JSF Tools Reference Guide

Version: 3.1.0.CR1

Introduction

JSF Tools are especially designed for supporting JSF and JSF-related technologies. JSF Tools provide extensible and exemplary 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.

In this guide we provide you with the information on JSF tooling which allows you to develop JSF applications much faster and with far fewer errors so sparing your time.

1.1. Key Features of JSF Tools

Here, we provide you with a key functionality which is integrated in JSF tooling.

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.	*jsf support*
Flexible and customizable project template management	Jump-start development with out-of-the-box templates or easily customized templates for re-use.	*projects*
Support for JSF Configuration File	Working on file using three modes: diagram, tree and source. Synchronization between the modes and full control over the code. Easy moving around the diagram using the Diagram Navigator.	*graphical editor for jsf*
Support for Managed Beans	Adding new managed beans, generating code for attributes, properties and getter/setter methods.	*managed beans*
Support for Custom Converters and Validators	Fast creating of custom converters and validators with tree view of faces-config.xml file.	*converters and validators*
Verification and Validation	All occuring errors will be immediately reported by verification feature, no matter in what view you are working. Constant validation and errors checking allows to catch many of the errors during development process that significantly reduces development time.	*verification and validation*

1.2. Other relevant resources on the topic

All JBoss Developer Studio/JBoss Tools release documentation you can find at*http://docs.jboss.org/tools* [http://docs.jboss.org/tools/] in the corresponding release directory.

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

JavaServer Faces Support

We don't lock you into any one JavaServer Faces implementation. You can always select the one which is necessary for you while *creating a new JSF project*, *adding JSF capability* to any existing Eclipse project or *importing existing JSF projects* as well.

At this point the special wizard will prompt you to specify a proper JSF environment. It may be JSF 1.1.02 RI or JSF 1.2 which integrates a number of new features and changes. The wizard also lets you select JSF implementation with a component orientation such as JSF 1.2 with Facelets or MyFaces 1.1.4.

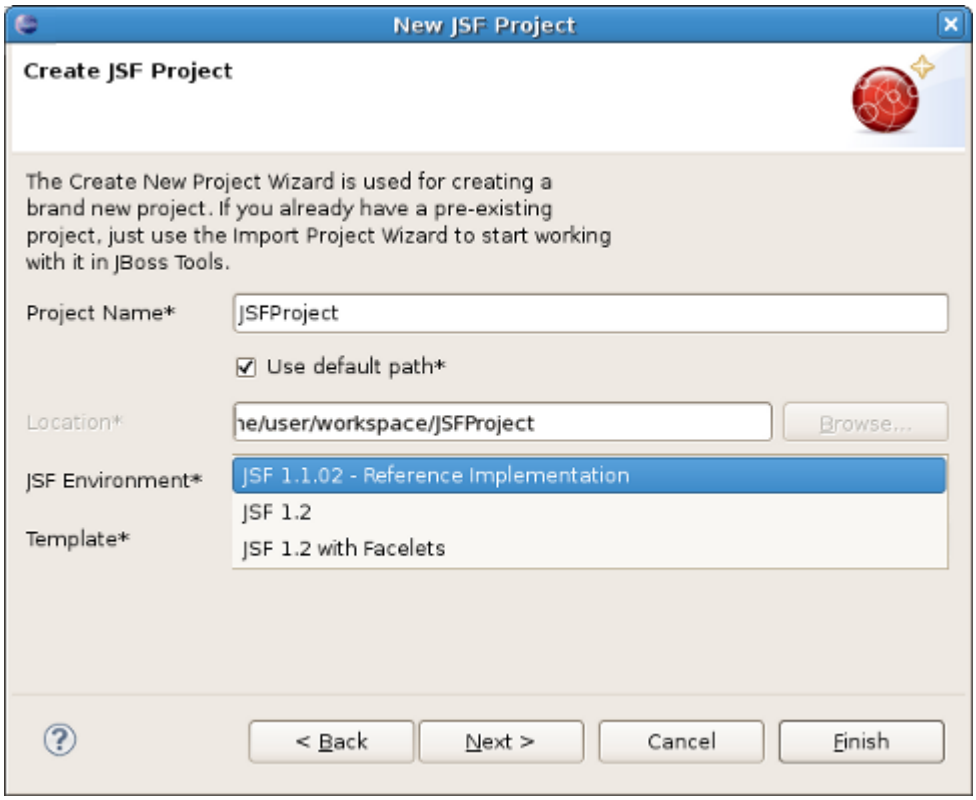

Figure 2.1. Choosing JSF Environment

After specifying a proper JSF environment all the required libraries for the selected version will be added to your project.

2.1. Facelets Support

In this section we will focus more on all concepts that are integrated for working with Facelets.

The Facelets extends JavaServer Faces by providing a lightweight framework that radically simplifies the design of presentation pages for JSF. 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, just create a project with Facelets based on version 1.2 of the JSF Reference Implementation, i. e. select the *JSF 1.2 with Facelets* in the JSF Environment section of the New JSF Project wizard.

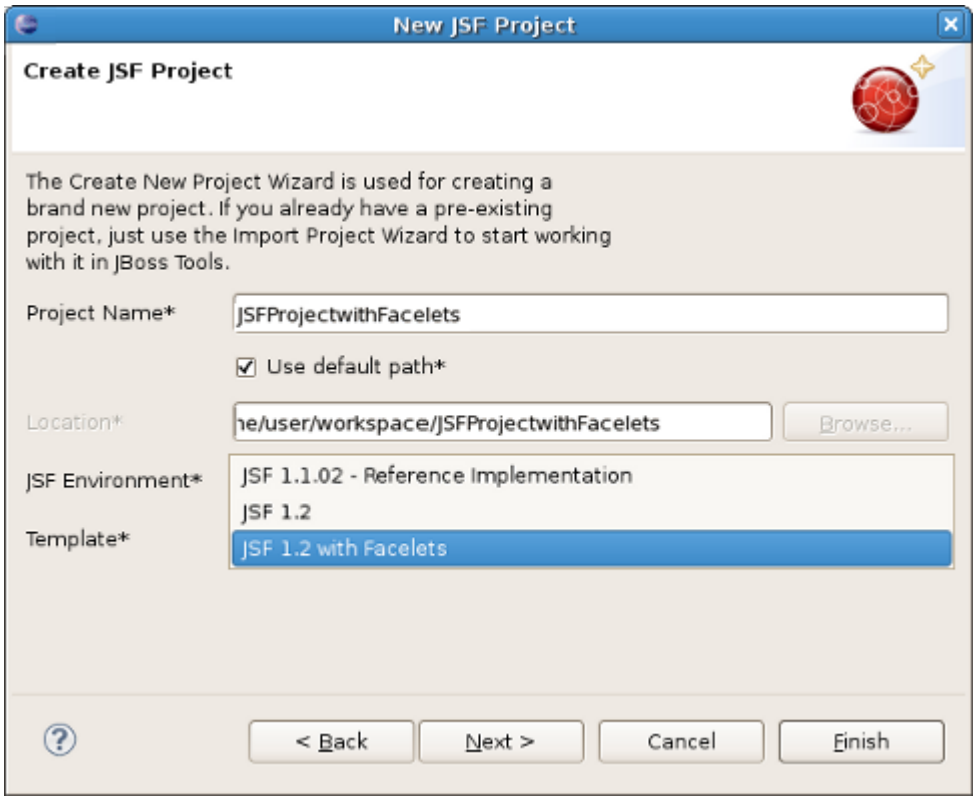

Figure 2.2. Choosing Facelets Environment

Once you've selected the environment, it's possible to specify the one of three available templates:

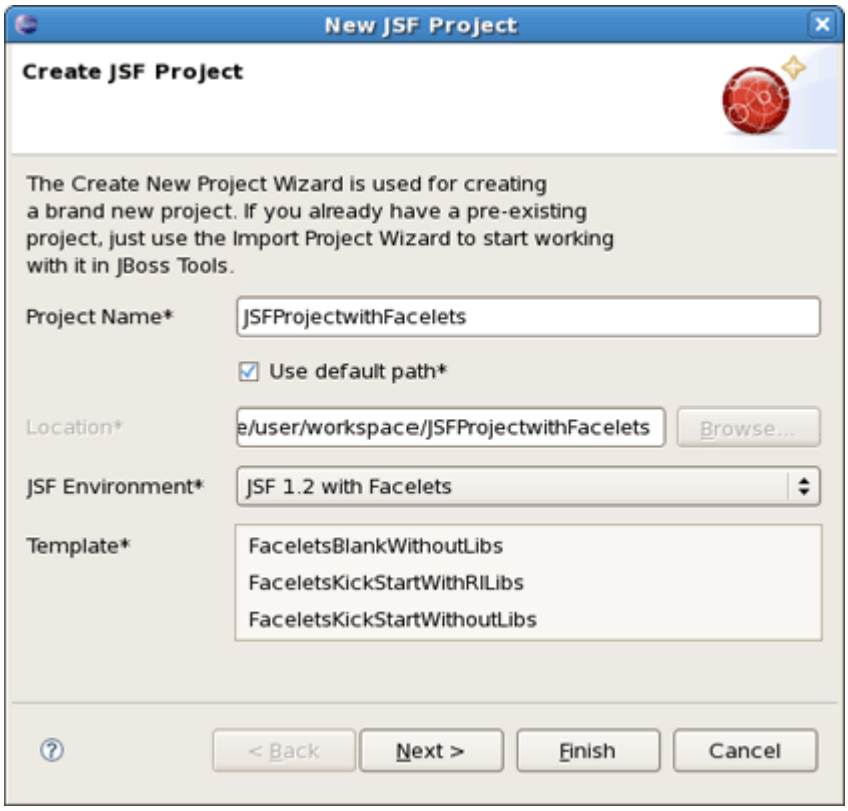

Figure 2.3. Choosing Facelets Template

The following table lists possible templates with Facelets for any JSF project and gives a proper description for each one.

Table 2.1. Facelets Templates

Template	Description
FaceletsBlankWithoutLibs	Some servers already provide jsf libs and you take risk of getting conflicting libraries 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* [../../jsf/html_single/index.html#palette] comes with the Facelets components ready to use. A useful tip appears when you hover the mouse cursor over the tag, the tip includes a detailed description of the tag component, the syntax and available attributes.

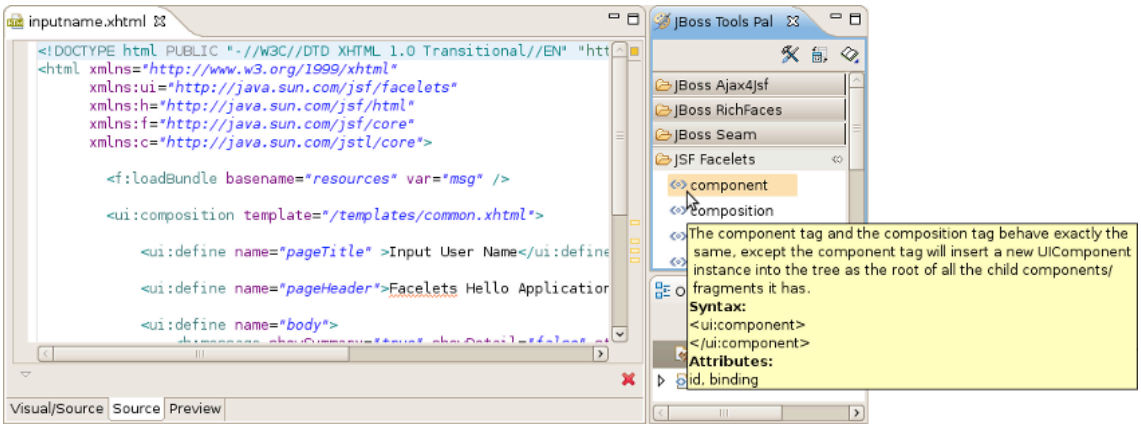

Figure 2.4. Facelets Components

2.1.3. Code assist for Facelets

One more feature which comes with Facelets support is code assist (Ctrl + Space). It is available for Facelets tags while editing *.xhtml* files.

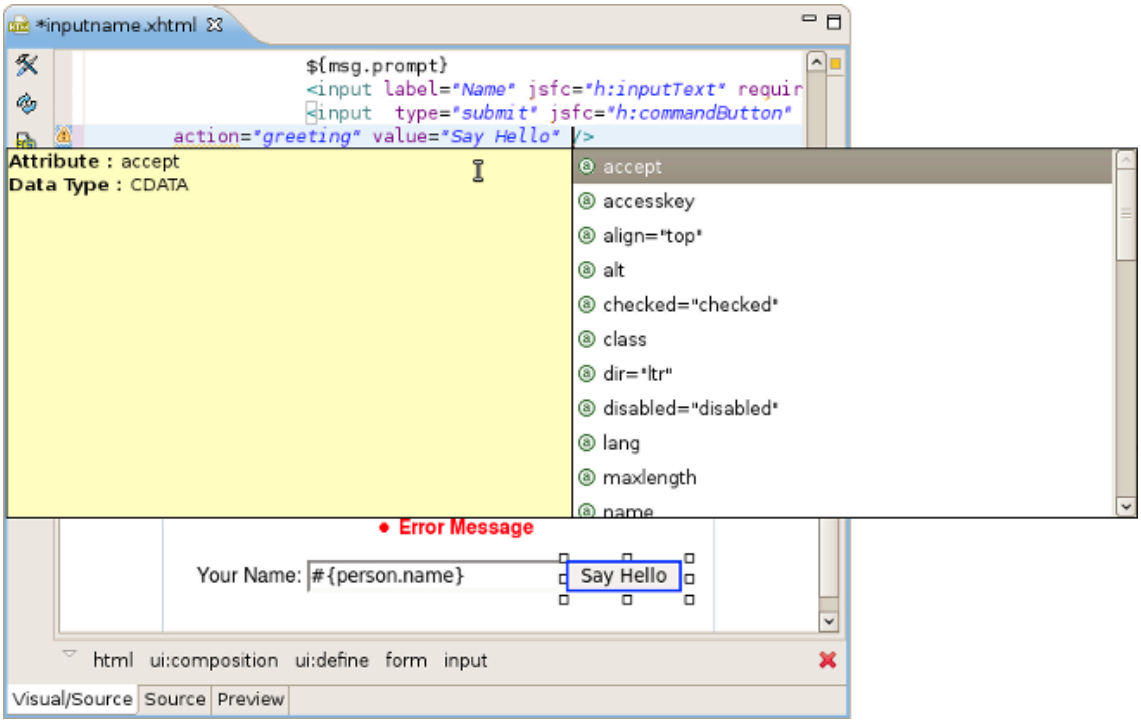

Figure 2.5. XHTML File Code Assist

What's more, code assist is also available for *"jsfc"* attribute in any HTML tag.

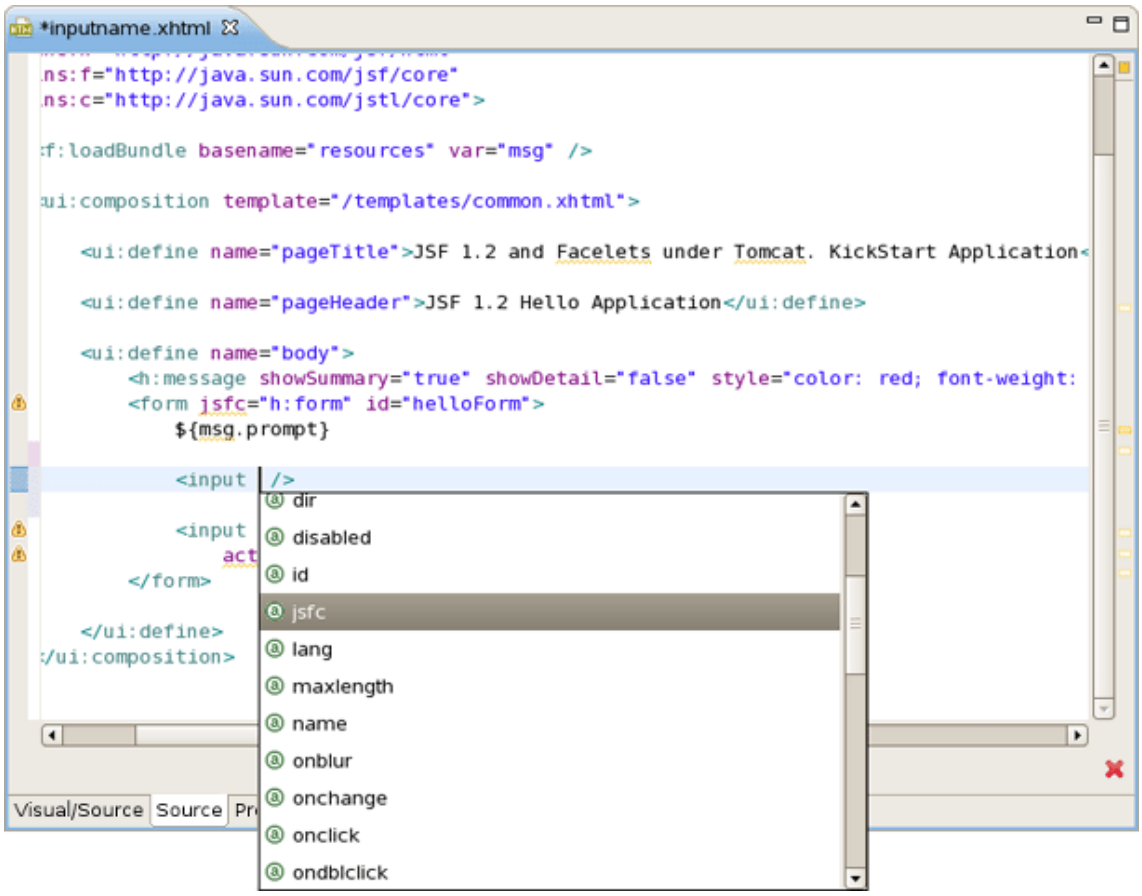

Figure 2.6. Code Assist for JSFC Attribute

After selecting *"jsfc"* you get the code assist for JSF components available on a page.

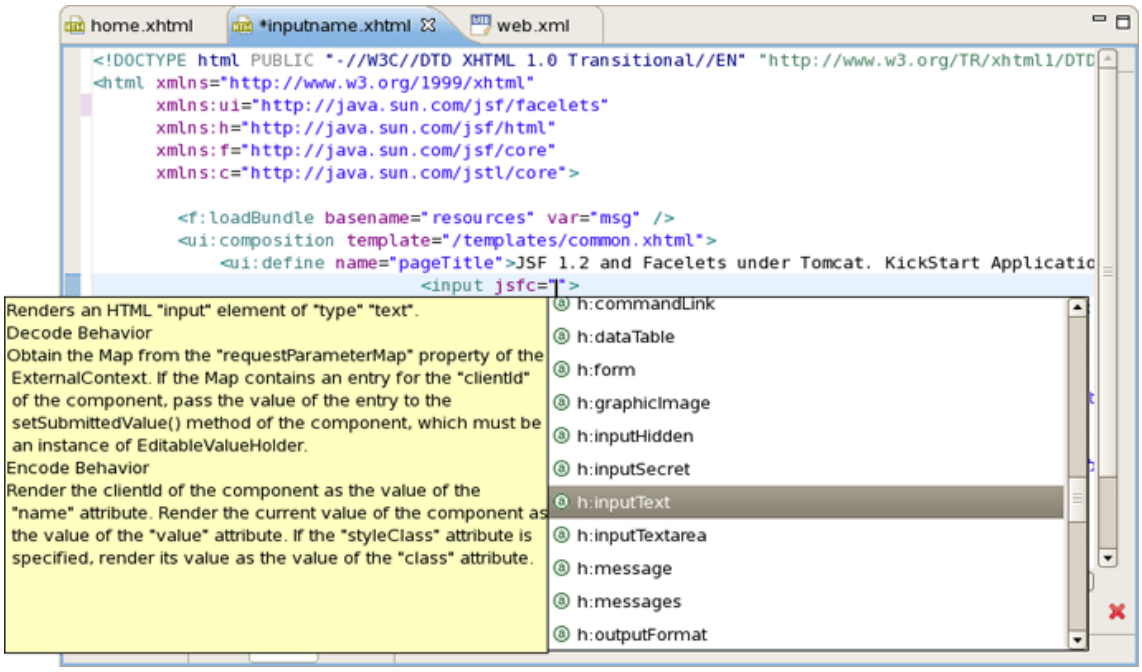

Figure 2.7. Code Assist for JSF Components

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

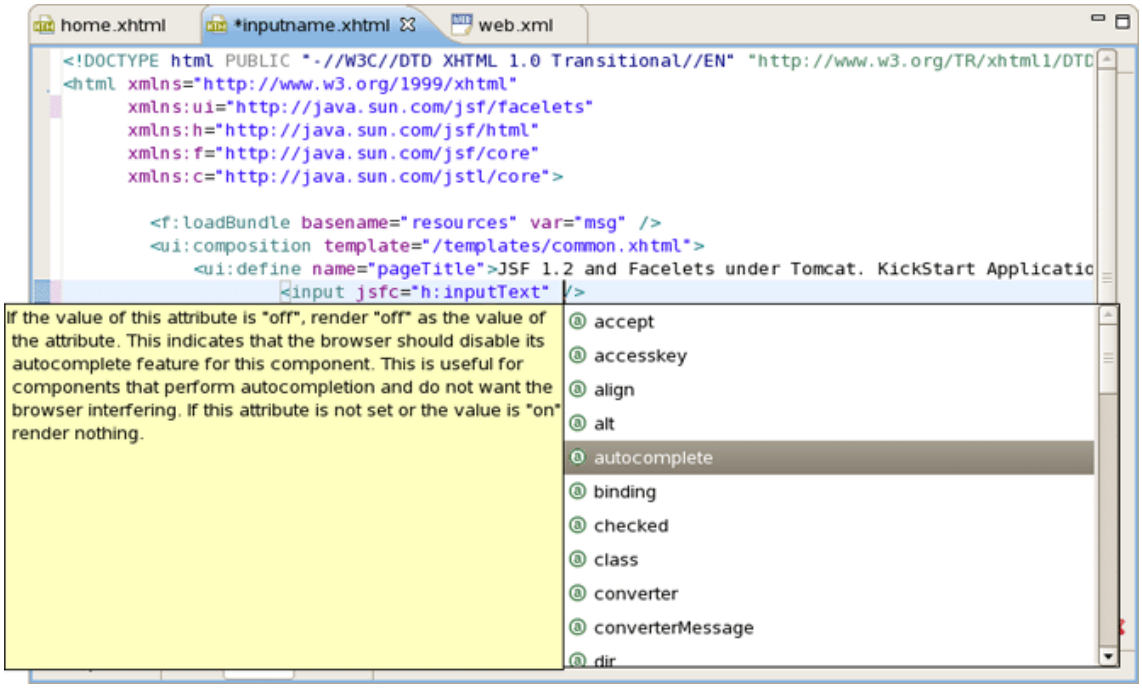

Figure 2.8. Available Attributes for the Component

2.1.4. Open On feature

Finally, Eclipse's *OpenOn* feature for editing Facelets files is supported. Using this feature, you can easily navigate between the Facelets templates and other parts of your projects. Just by

holding down the Control key while hovering the mouse cursor over a reference to a template, the reference becomes a hyperlink to open that template.

