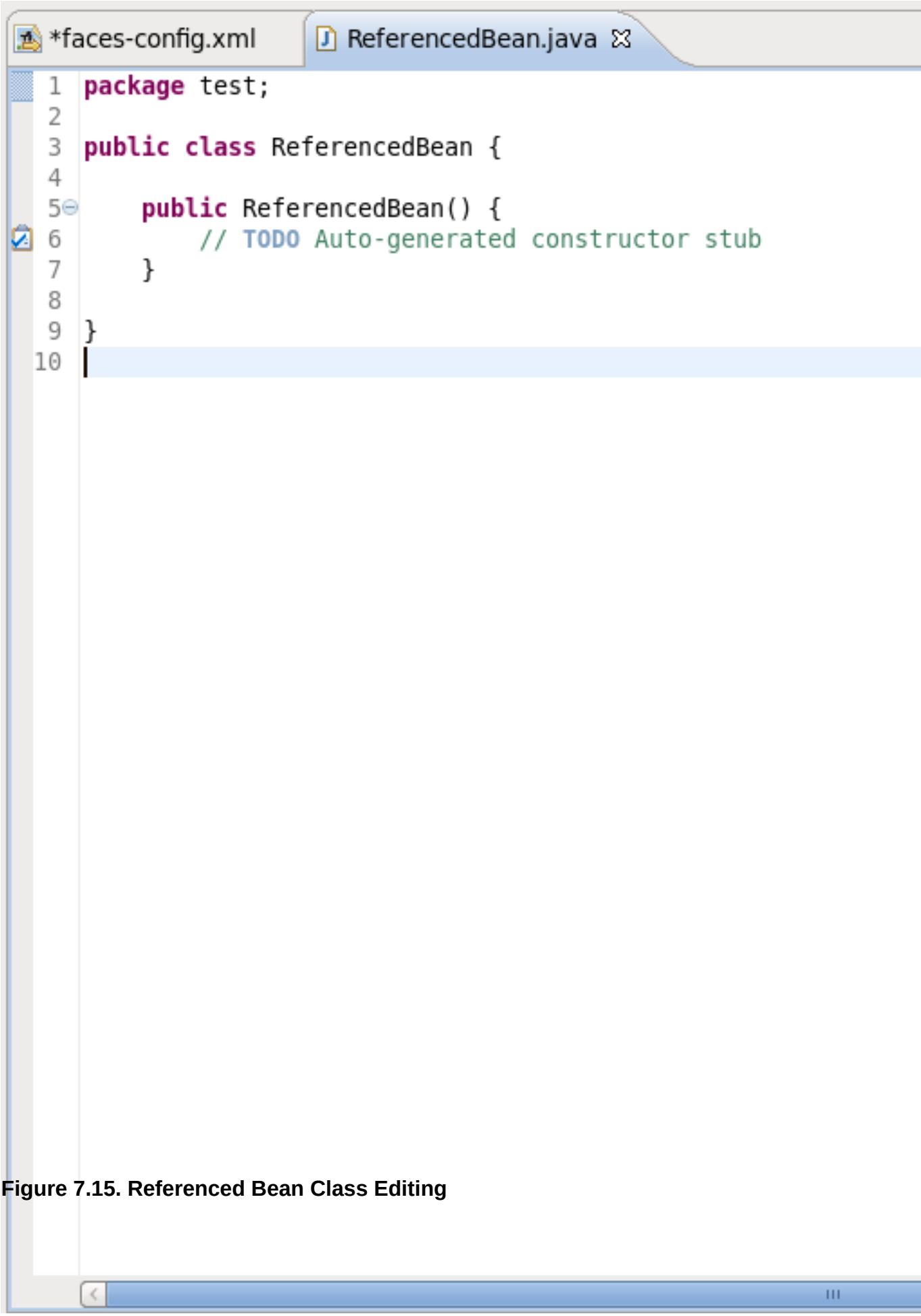

Figure 7.15. Referenced Bean Class Editing

JSF Project Verification

In this chapter we'll discuss a possible verification that you can take advantage of.

Many different rules are checked for a JSF project that can be configured by selecting **Window** → **Preferences** from the menu bar, selecting **JBoss Tools** → **Web** → **Verification** from the Preferences dialog box and then expanding the JSF Rules node.