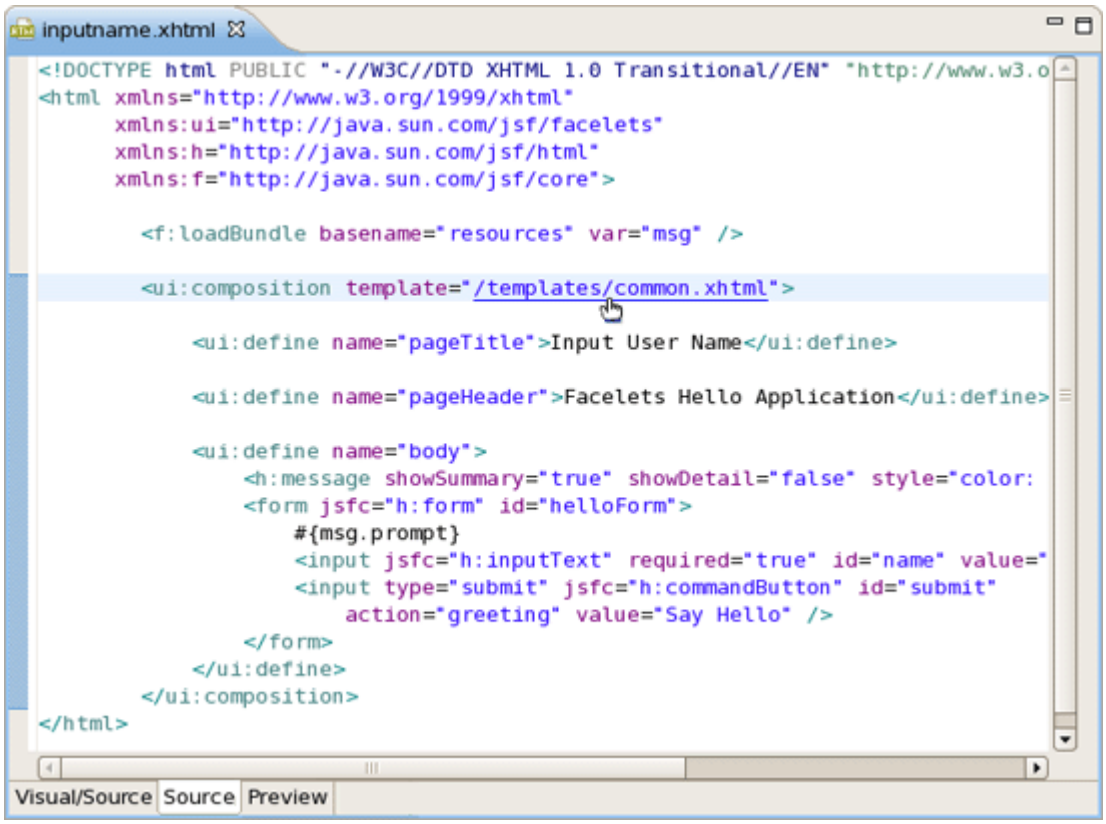

Figure 2.9. Template Hyperlink

Projects

To take an advantage of JSF firstly you should 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.

In this section we're going to stop on each of them in detail.

3.1. Creating a New JSF Project

If you want your project to already contain all JSF libraries, tag libraries and JSF configuration file, just organize a new brand JSF project. It is possible to do this easily with the help of the special wizard. To get it, select *File > New > Project > JBoos Tools Web > JSF > JSF Project* and click *Next*.

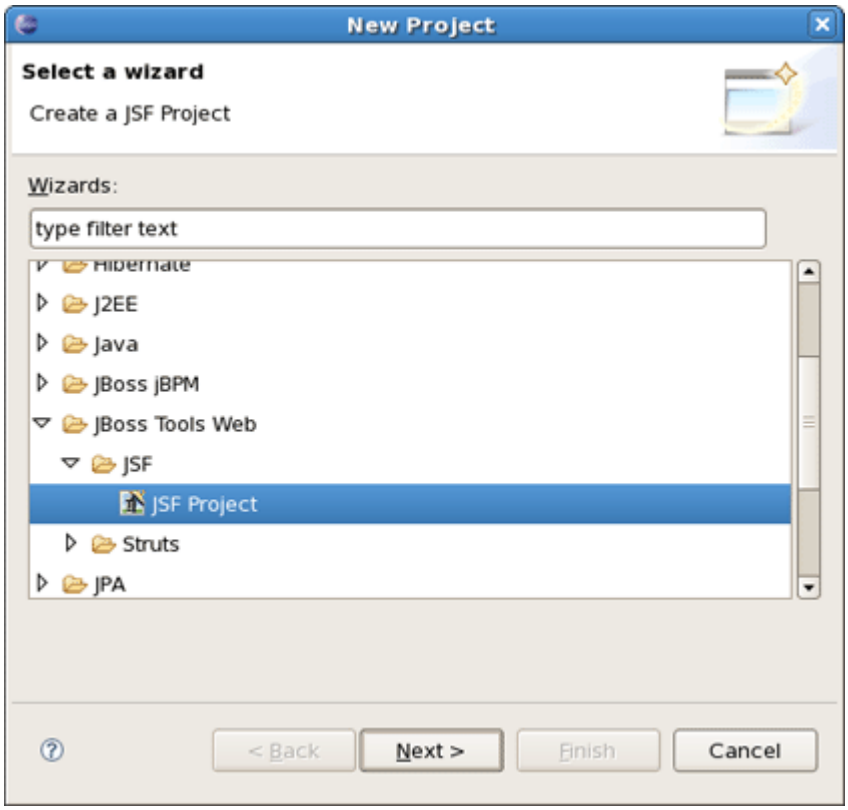

Figure 3.1. Choosing a JSF Project

On the next form you'll be prompted to enter Project Name and select a location for the project or just leave a default path.

Here, JSF Version also allows you to select which JSF implementation to use.

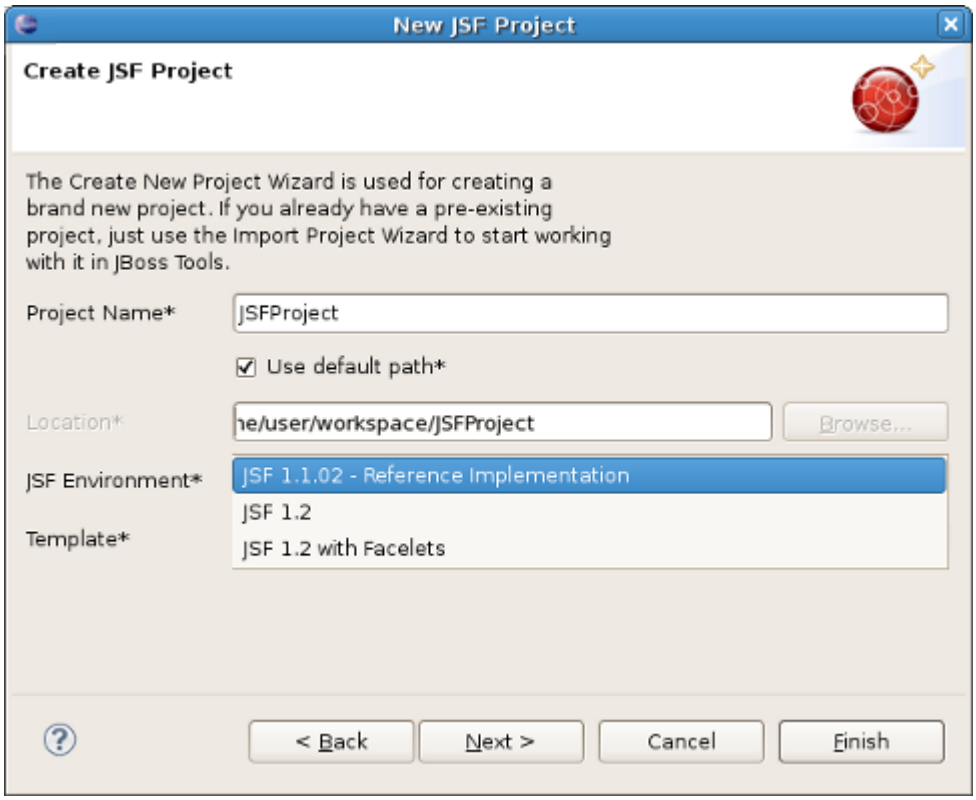

Figure 3.2. Creating a New JSF Project

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

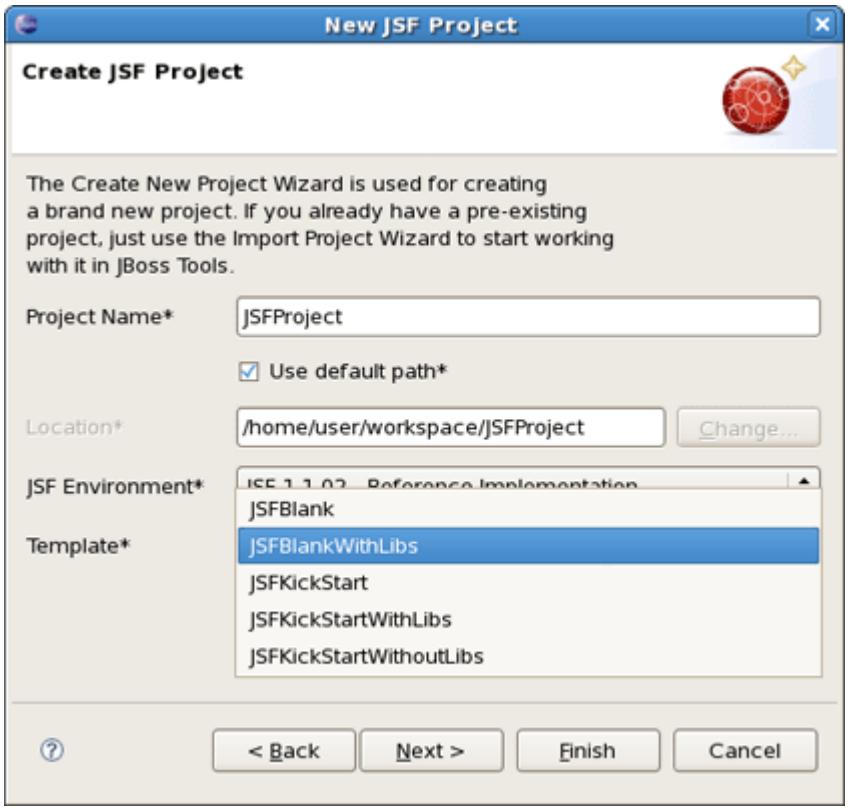

Figure 3.3. Choosing JSF Templates

The table below provides description for each possible JSF template.

Table 3.1. JSF Project Templates

Template	Description
JSFBlankWithLibs	This template will create a standard Web project structure with all JSF capabilities
JSFKickStartWithLibs	This template will create a standard Web project structure but will also include a sample application that is ready to run
JSFKickStartWithoutLibs	Some servers already provide jsf libs and you take risk of getting conflicting libraries while deploying your project. To avoid such conflicts, use a template without libs if you have a server with its own jsf libraries

On the next screen select what *Servlet version* to use and whether to register this application with JBoss AS (or other server) for running and testing your application.

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

The *Runtime* value tells Eclipse where to find Web libraries in order to build (compile) the project. It is not possible to finish project creation without selecting Runtime. If you don't have any values, select *New...* to add new Runtime.

The *Target Server* allows you specifying whether to deploy the application. The Target Server corresponds to the Runtime value selected above. If you don't want to deploy the application, uncheck this value.

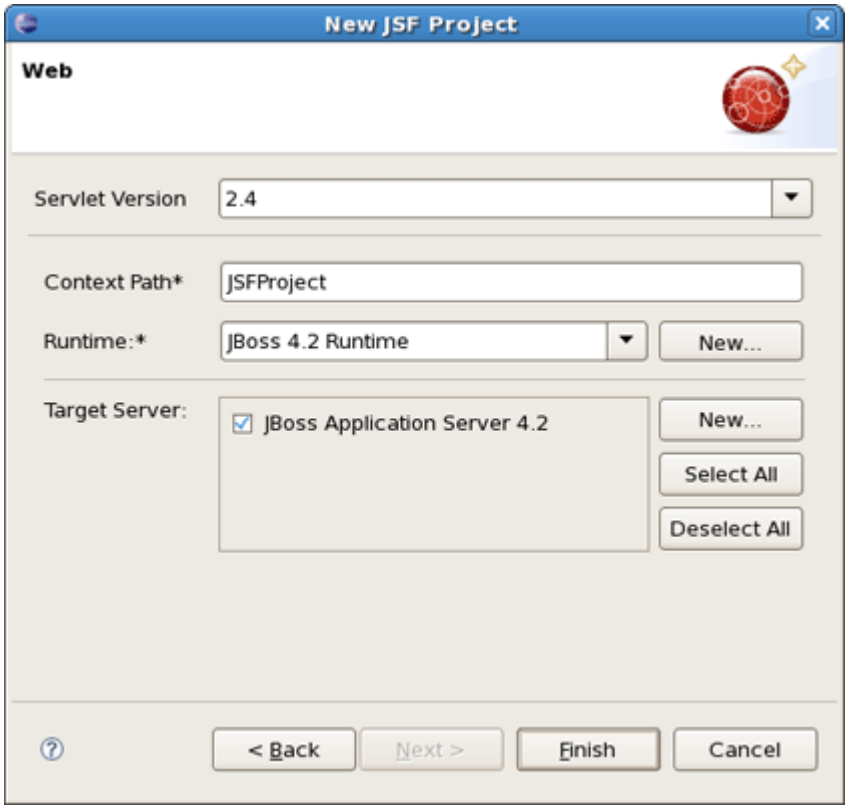

Figure 3.4. Registering the Project on Server

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

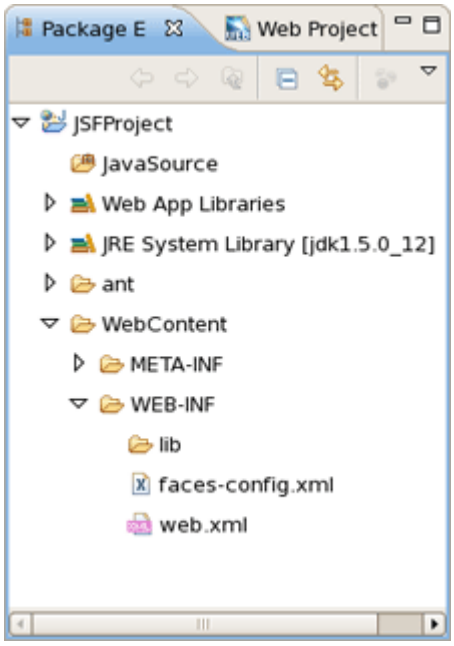

Figure 3.5. A New Project in the Package Explorer

At this point you can open *faces-config.xml* and start working on your application. There are a lot of features to develop JSF applications. We will describe the features further.

3.2. Importing Existing JSF Projects with Any Structure

For detailed information on migration of JSF projects into a workspace see *Migration Guide* [../../Exadel-migration/html_single/index.html#jsf_struts].

3.3. Adding JSF Capability to Any Existing Eclipse Project

It's also possible to add JSF capability (JSF libraries, tag libraries) to any existing Eclipse project in your workspace. After that you'll be able to make use of such editors as JSF configuration editor, JBoss Tools JSP editor and any others.

Right click the project and select *JBoss Tools > Add JSF Capabilities*. This will start the process of adding all necessary libraries, files to make this a Web JSF project.

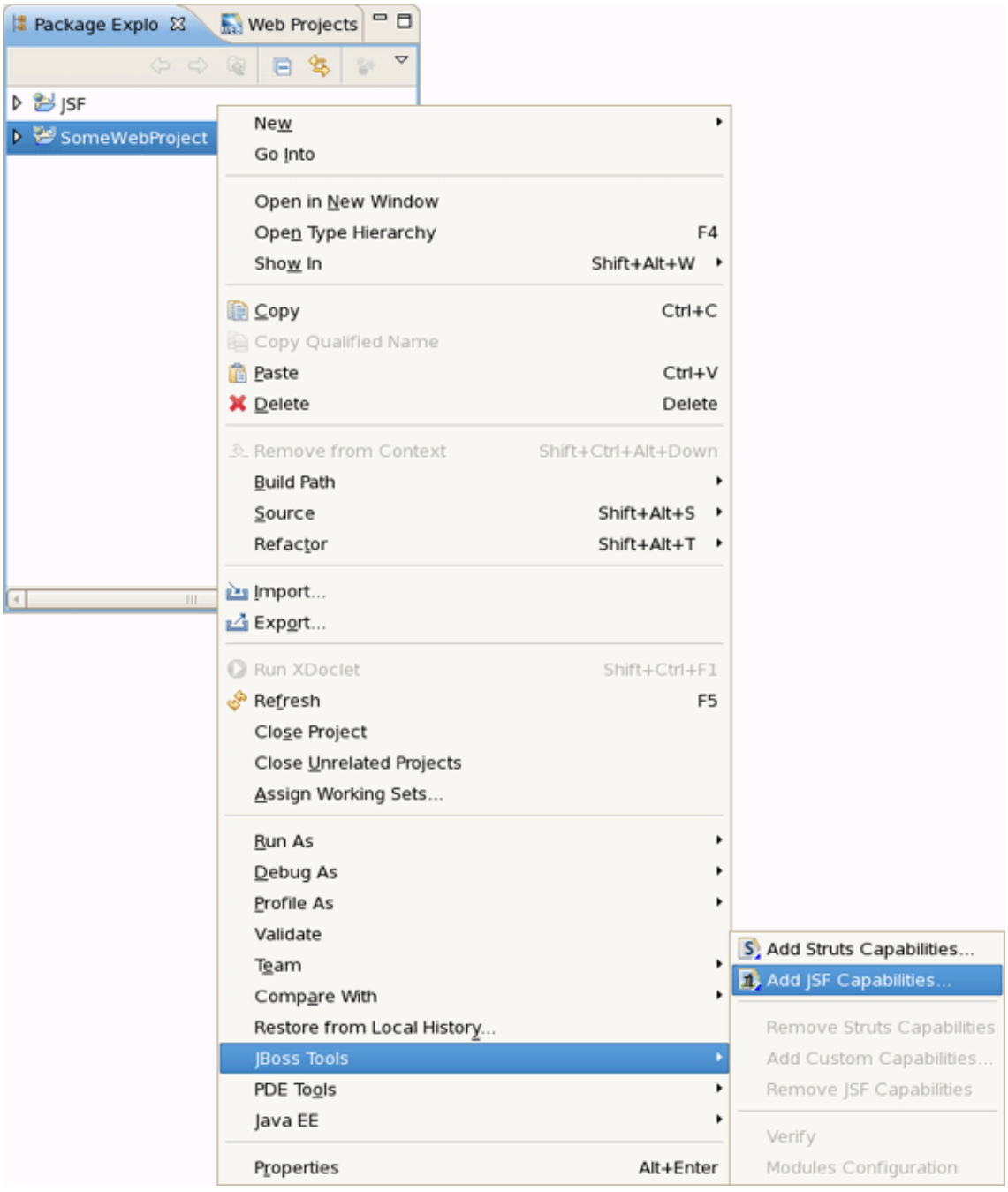

Figure 3.6. Adding JSF Capabilities

The wizard will first ask you to show the *web.xml* file location and the project name.

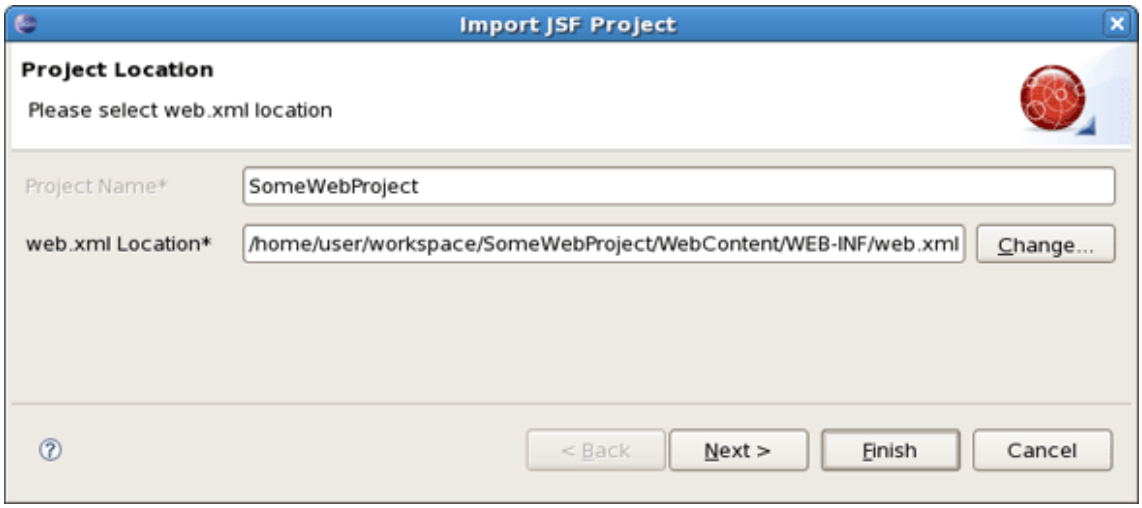

Figure 3.7. Project Location

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

Make sure to select *Add Libraries* to add all required JSF related libraries to this project.

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

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

The *Target Server* allows you to specify whether to deploy the application. The Target Server corresponds to the Runtime value selected above. If you don't want to deploy the application, uncheck this value.

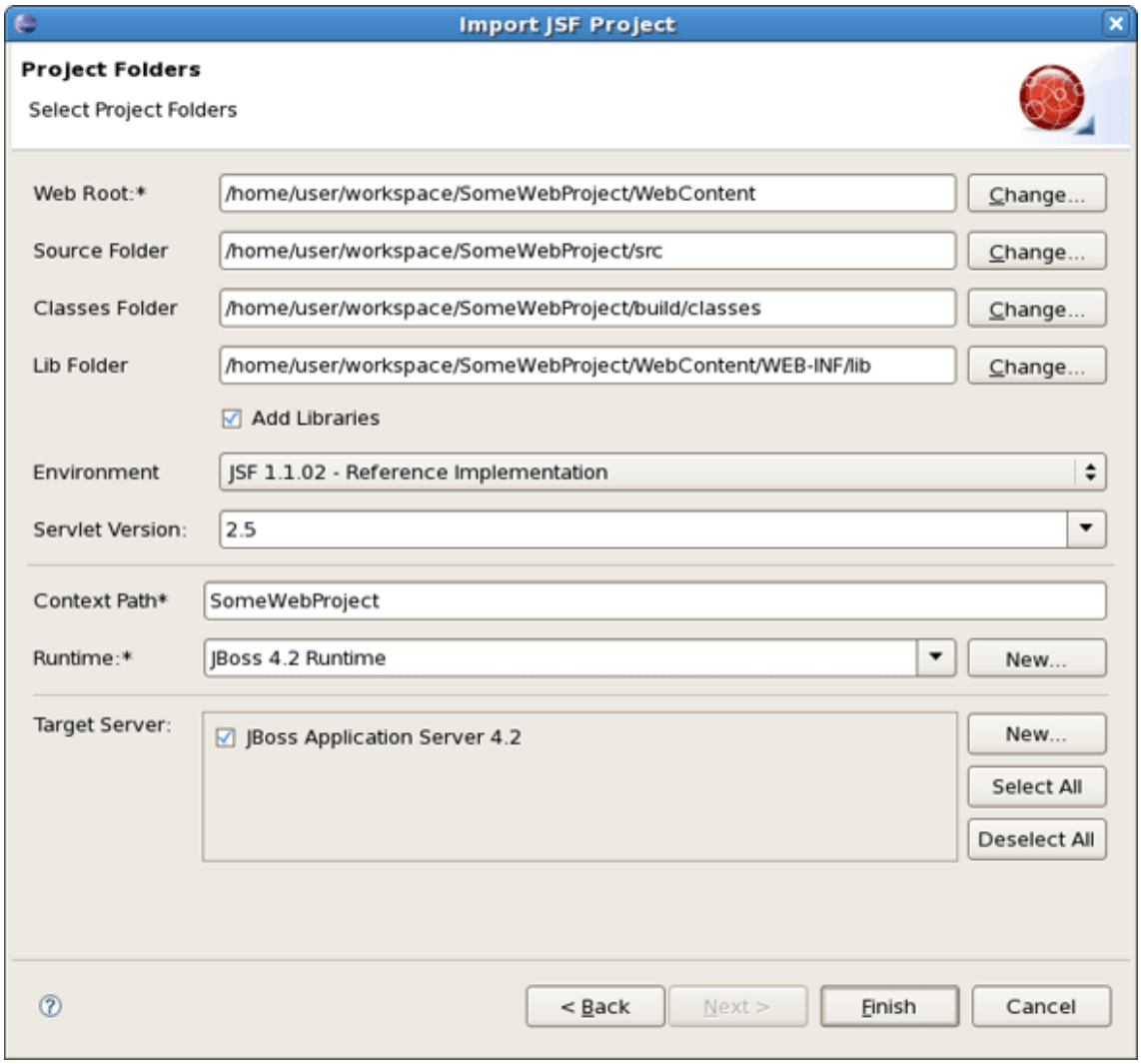

Figure 3.8. Project Folders

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

> **Note:**
>
> Some application servers provide their own jsf implementation libraries. Thus, 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:

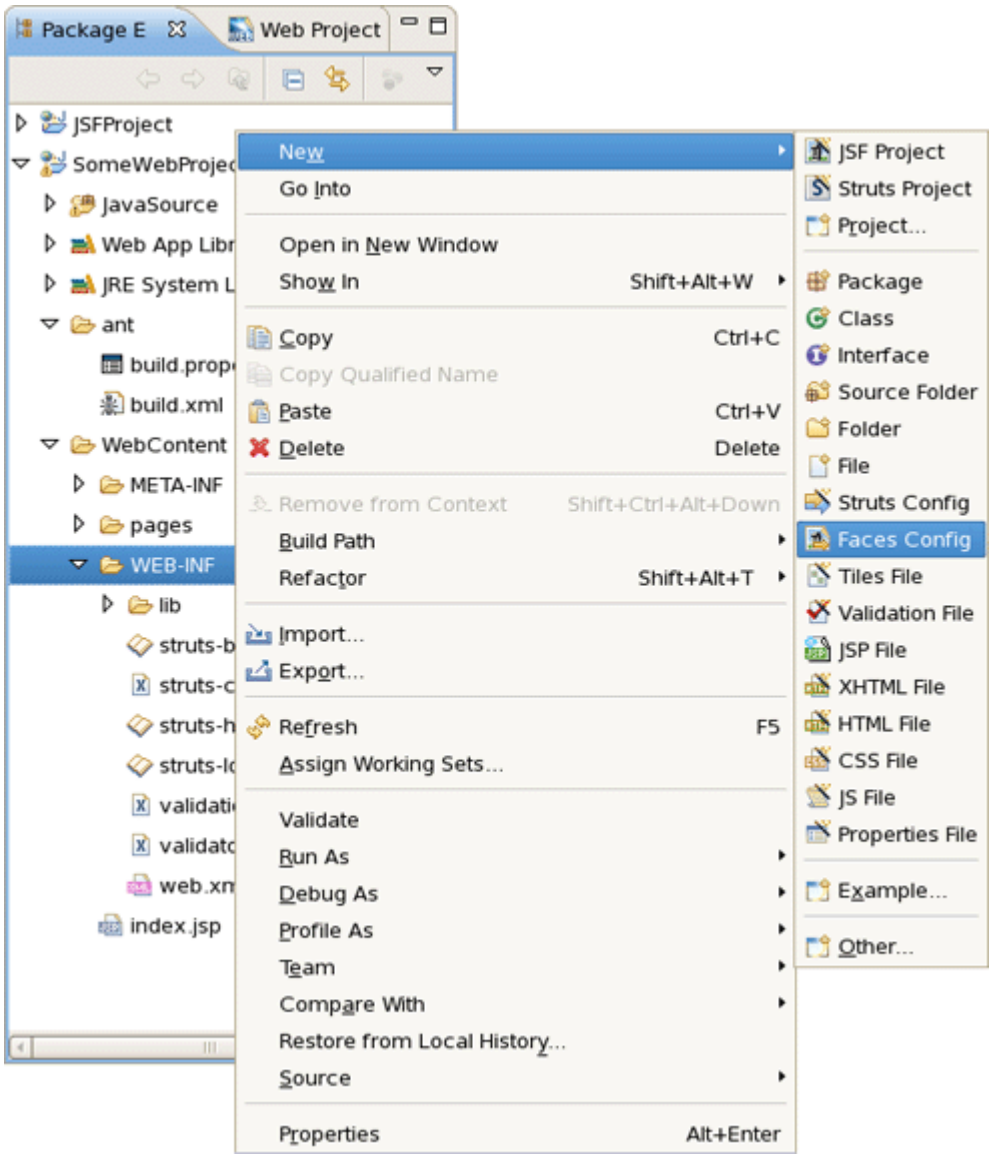

Figure 3.9. Creating a New JSF Configuration File

Once the file has been created, it should be opened in a special *Faces Config Editor*.

3.4. Adding Your Own Project Templates

Template is a set of files that is served as a basis to facilitate the creation of a new project. Project templates provide content and structure for a project.

There is a powerful templating capability for creating new and importing existing Struts and JSF projects. This templating facility has a variety of aspects to consider. But, 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. Follow these steps to make a template out of it:

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

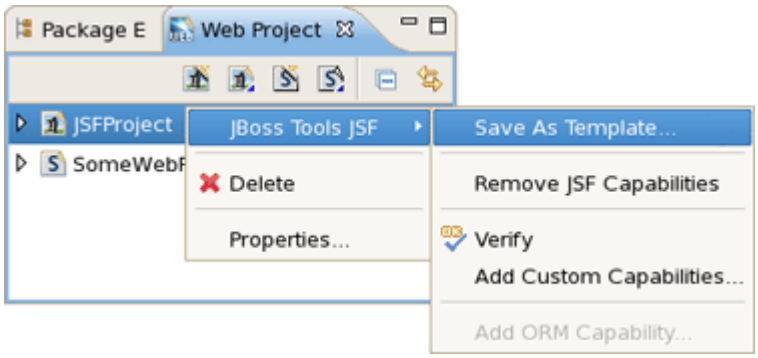

Figure 3.10. Saving Your Project as Template

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

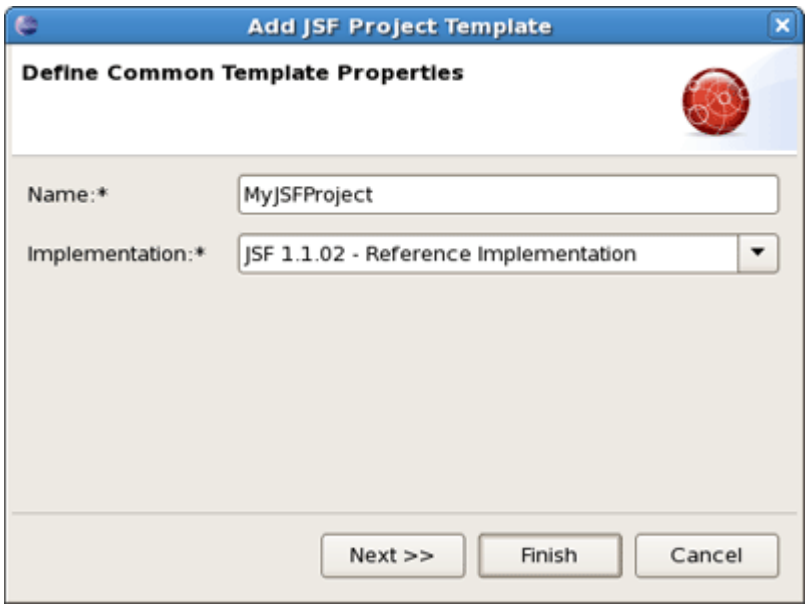

Figure 3.11. Define Template Properties

- Select *Next* and you will be sent to a dialog box with your project structure displayed with check boxes. Here you can check only those parts and files in your project directory that should be part of the template

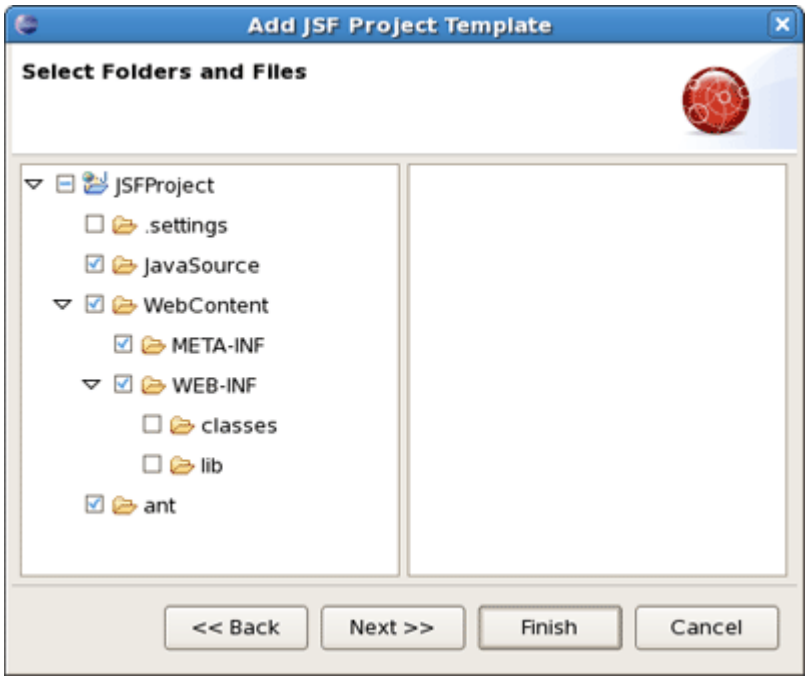

Figure 3.12. Define Template Properties

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

That's it. Now, you can use this template 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 and now you can bring some new logic to it starting from JSF configuration file.

3.5. Relevant Resources Links

You can find more in-depth explanation on how to work with special wizards, editors and views that can be used in various scenarios while developing JSF applications in our *Visual Web Tools guide* [../../jsf/html_single/index.html].

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 that gives a descriptive information on the *web.xml editor* [../../jsf/html_single/index.html#GraphicalWebApplicationFileEditor].

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 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 in the faces-config.xml:

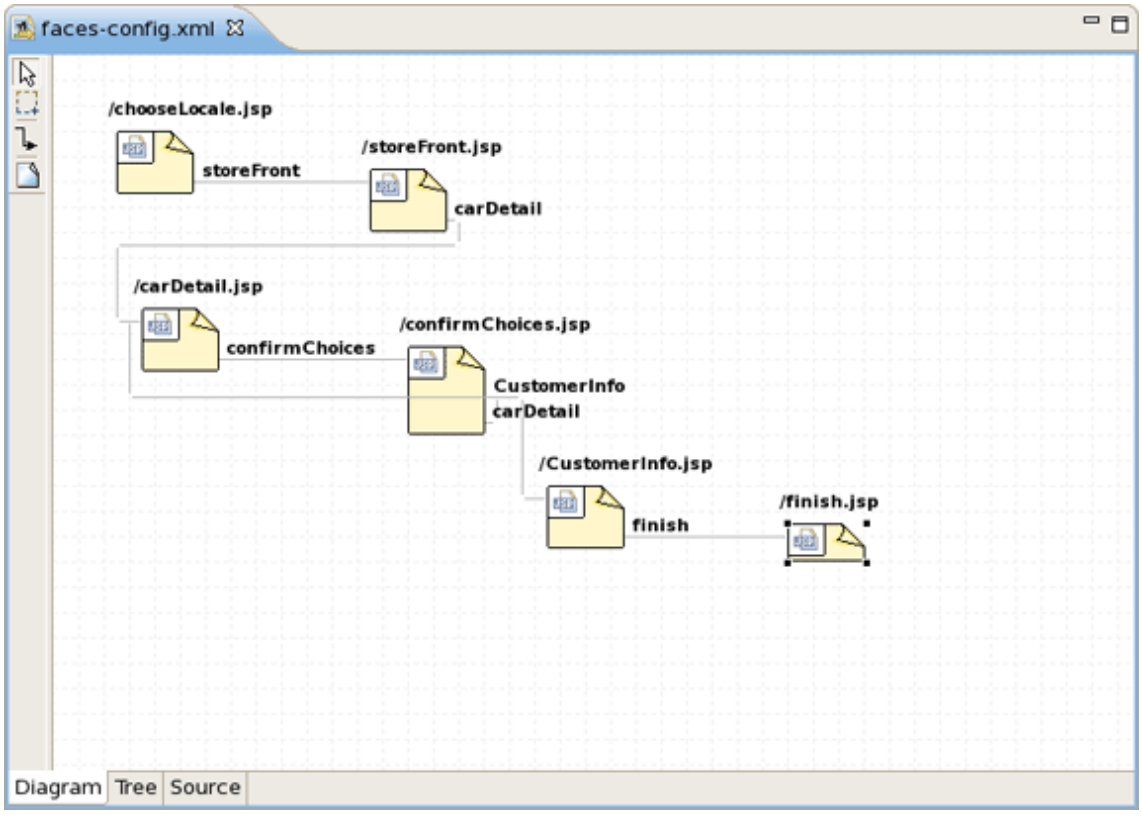

Figure 5.1. Diagram View

If your diagram is large, 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. It 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:

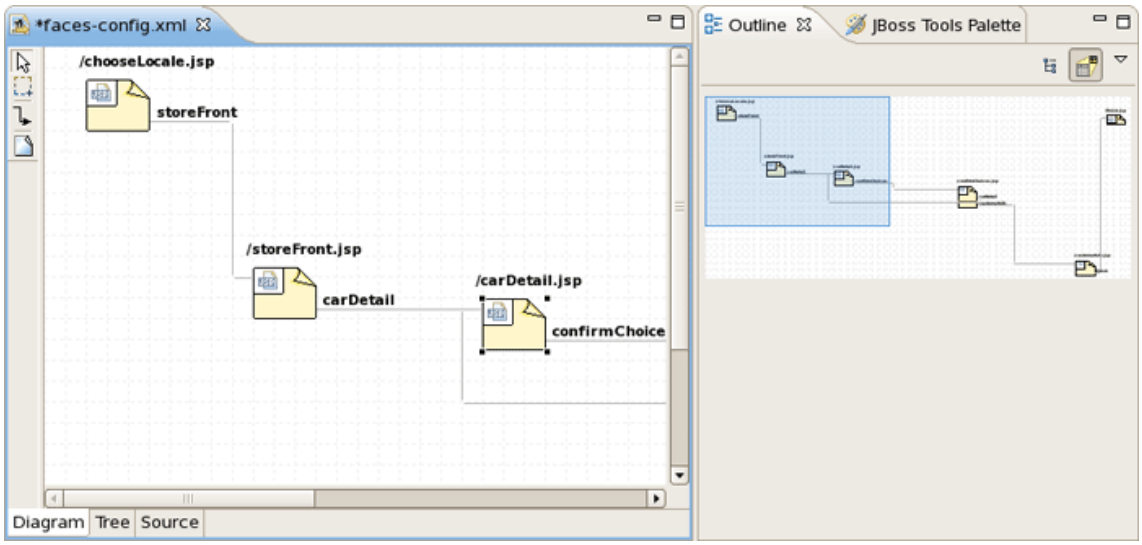

Figure 5.2. Outline View for Diagram

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:

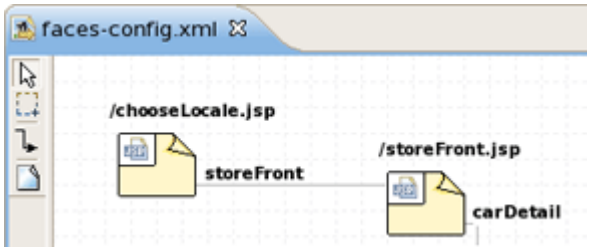

Figure 5.3. Transition between JSP Pages

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

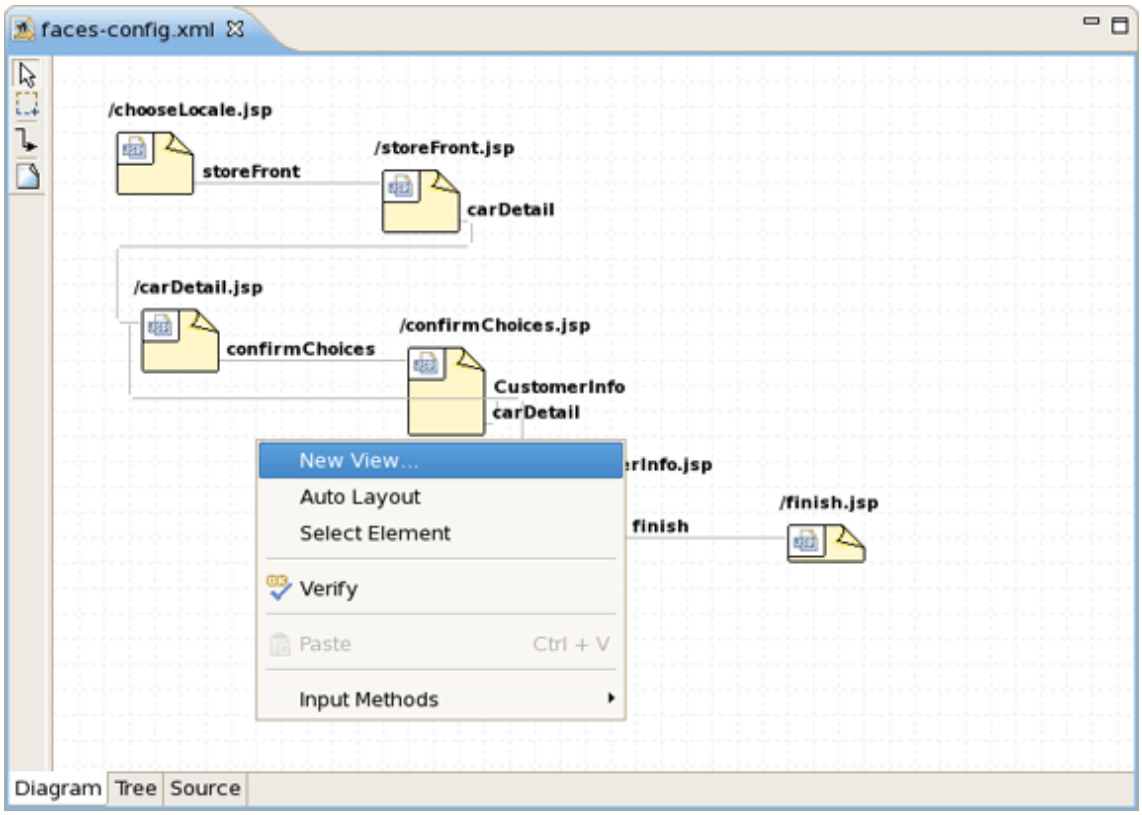

Figure 5.4. Creating a New View

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:

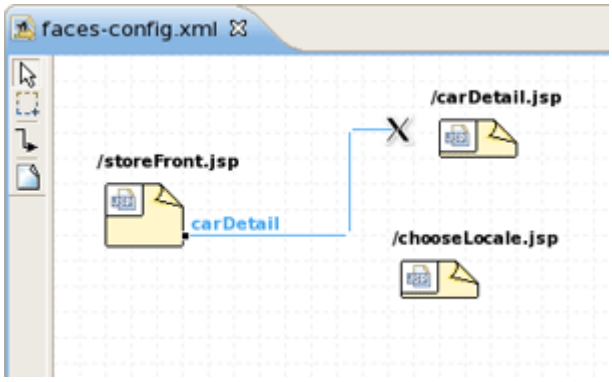

Figure 5.5. Editing Transition between Views

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 see 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, possibility to set extensions, add property and variable resolvers, etc.