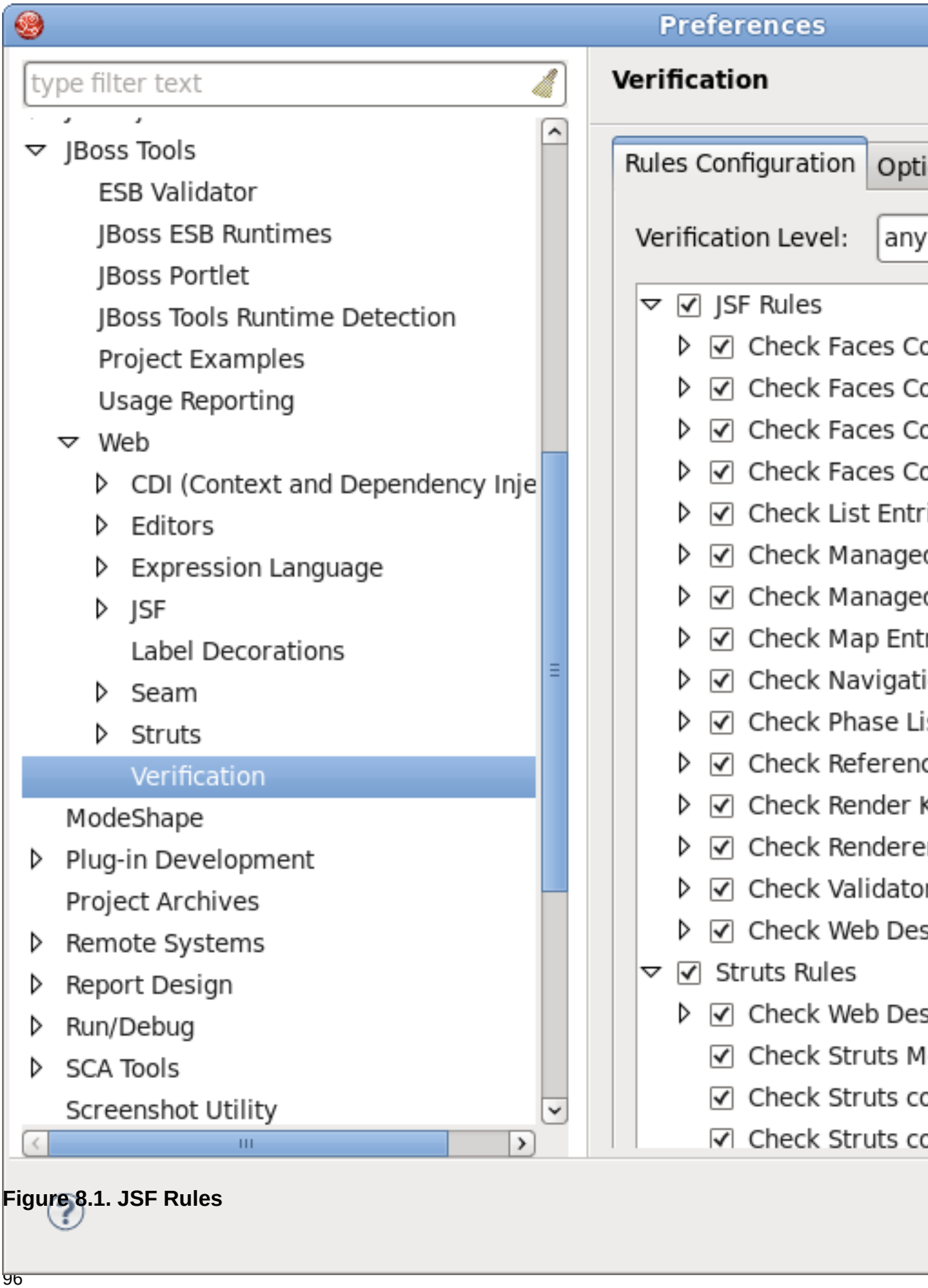

Figure 8.1. JSF Rules

Suppose you are working in the Source viewer for a JSF configuration file as shown below:

faces-config.xml

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <faces-config version="1.2" xmlns="http://java.sun.com/xml/
3   xmlns:xi="http://www.w3.org/2001/XInclude"
4   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi
5   <converter>
6    <converter-id>MyConverter</converter-id>
7    <converter-class>test.Customconverter</converter-class>
8   </converter>
9   <managed-bean>
10   <managed-bean-name>person</managed-bean-name>