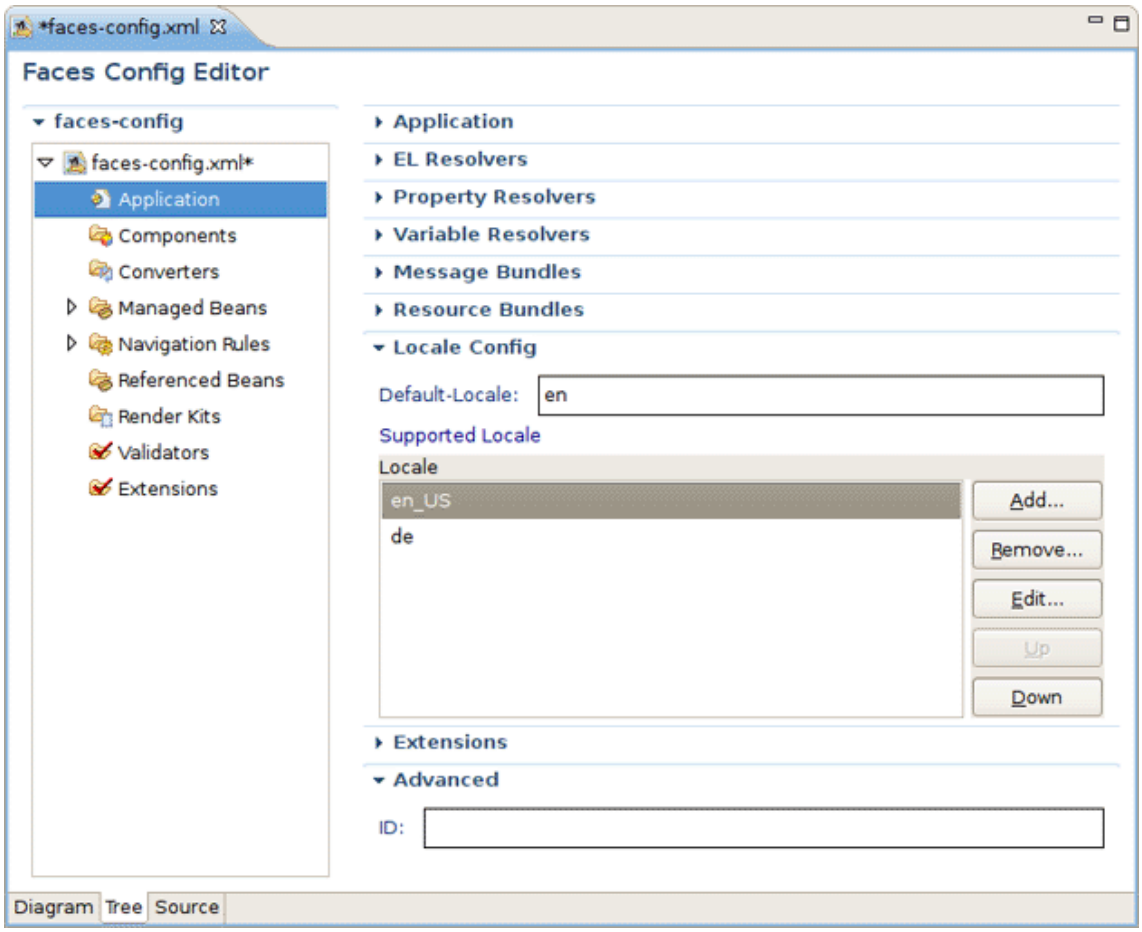

Figure 5.6. JSF Application Specific Settings

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

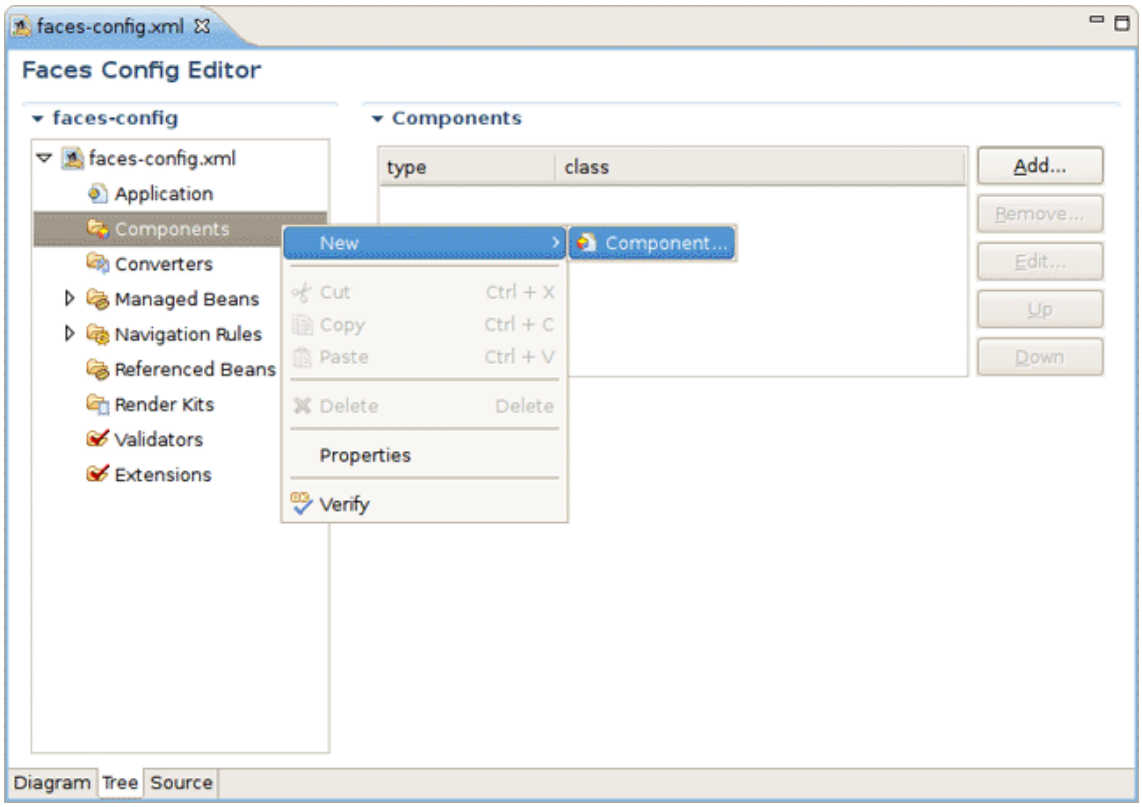

Figure 5.7. Registering a New JSF Component

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.

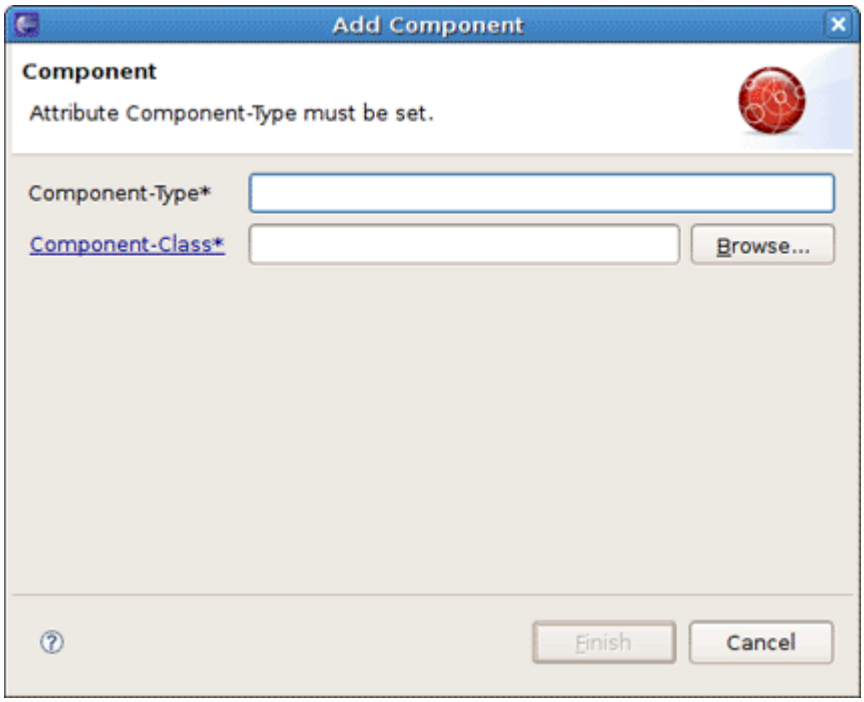

Figure 5.8. Adding a New JSF Component to the JSF Configuration File

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

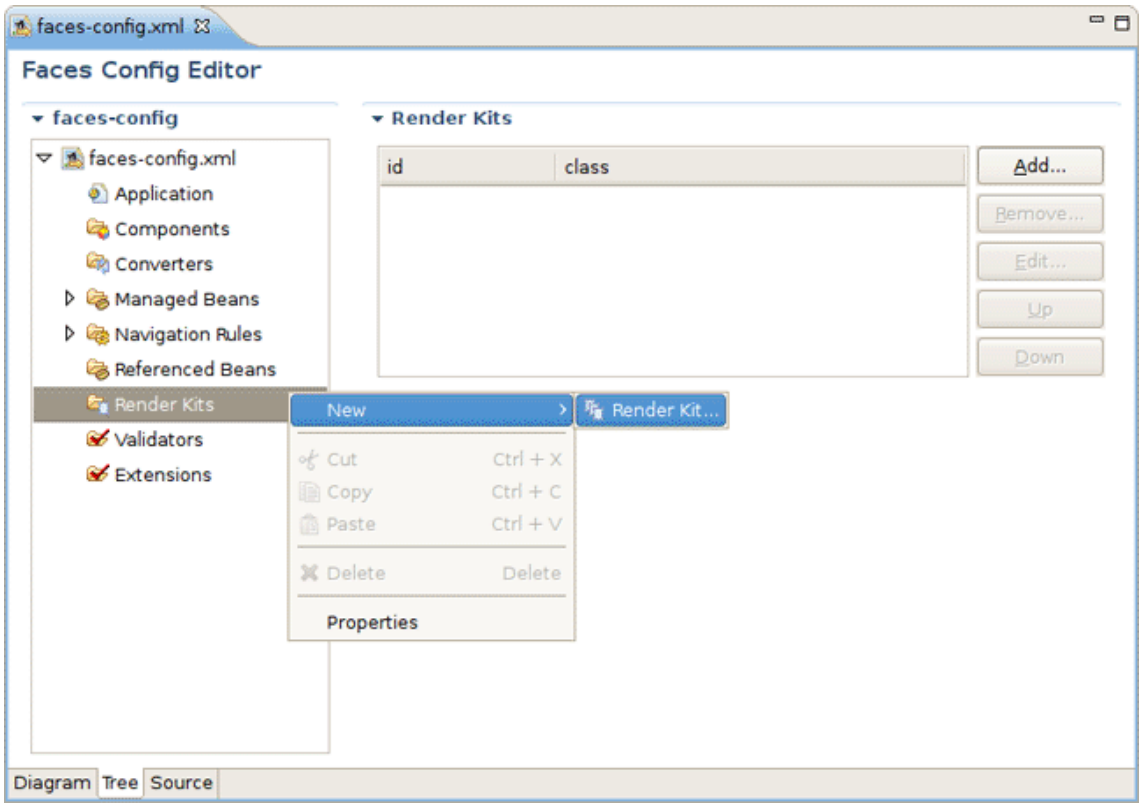

Figure 5.9. Adding a New JSF Component to the JSF Configuration File

- Under the *Converters* node you can create a converter class for your JSF application either with id or for a proper class. How to do that see the *Create and Register a Custom Converter* section.

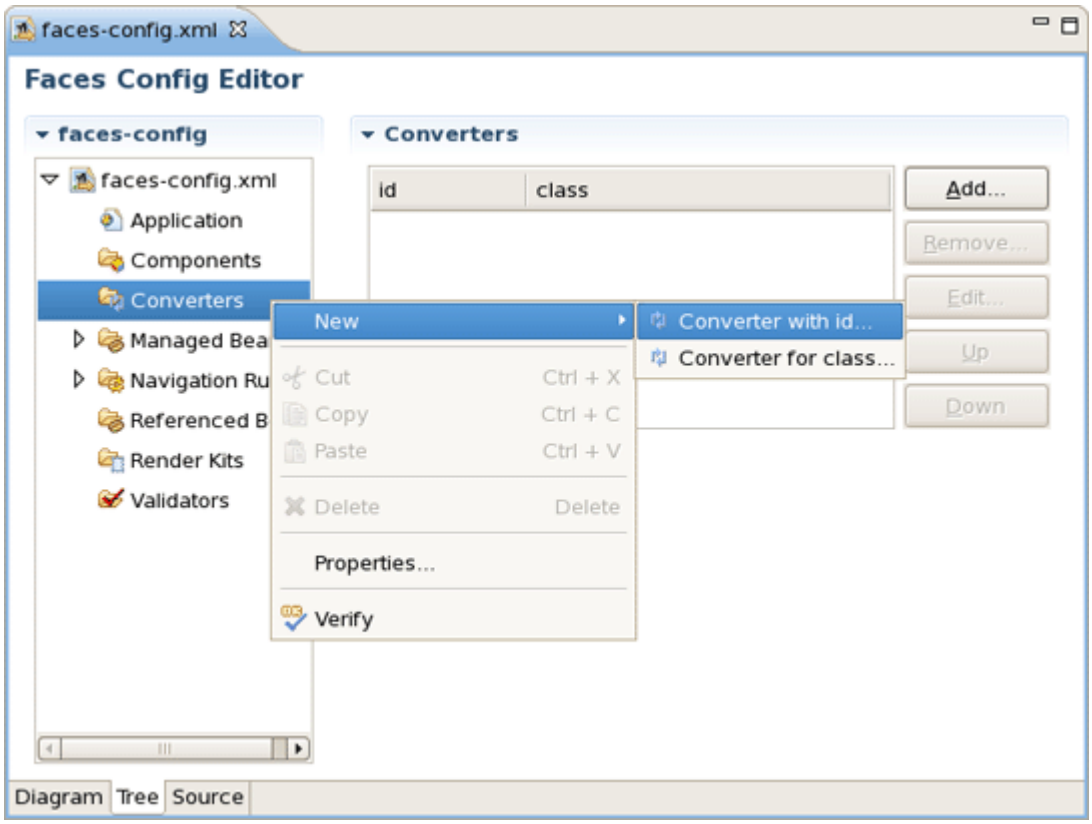

Figure 5.10. Creating a New Custom Converter

- The *Managed Bean* node is meant for creating and registering Bean classes in your JSF application. Read more on the topic in the *Managed Beans* chapter.

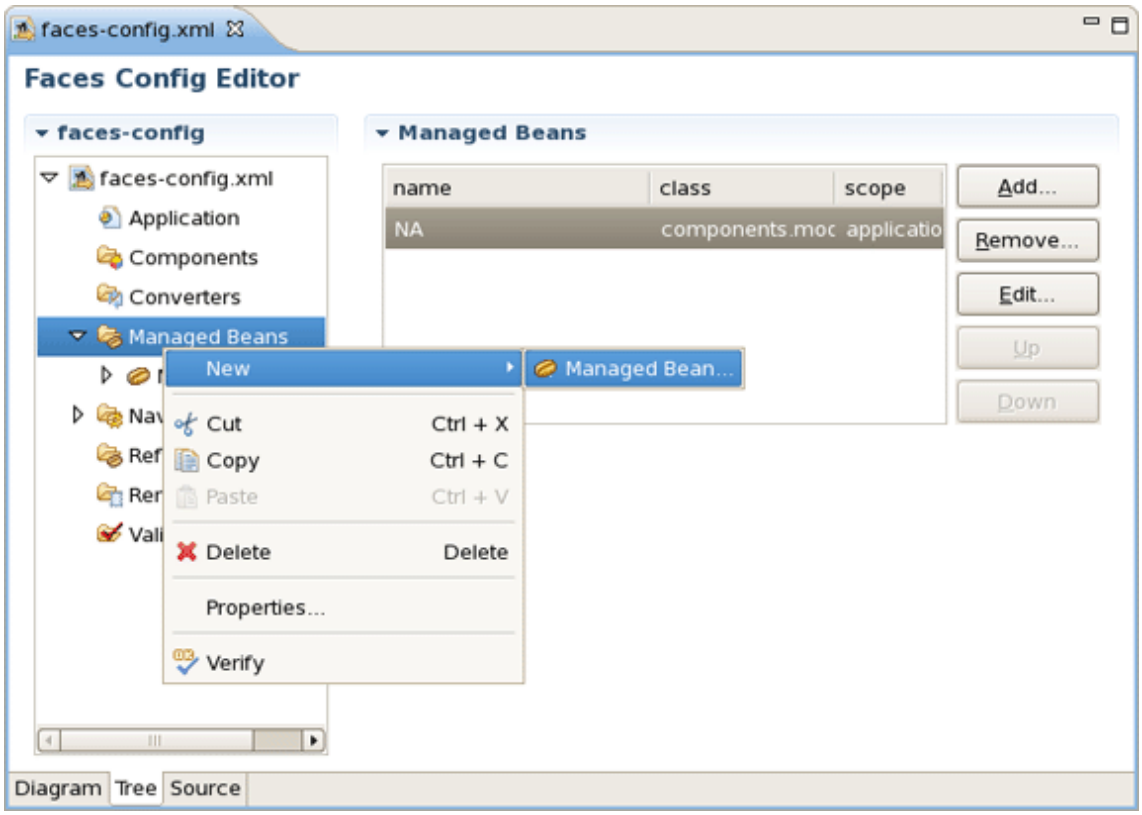

Figure 5.11. Managed Beans

- Use the *Navigation Rules* node to configure a navigation between the pages in your application: 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.

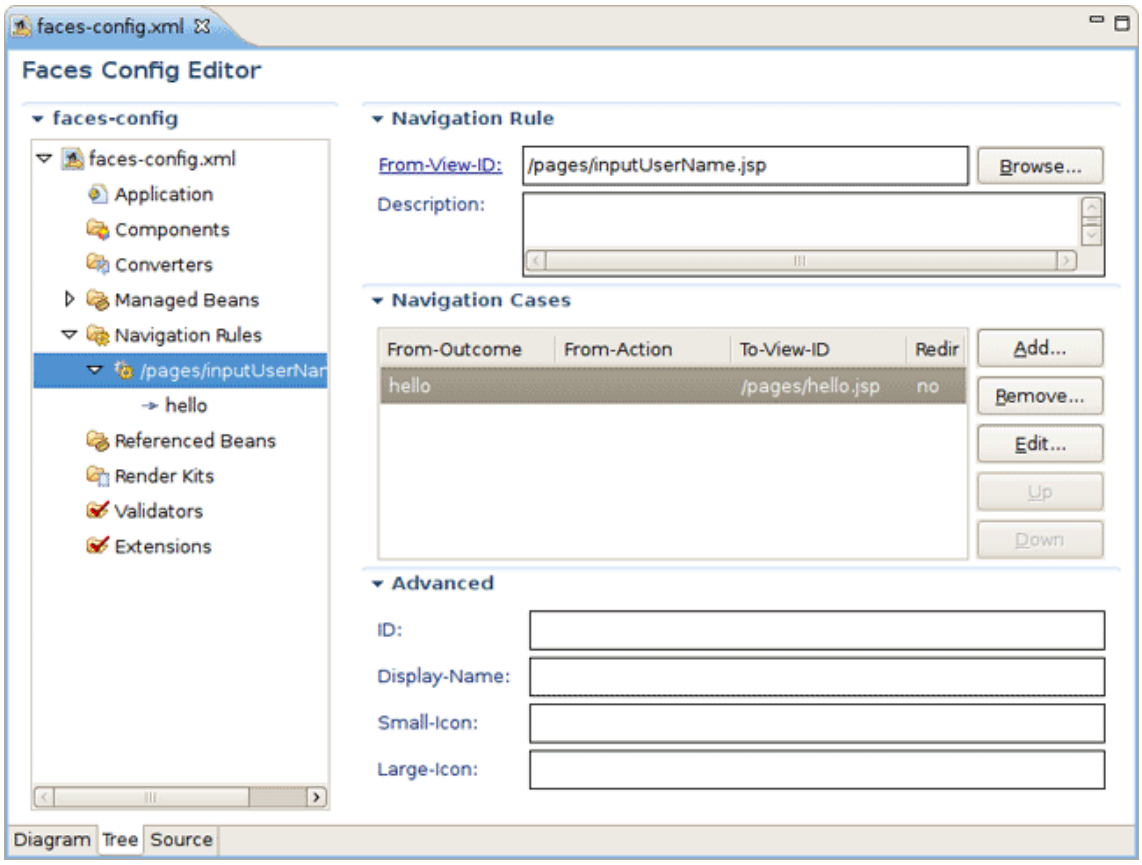

Figure 5.12. Configuring Navigation Rules

- 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 the *Create and Register Referenced Beans* section.