11   <managed-bean-class>demo.Person</managed-bean-class>
12   <managed-bean-scope>request</managed-bean-scope>
13   <managed-property>
14    <property-name>name</property-name>
15    <value/>
16   </managed-property>
17  </managed-bean>
18  <managed-bean>
19   <managed-bean-name>carBean</managed-bean-name>
20   <managed-bean-class>example.carBean</managed-bean-class>
21   <managed-bean-scope>request</managed-bean-scope>
22   <managed-property>
23    <property-name>carName</property-name>
24    <property-class>java.lang.String</property-class>
25    <value/>
26   </managed-property>
```

Diagram | Tree | Source

Figure 8.2. Faces-config.xml File

While typing a class name, you might make a minor typo (like *"demo.Person9"* instead of *"demo.Person"*). After saving the file, verification checks to make sure everything is correct and finds the error below:

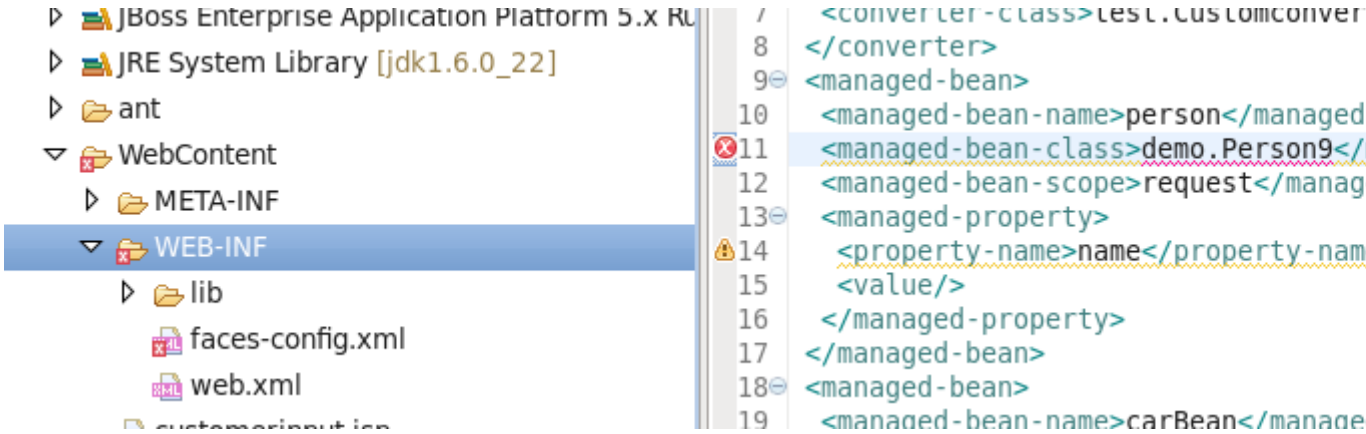

Figure 8.3. Error in Source View

Notice that the Package Explorer View shows a marked folder and a marked file where the error is.

You can place the cursor over the line with an error message and get a detailed error message:

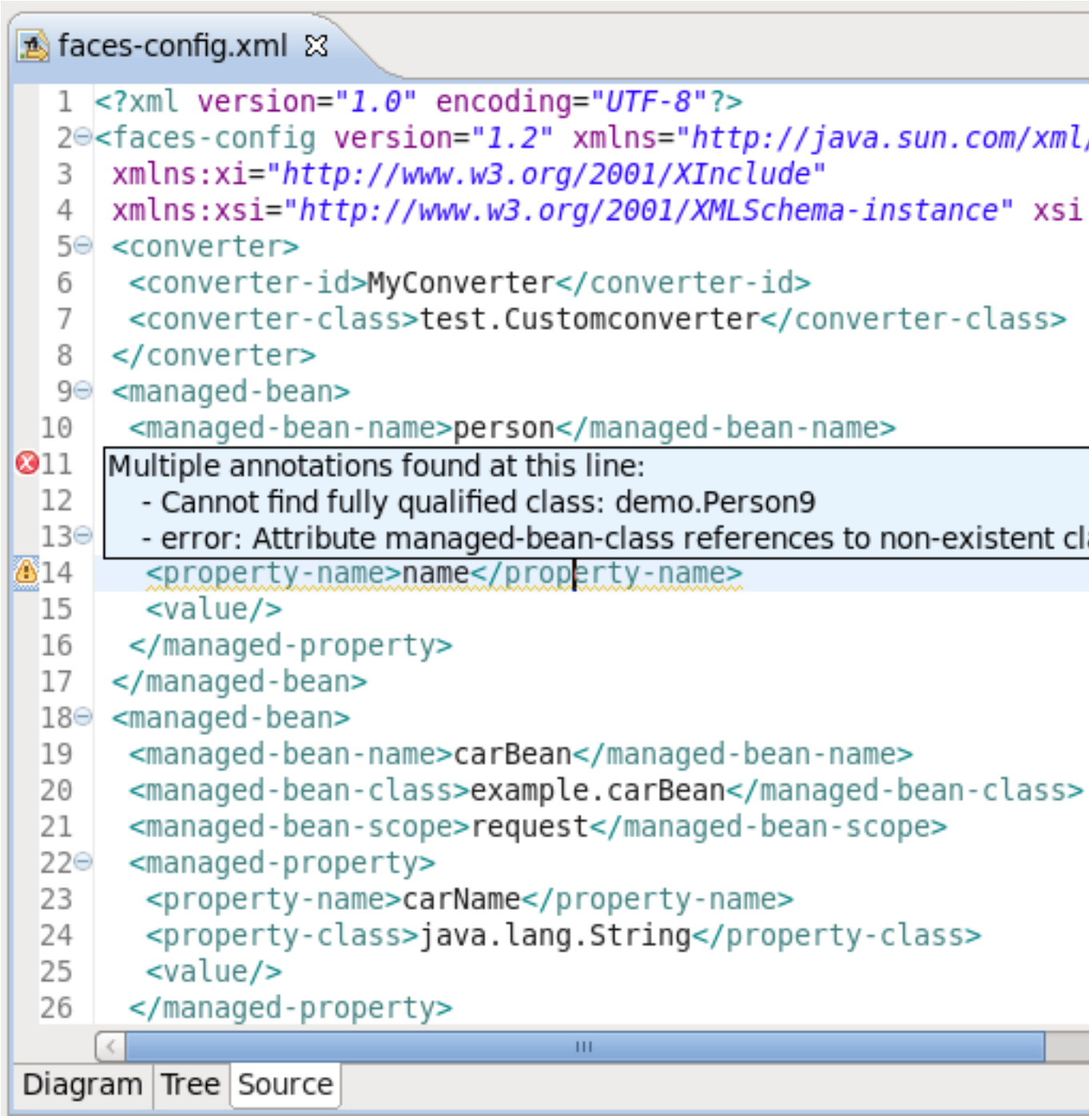

Figure 8.4. Error Message

Verification also checks navigation rules:

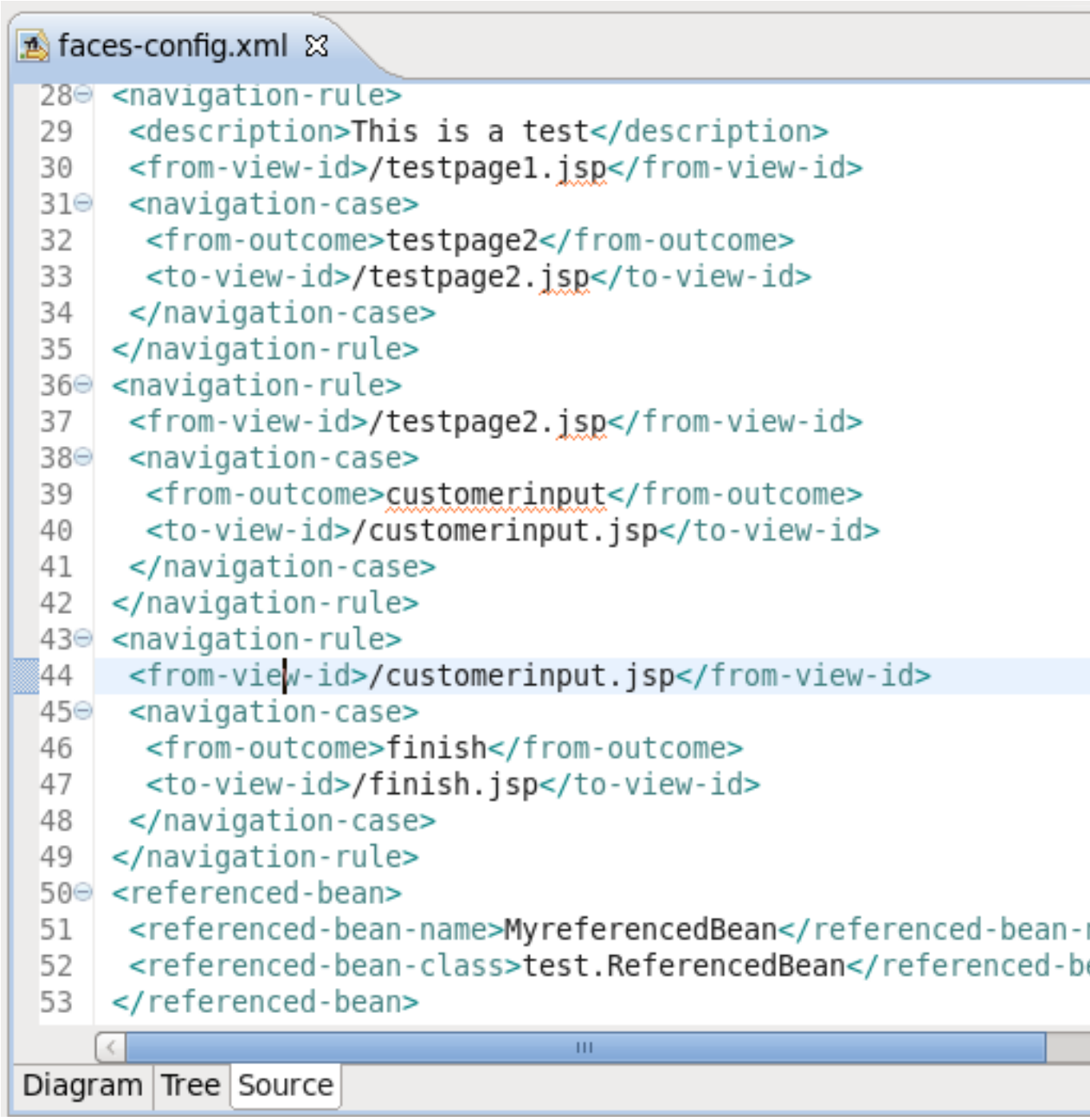

Figure 8.5. Checking Navigation Rules

If you provide a page name that does not exist, verification will let you know about that:

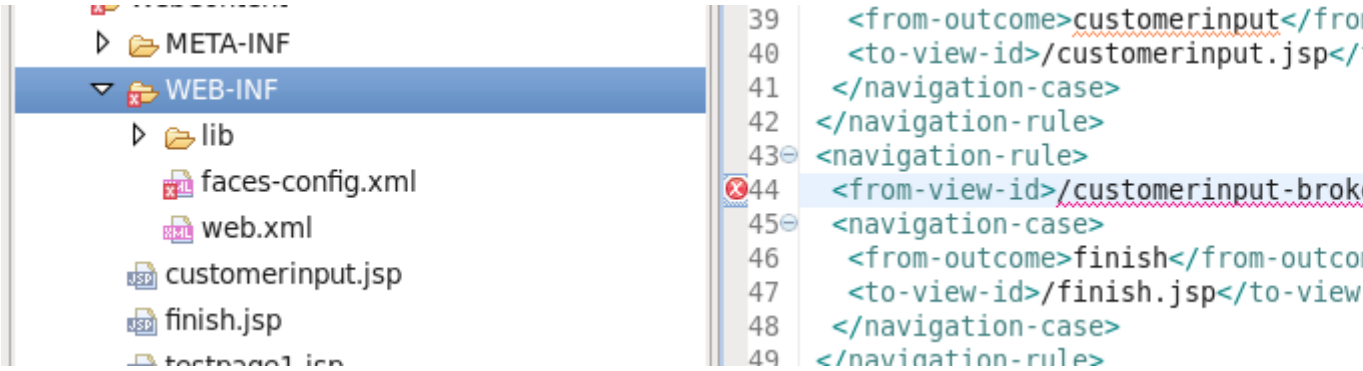

Figure 8.6. Page Name Verification

You can always call up verification explicitly by right-clicking any element in the tree and selecting Verify from the context menu. This works from both the Tree and Diagram viewers for the JSF configuration file editor. You can also invoke verification from the Web Projects view. Below we are checking all of the elements in the configuration file.