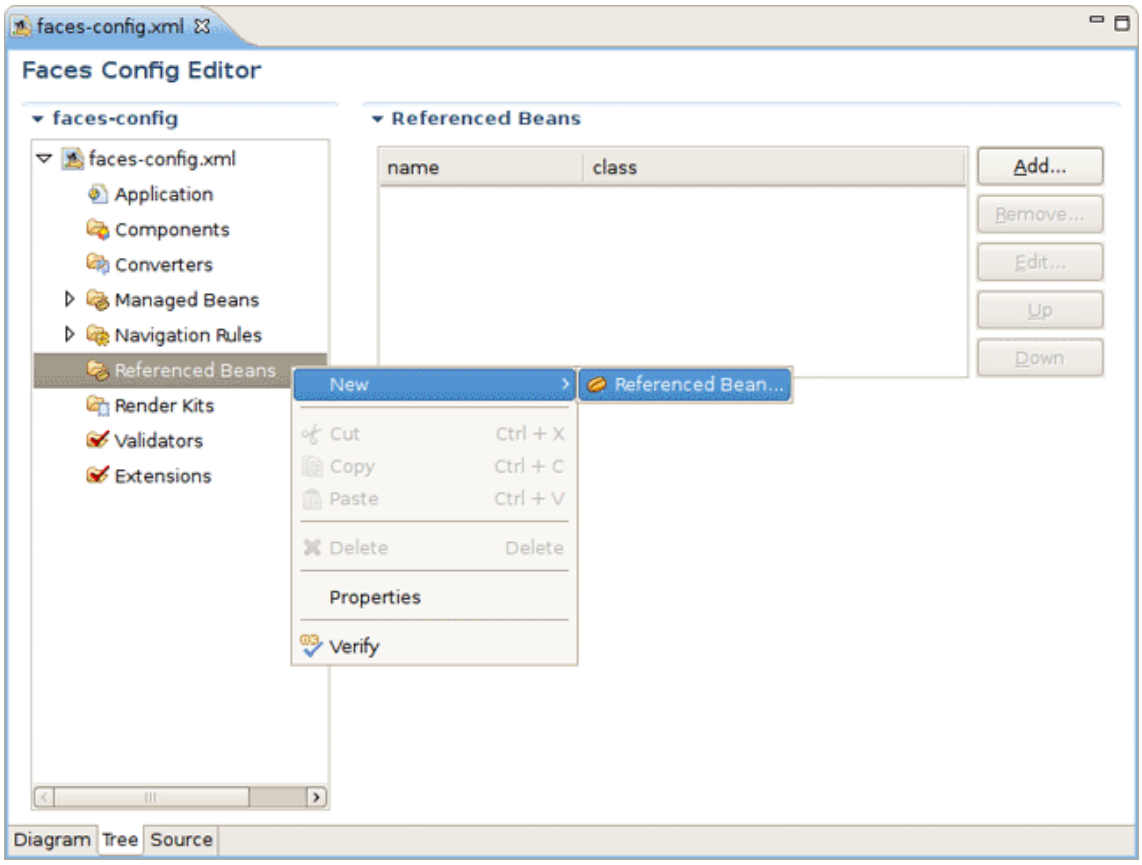

Figure 5.13. Referenced Beans

- 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 the *Create and Register a Custom Validator* section.

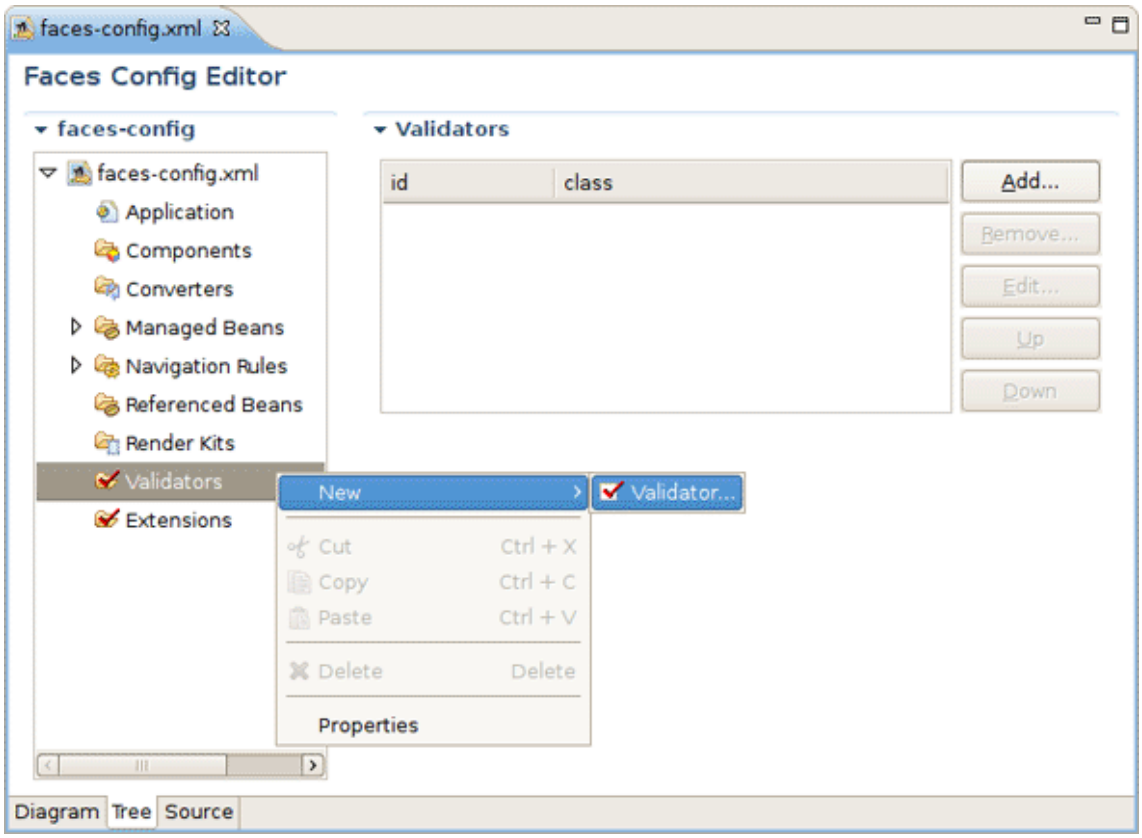

Figure 5.14. Validators

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

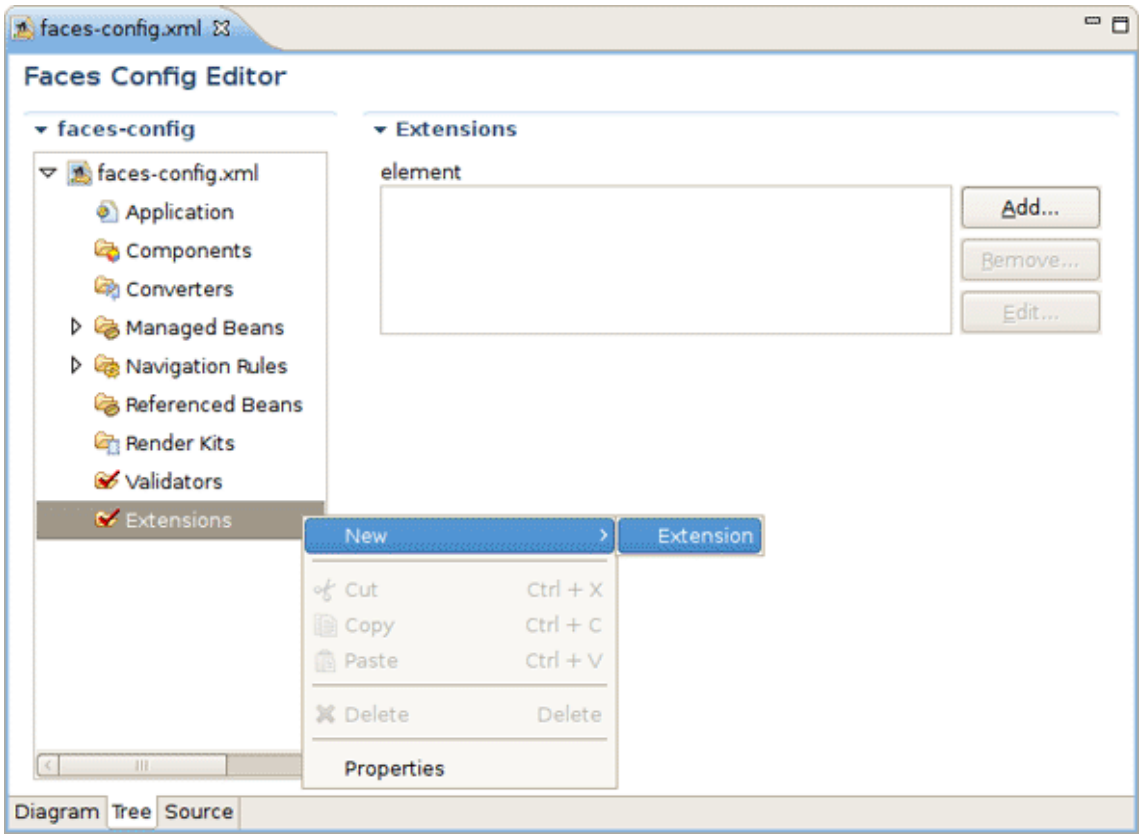

Figure 5.15. Adding Extensions

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

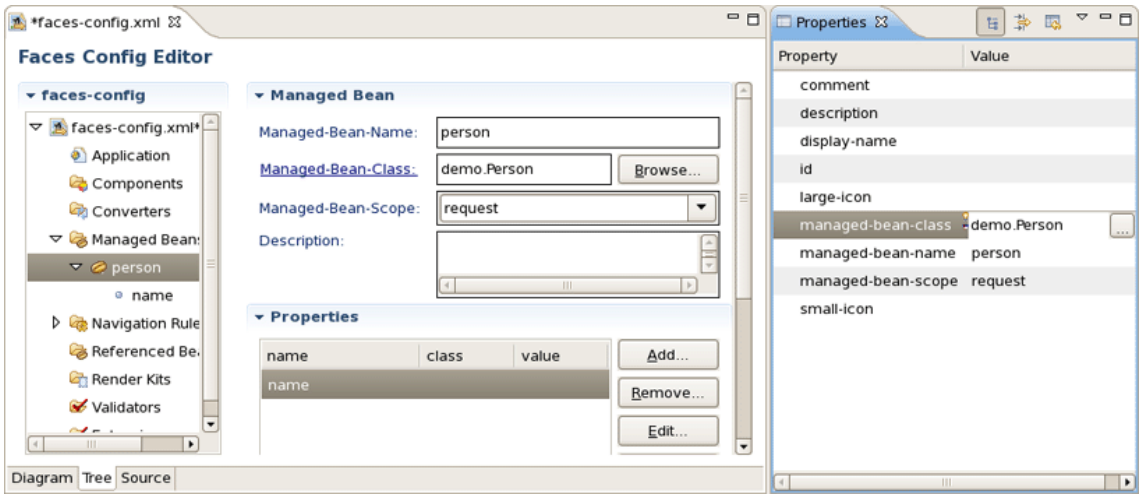

Figure 5.16. Properties View

5.3. Source View

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

The Source view for the editor displays a 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:

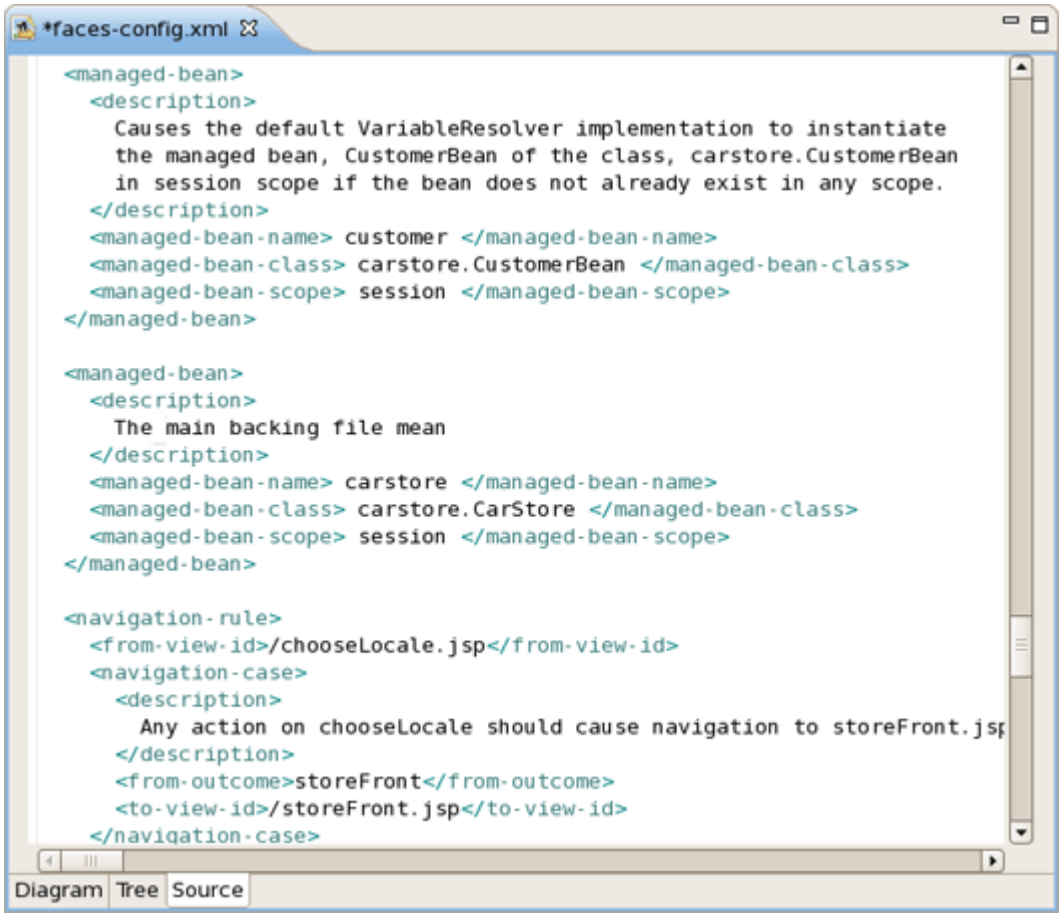

Figure 5.17. Source View

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.

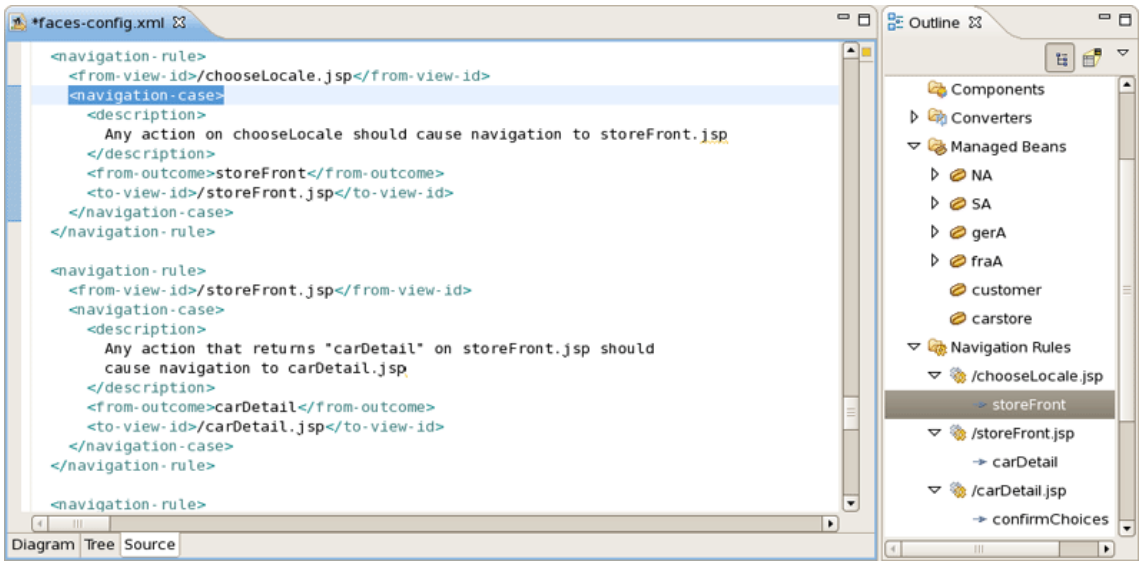

Figure 5.18. Outline View

5.4. Editor Features

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

5.4.1. Open On

The JSF configuration file editor comes with a very useful OpenOn navigating feature. More fully you can read about it in our *Visual Web Tools Guide* [../../jsf/html_single/index.html#OpenOnSelection4Hyperlinknavigation].

5.4.2. Code Assist

Code Assist provides 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:

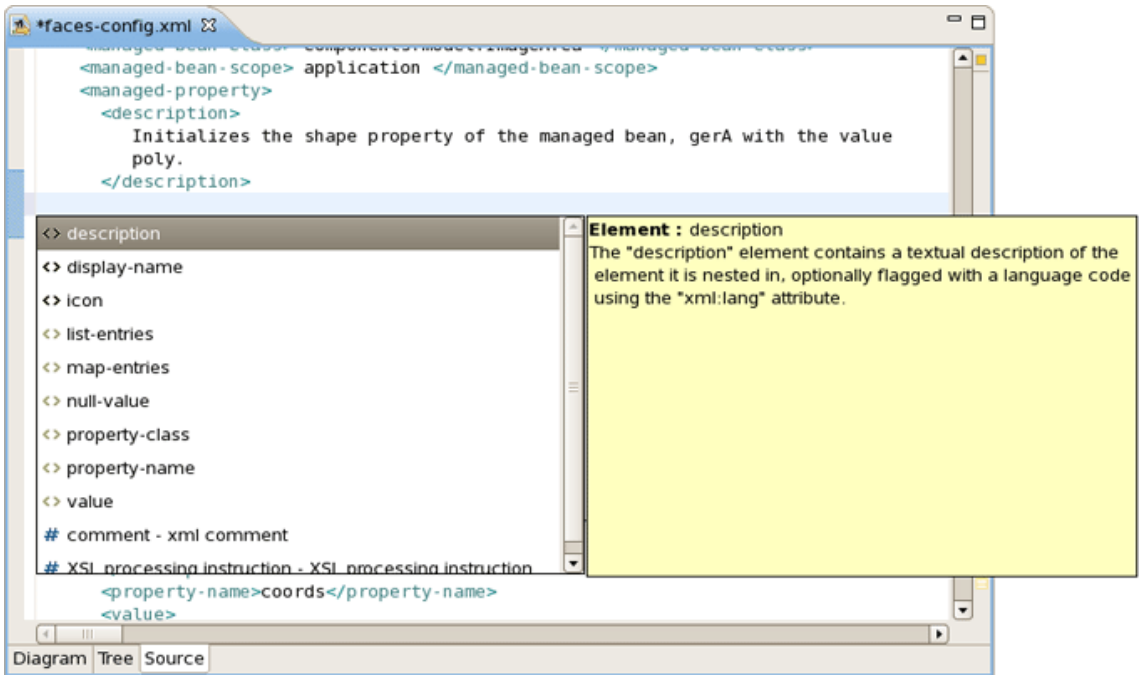

Figure 5.19. Code Assist in Source View

5.4.3. Error Reporting

When you are developing your project, error checking is constantly provided. This greatly reduces your development time as it allows you to catch many of the errors during development.

Errors will be reported by *verification* facility:

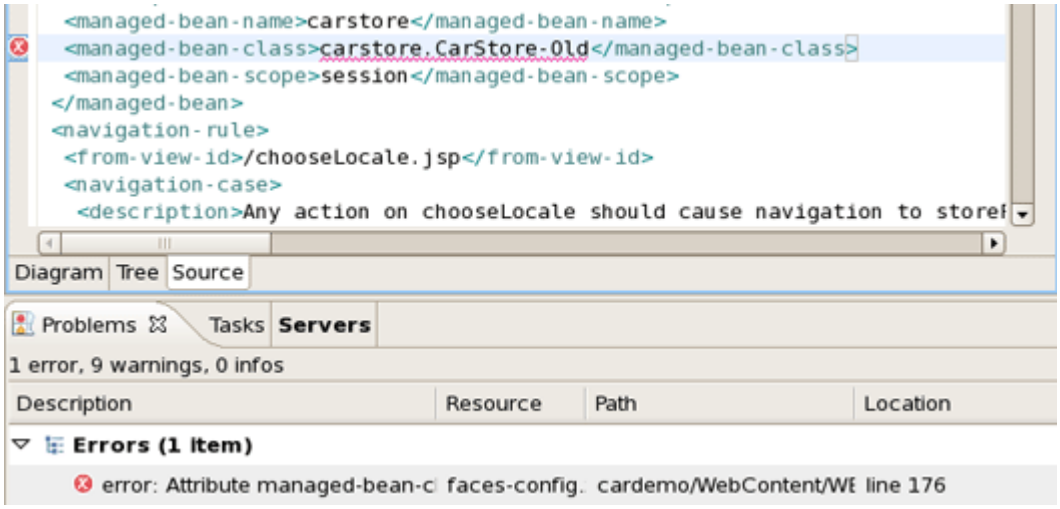

Figure 5.20. Error Reporting in Source View

Other errors are also reported.

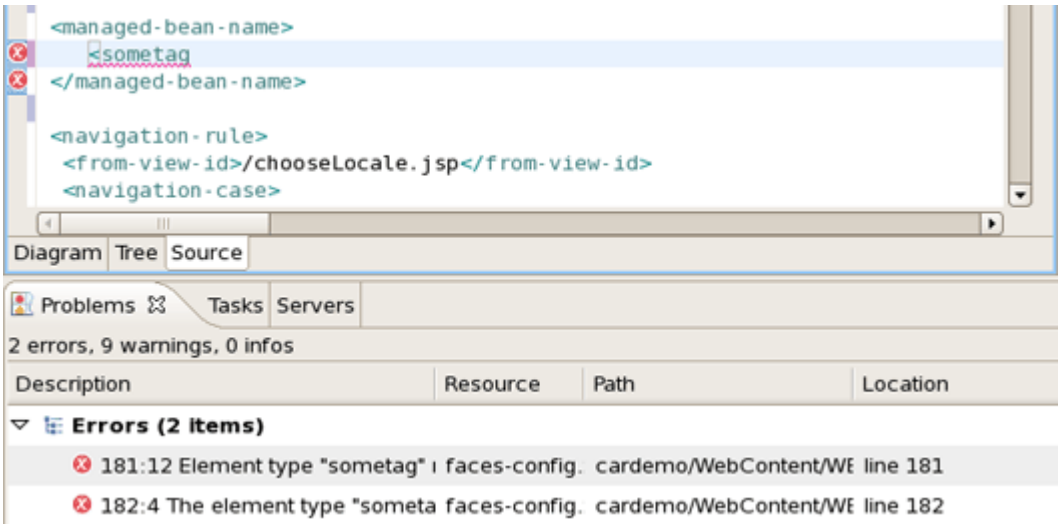

Figure 5.21. Other Errors Reporting

Managed Beans

There is lots of power to work with managed beans.

- Add and generate code for new managed beans
  - Generate code for attributes and getter/setter methods
- Add existing managed beans to JSF configuration file

Thus, in this section we will guides you through all this possibilities.

6.1. Code Generation for Managed Beans

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

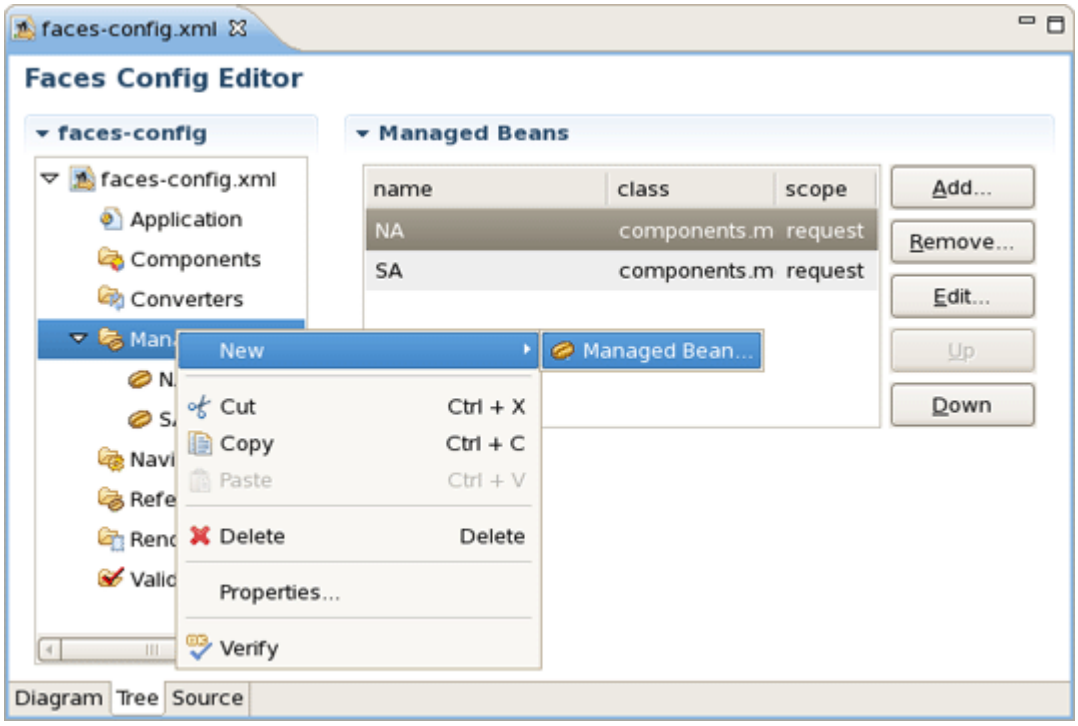

Figure 6.1. Creation of New Managed Bean

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

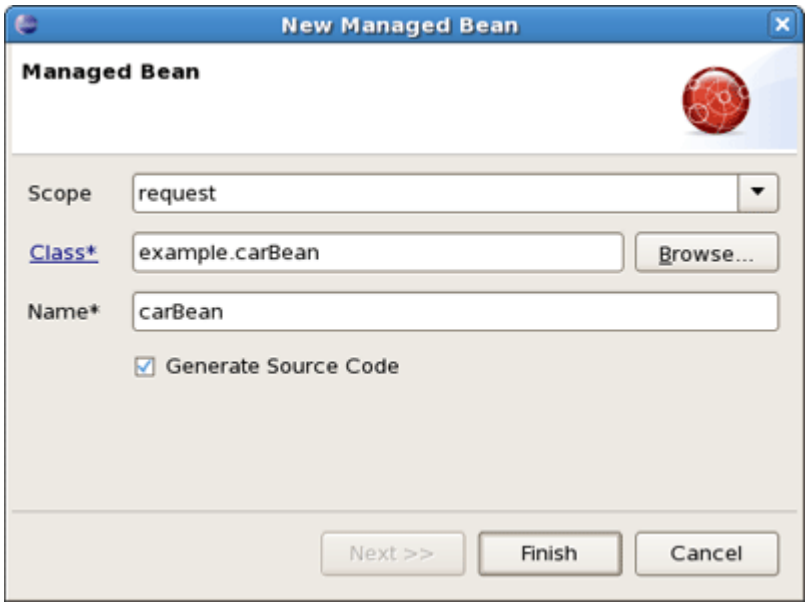

Figure 6.2. New Managed Bean

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

- click on *Managed-Bean-Class* link in the editor

or

- right click the *managed bean* and select *Open Source*

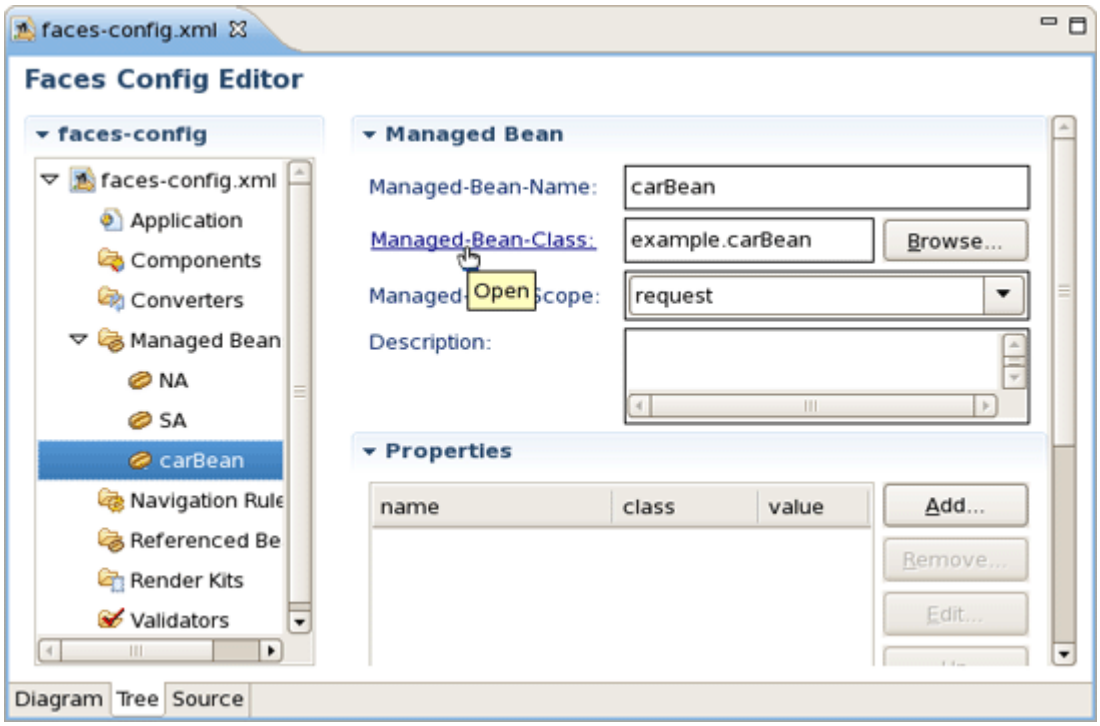

Figure 6.3. Opening of Created Managed Bean

The generated Java source should look as follows:

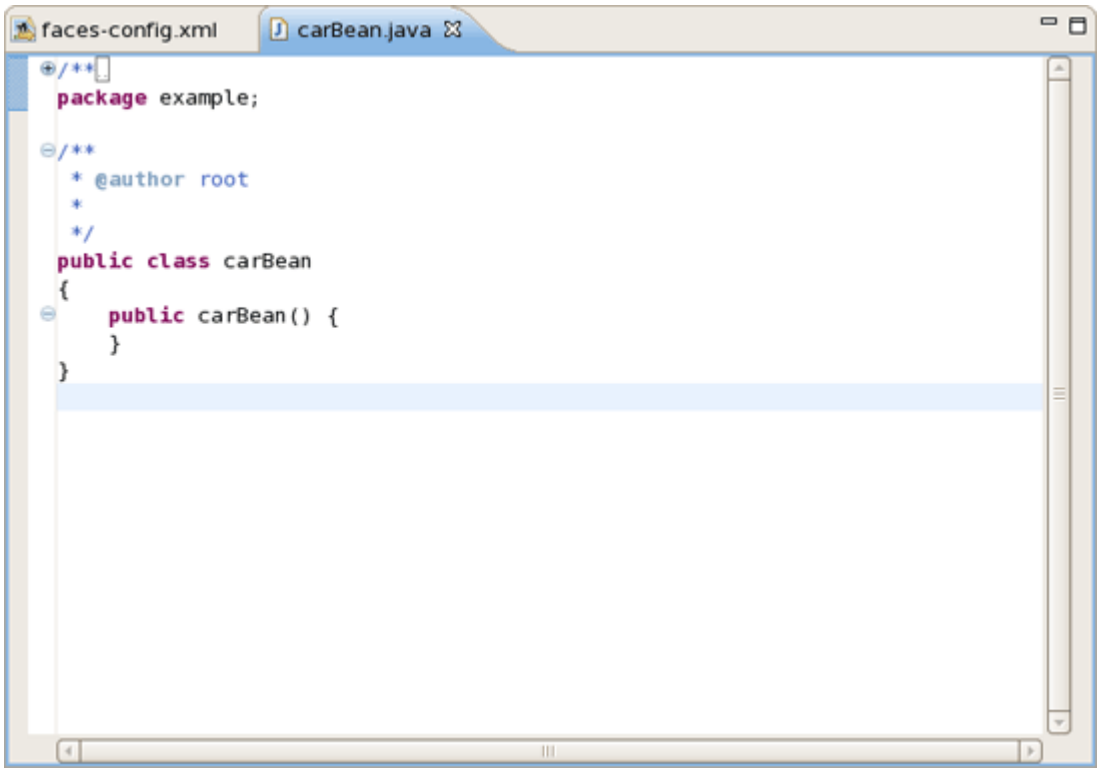

Figure 6.4. Java Source Code

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 see Add Property dialog.

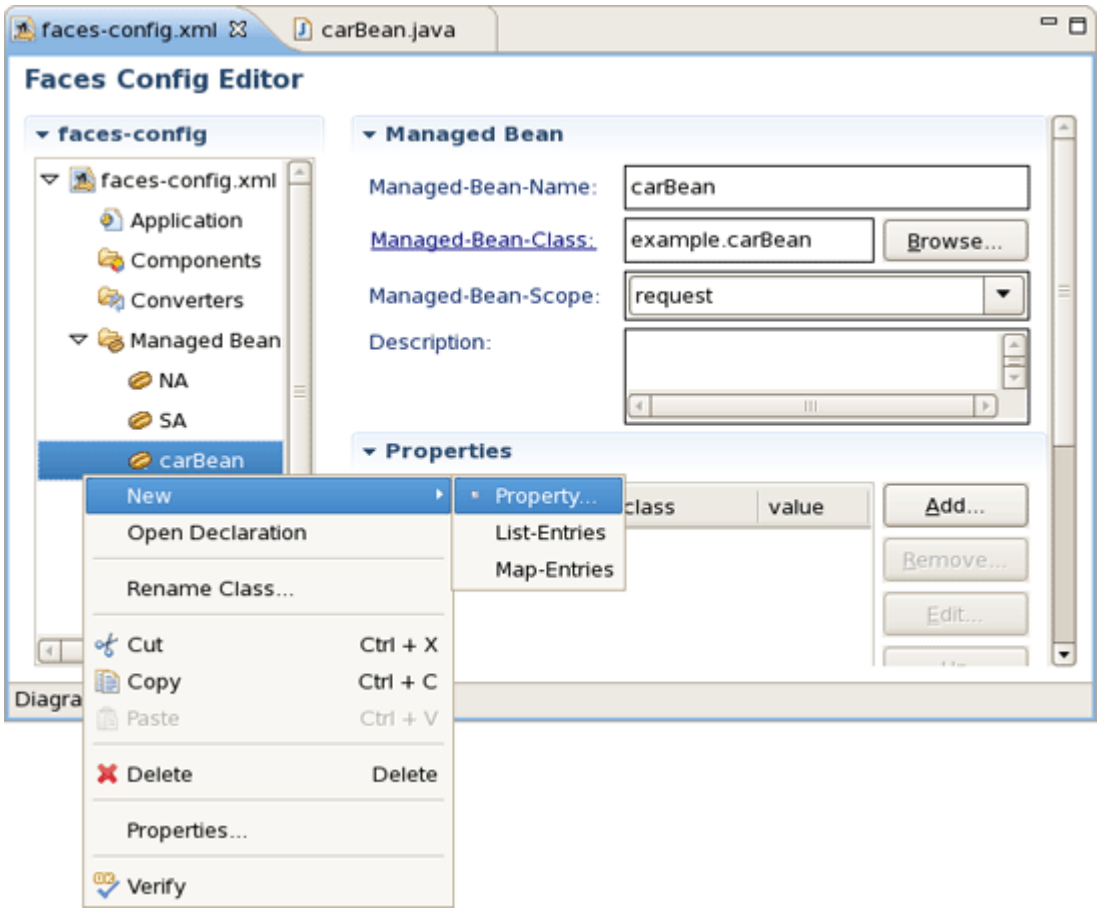

Figure 6.5. Generation of Source Code for Properties

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 added property with *"get"* and *"set"* methods:

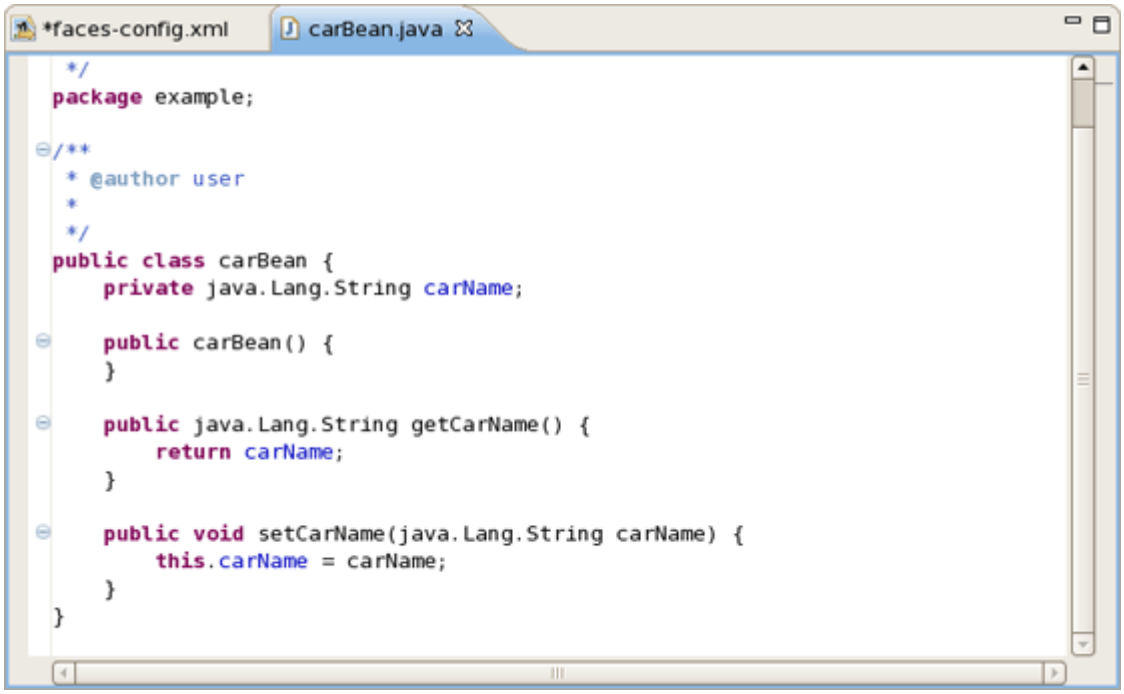

Figure 6.7. Generated Java Source Code for Property

Thus, we've discussed everything which comes to 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.

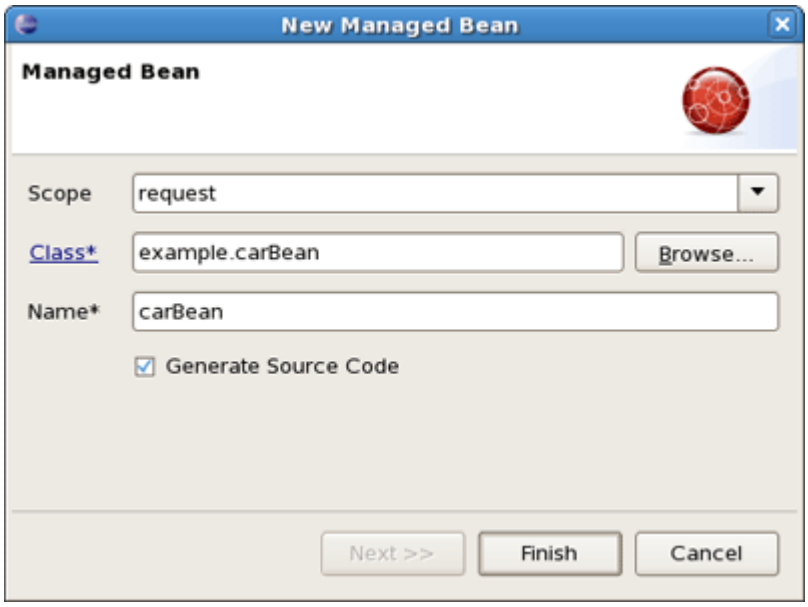

Figure 6.8. New Managed Bean Form

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 *Next* button will be activated. Pressing it you'll get *Managed Properties* dialog where all corresponding properties are displayed. Check the necessary ones to add them into your JSF Configuration File.

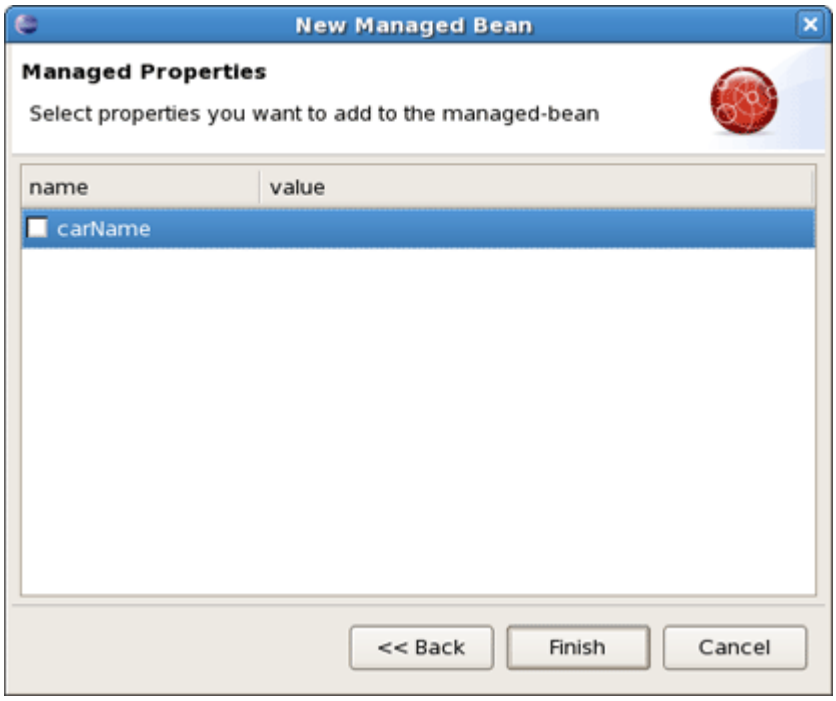

Figure 6.9. Selection of Bean's Properties.