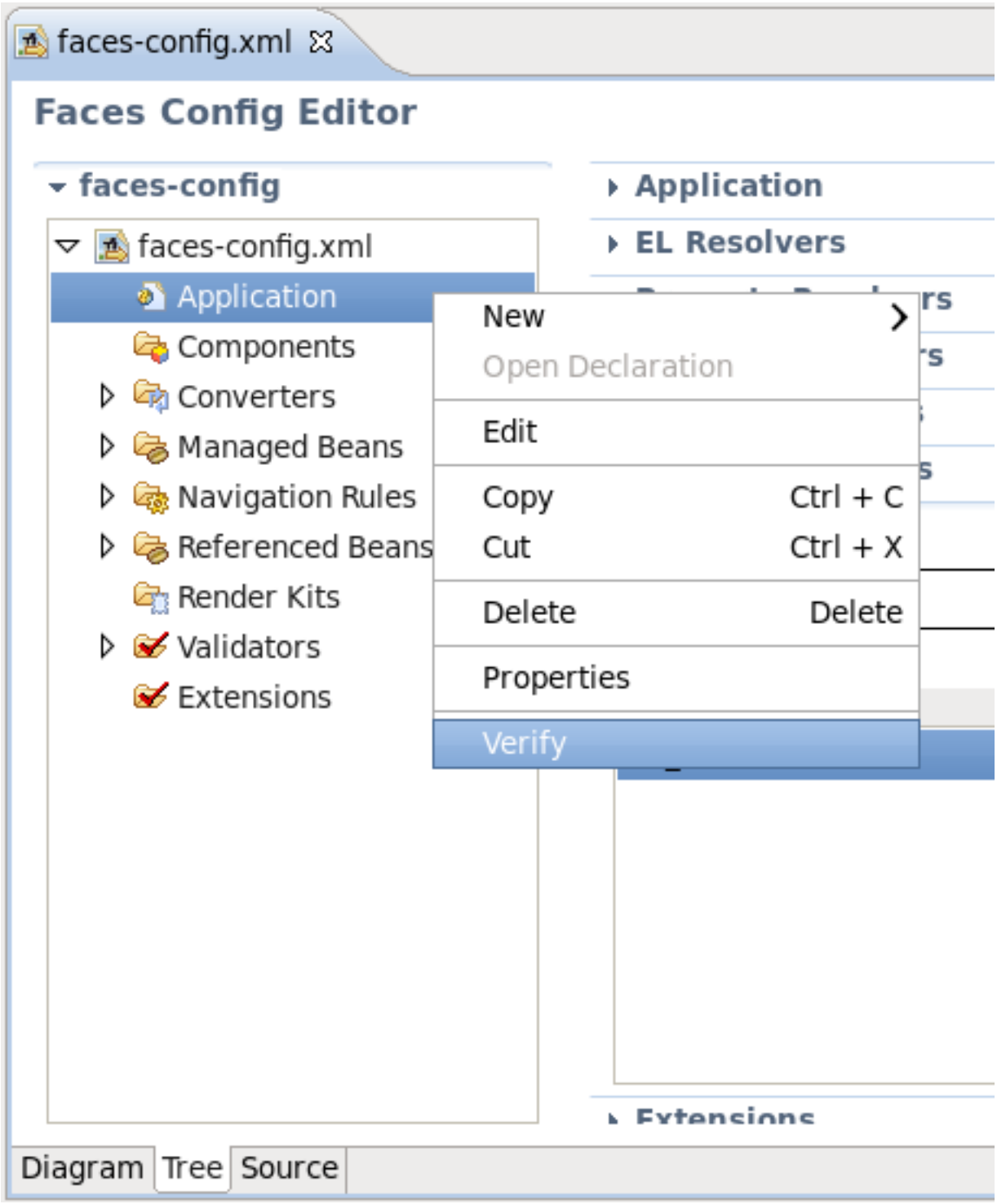

Figure 8.7. Verify Command

In summary, this document highlights all the JSF-specific features of JBoss Tools meant for enhancing the development of rich Web applications based on JSF technology. The reference introduces you to wizards for creating and importing JSF projects, JSF Configuration File editor features, functionality for enabling JSF capabilities and etc.

If you have questions or good suggestions, please refer to *JBoss Tools Forum* [http://www.jboss.com/index.html?module=bb&op=viewforum&f=201].