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

Above-listed steps have demonstrated how you can specify an existing Bean in the JSF configuration file, i.e. *faces-config.xml*. In the next chapter you'll know how to organize and register another kind 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's necessary to go through the following steps:

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

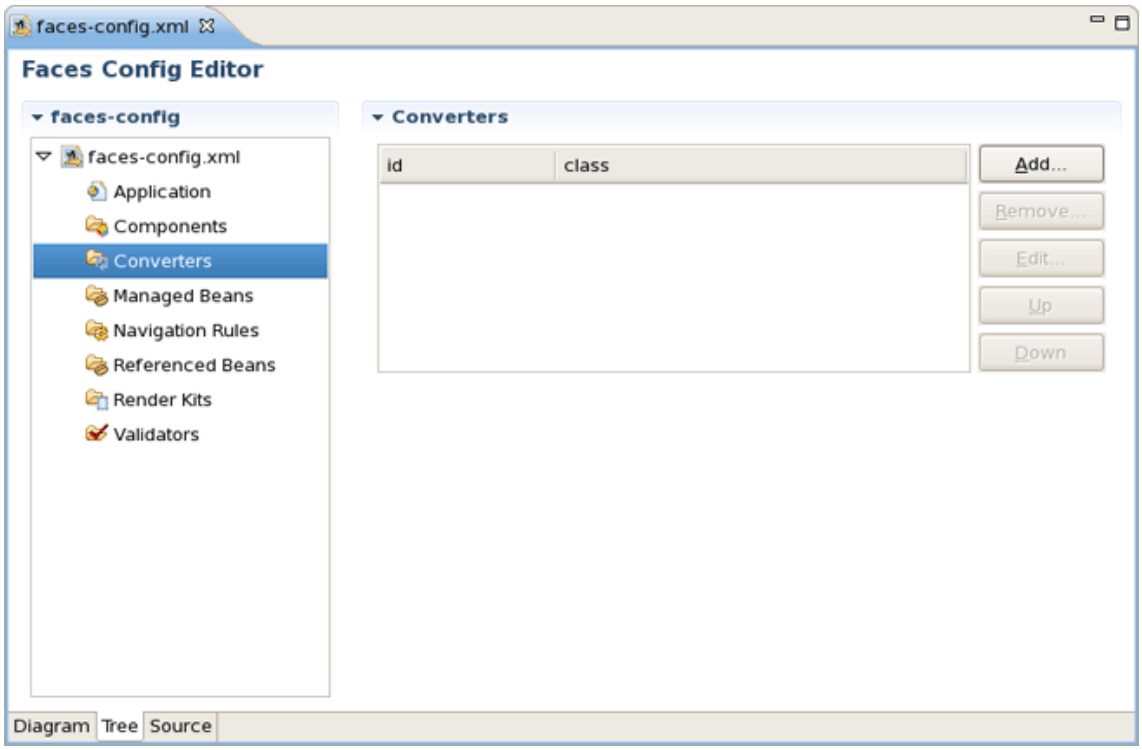

Figure 7.1. Converters

- Select *Converters* and click on *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.

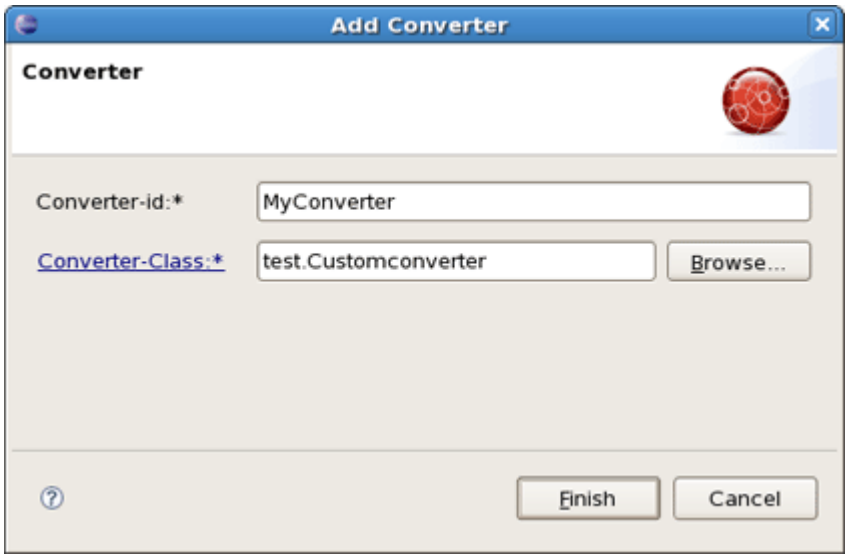

Figure 7.2. Add Converter Form

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

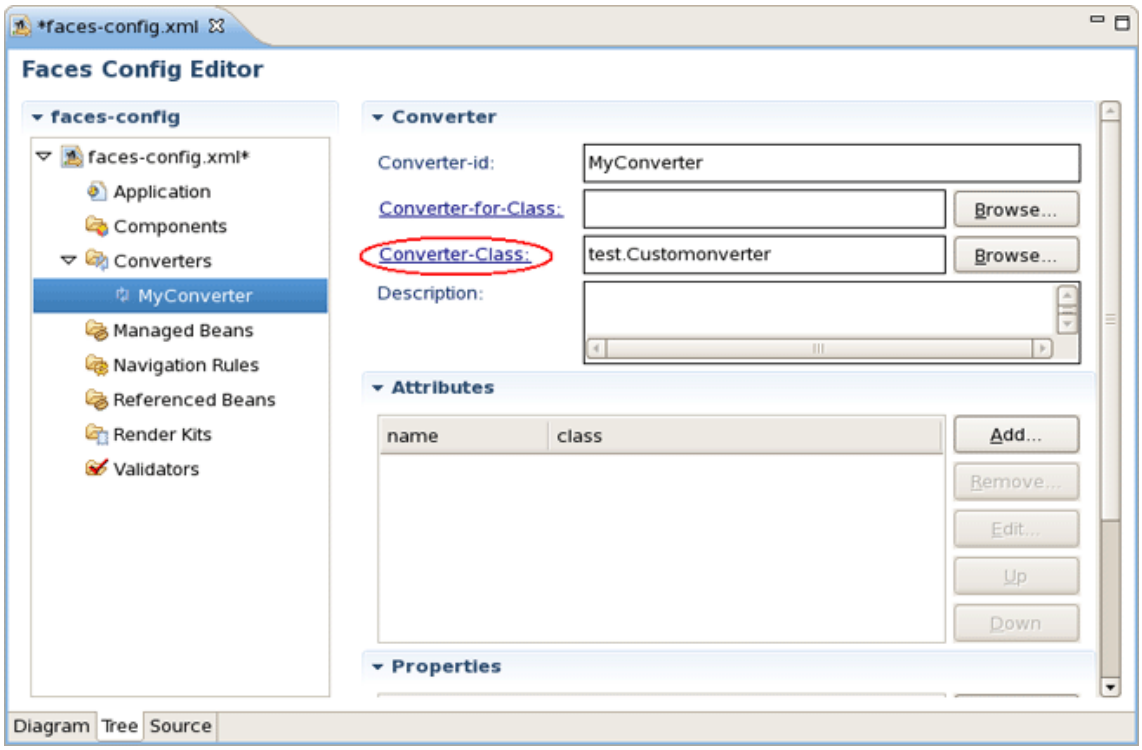

Figure 7.3. Generation of Source Code for Converter Class

- 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 *Finish*.

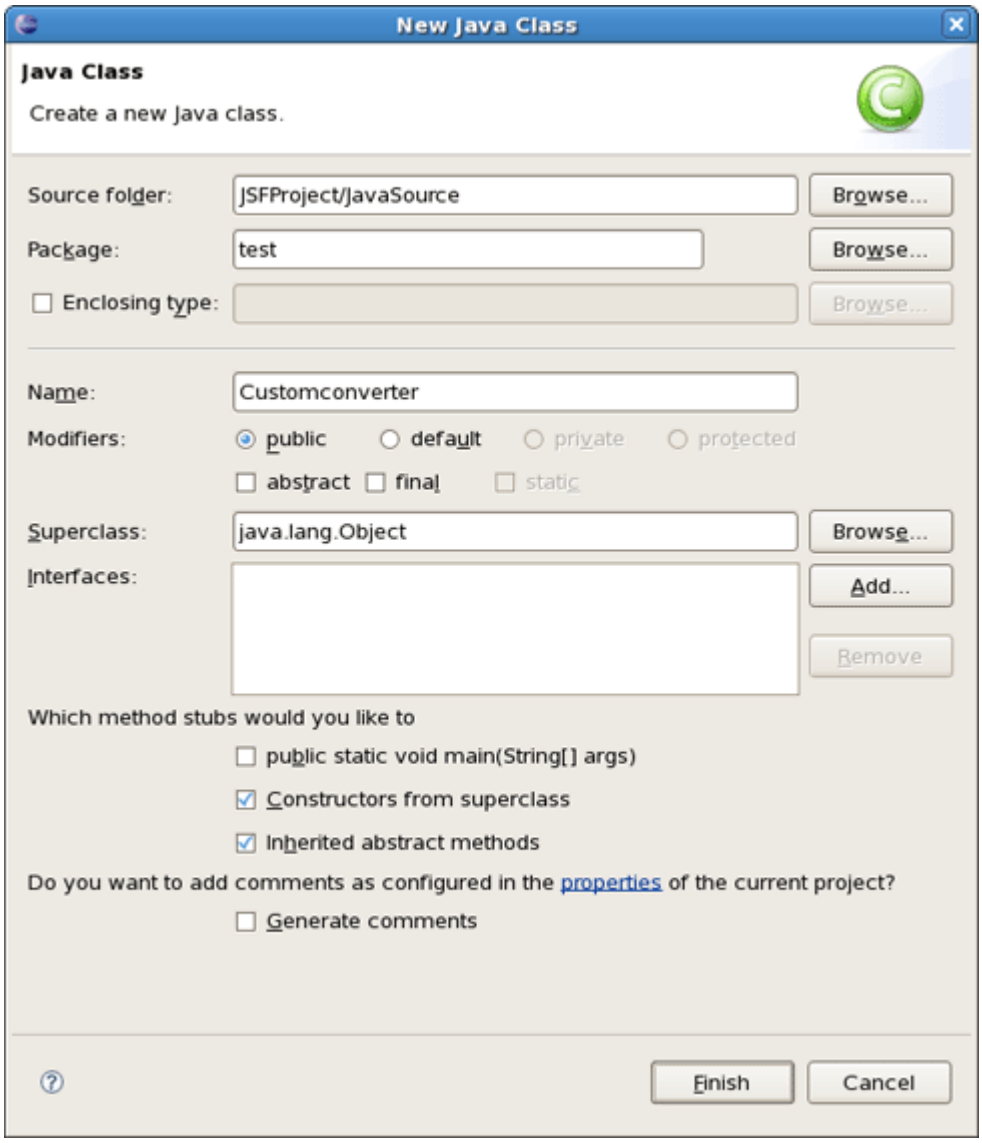

Figure 7.4. New Java Class Form

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

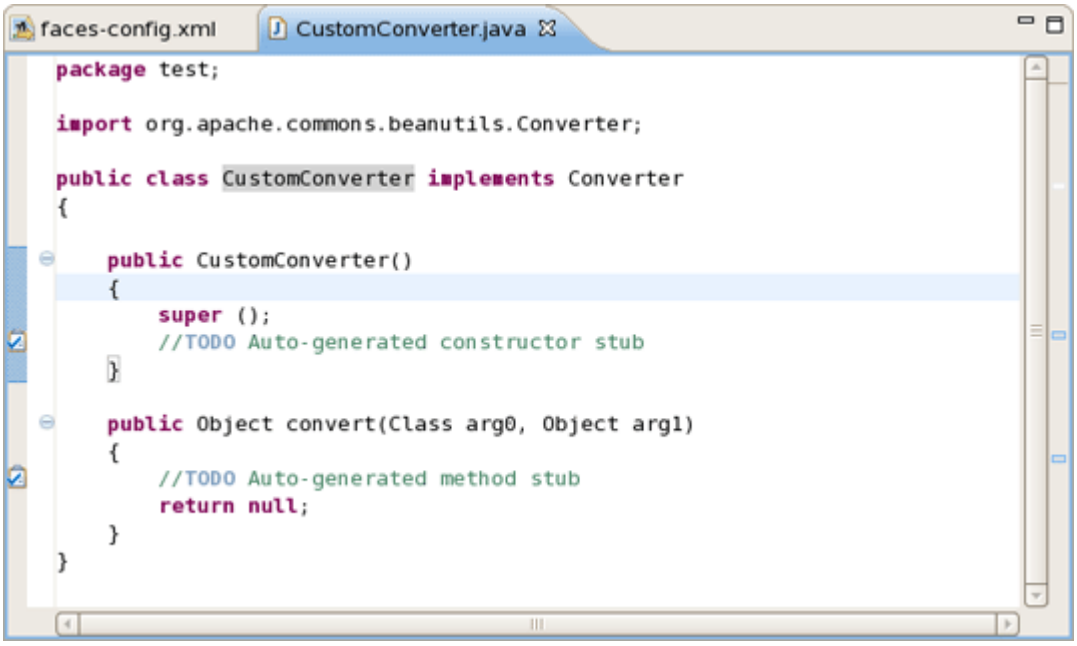

Figure 7.5. Converter Class

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

7.2. Create and Register a Custom Validator

It's also quite easy to develop your own custom Validators. You should perform the actions similar to the previous one. Go through the following steps:

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

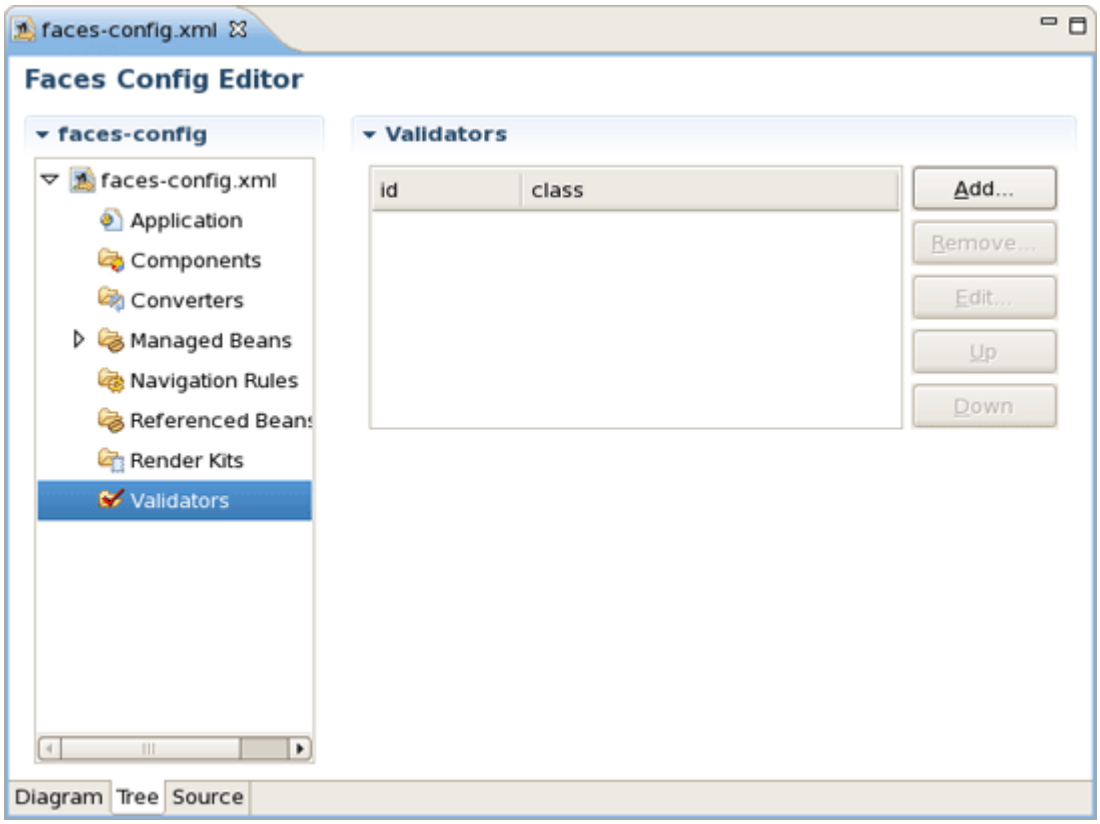

Figure 7.6. Validator in Faces Config Editor

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

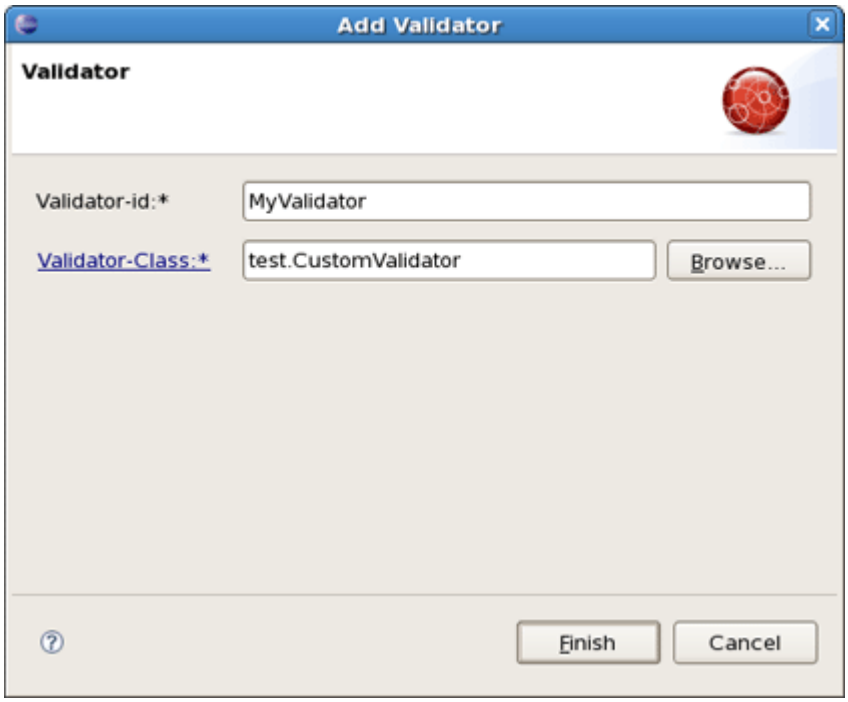

Figure 7.7. Adding Validator

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*.

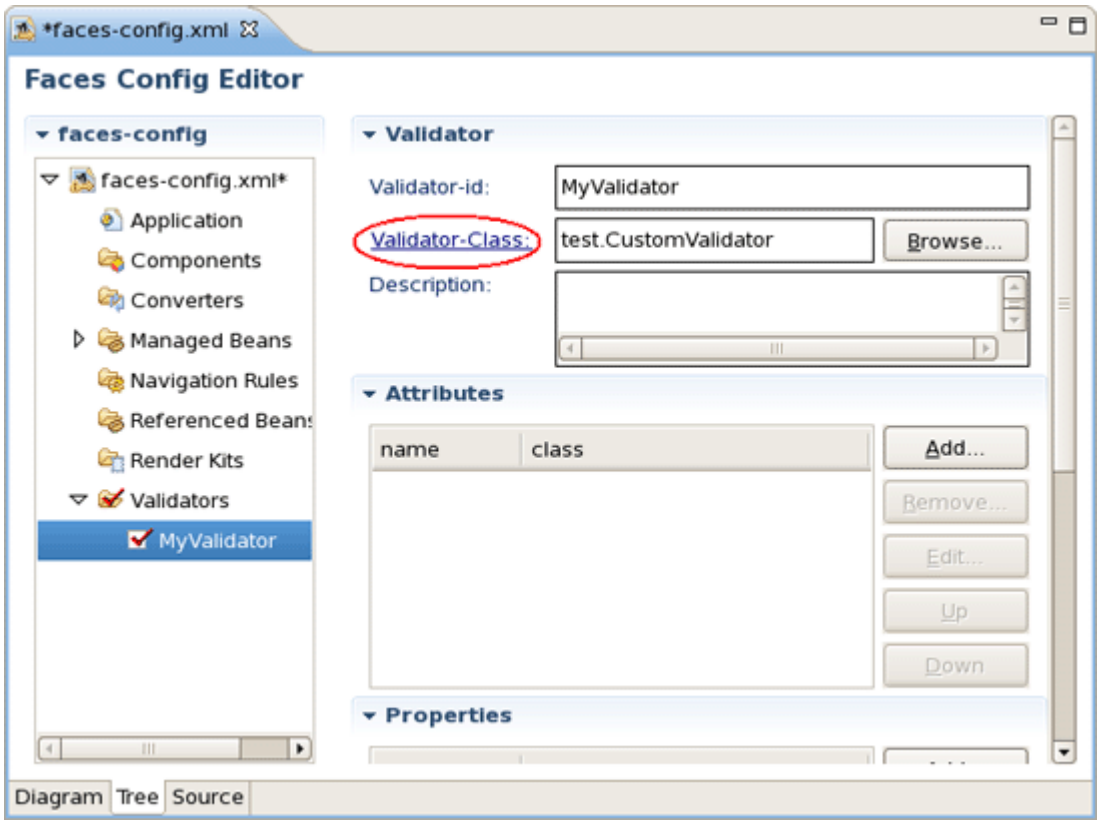

Figure 7.8. Creating Validator Class

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

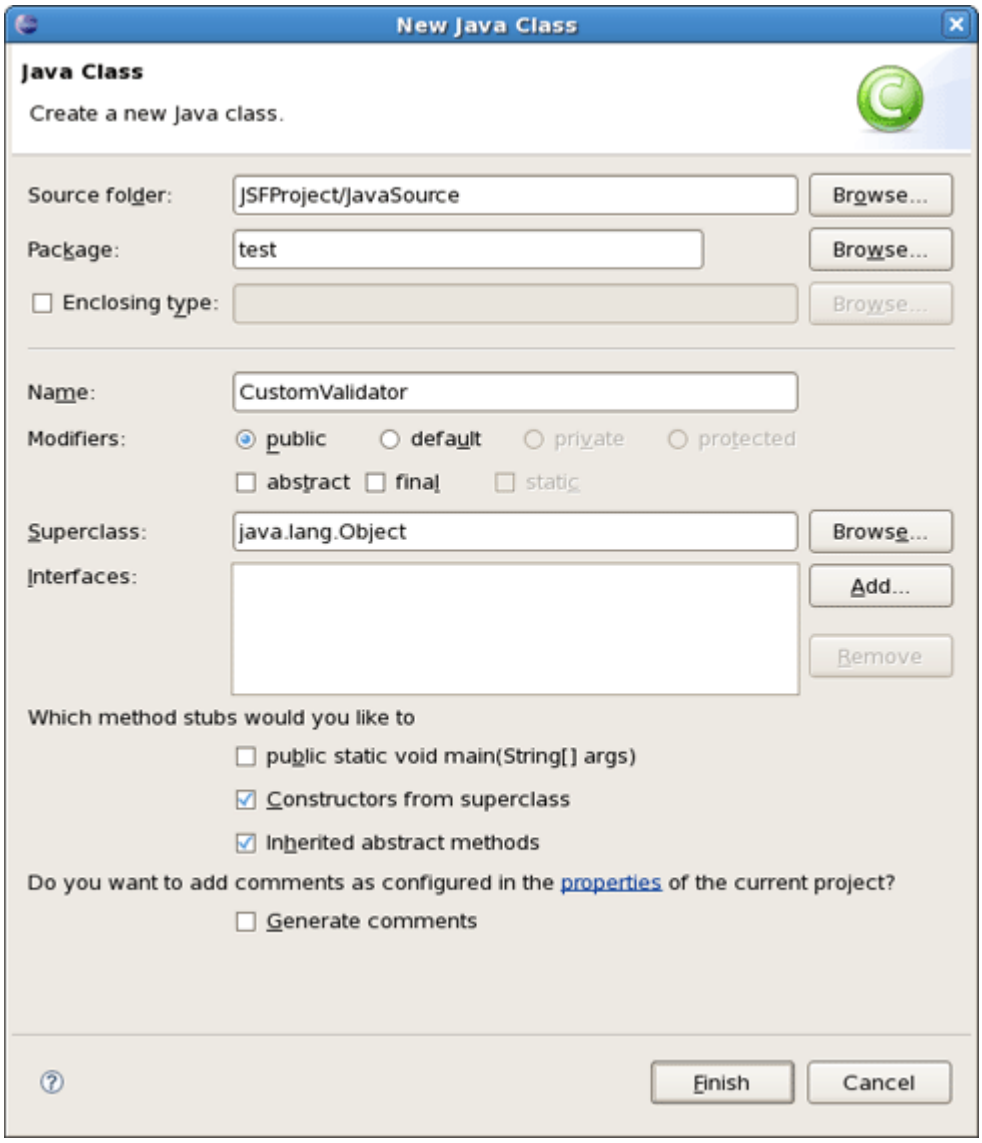

Figure 7.9. New Java Class Form

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

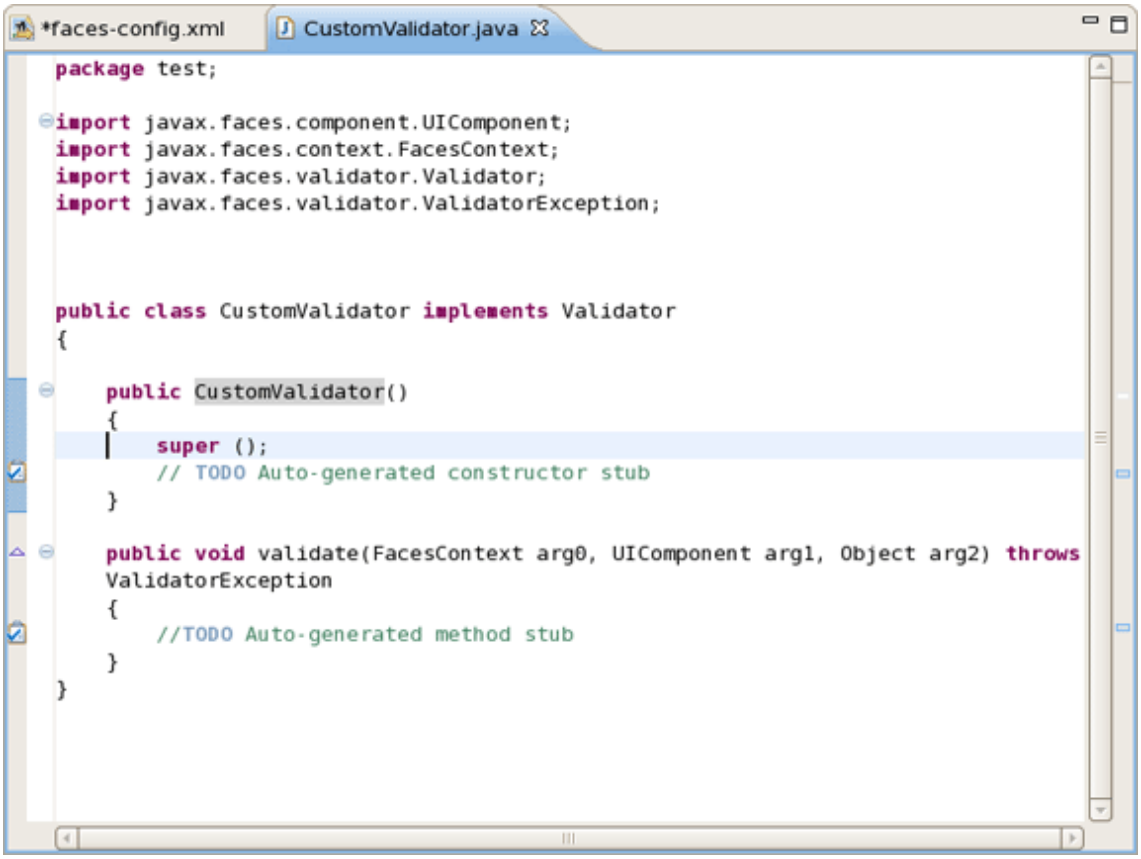

Figure 7.10. Converter Class Editing

7.3. Create and Register Referenced Beans

Creation of Referenced Beans is similar to creation of Custom Validator as well. To perform this, let's walk through the necessary steps.

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

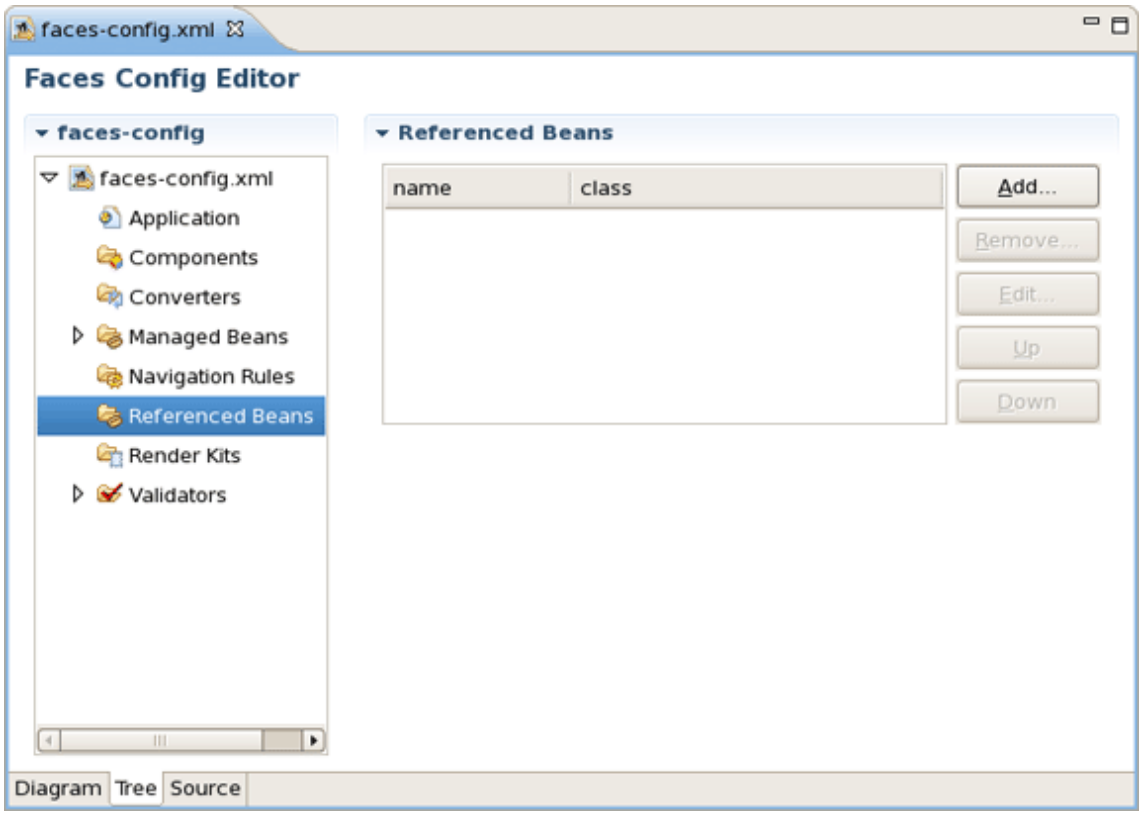

Figure 7.11. Referenced Beans in Faces Config Editor

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

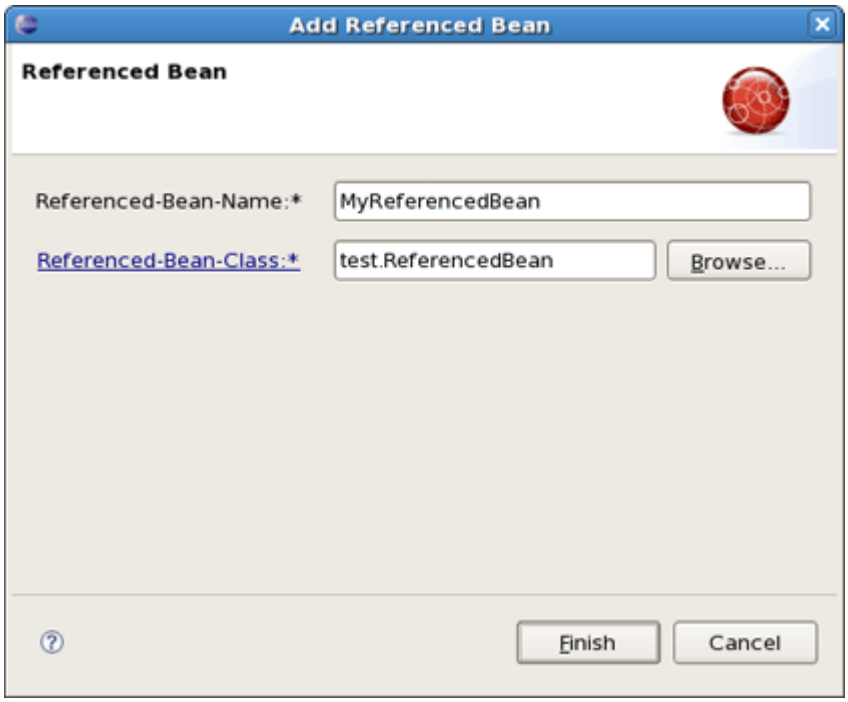

Figure 7.12. Add Referenced Bean

- 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.

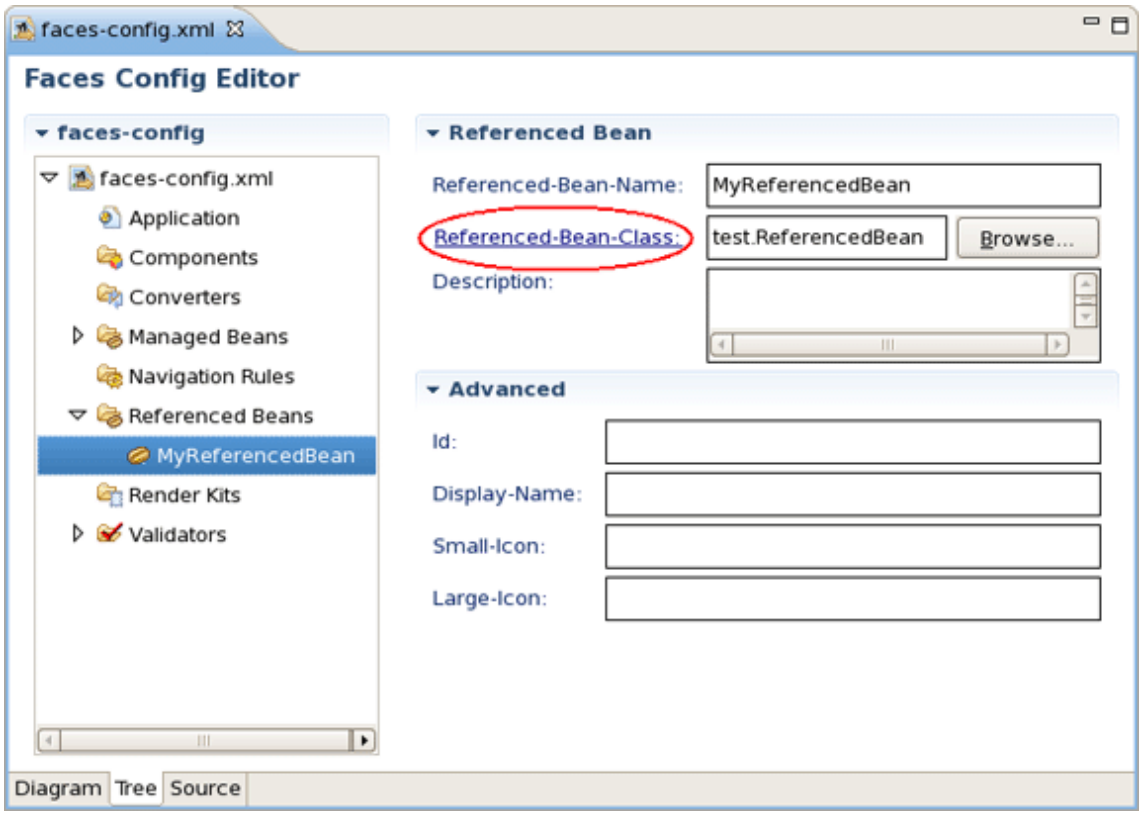

Figure 7.13. Create Referenced Bean Class

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

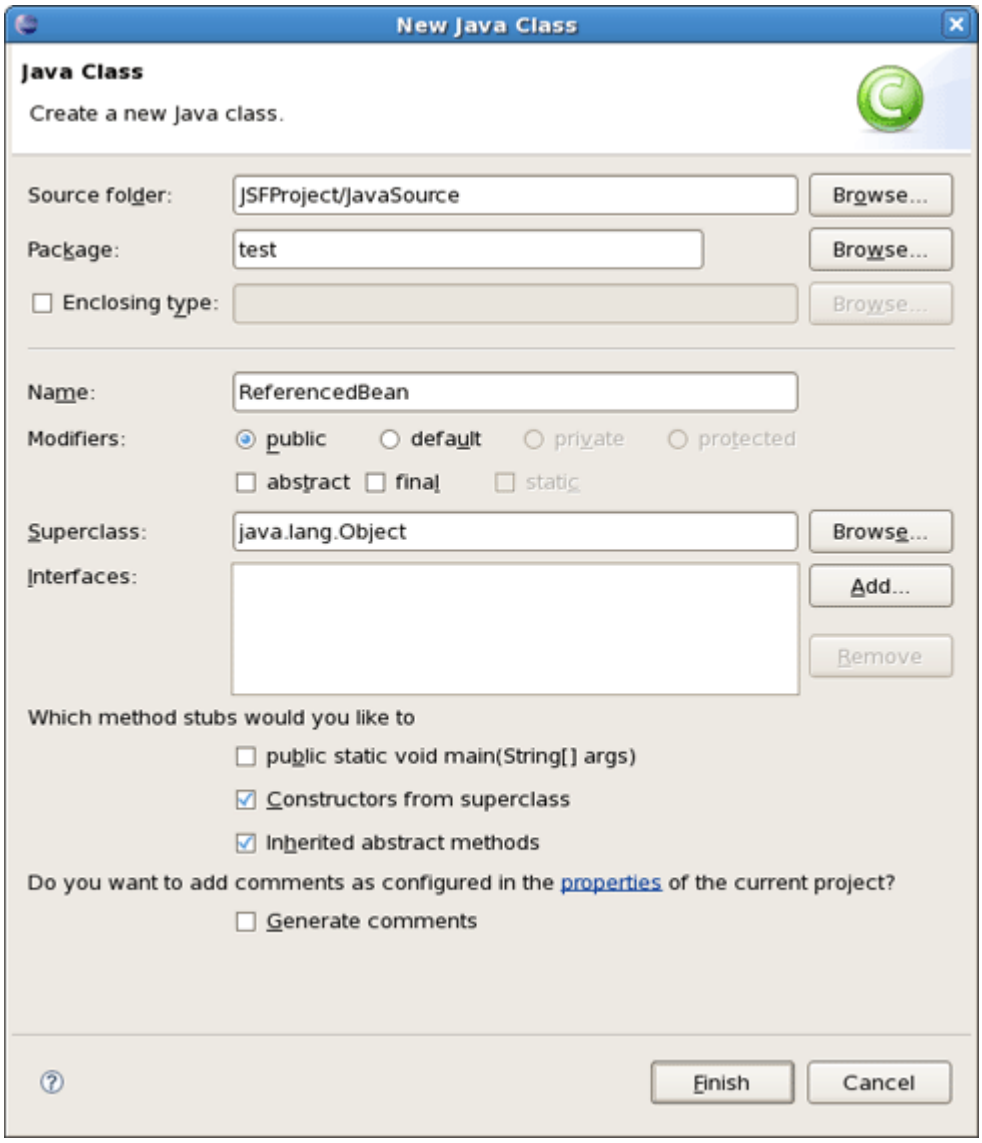

Figure 7.14. New Java Class Form

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

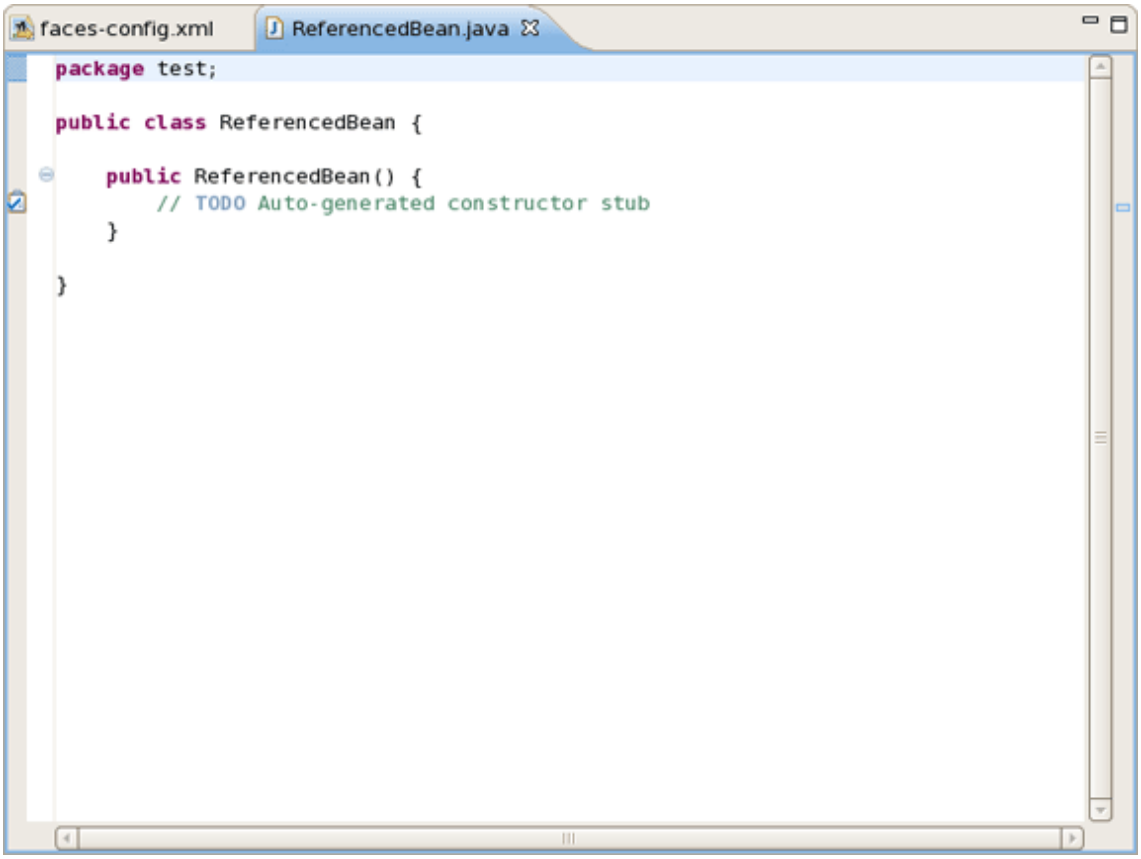

Figure 7.15. Referenced Bean Class Editing

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.

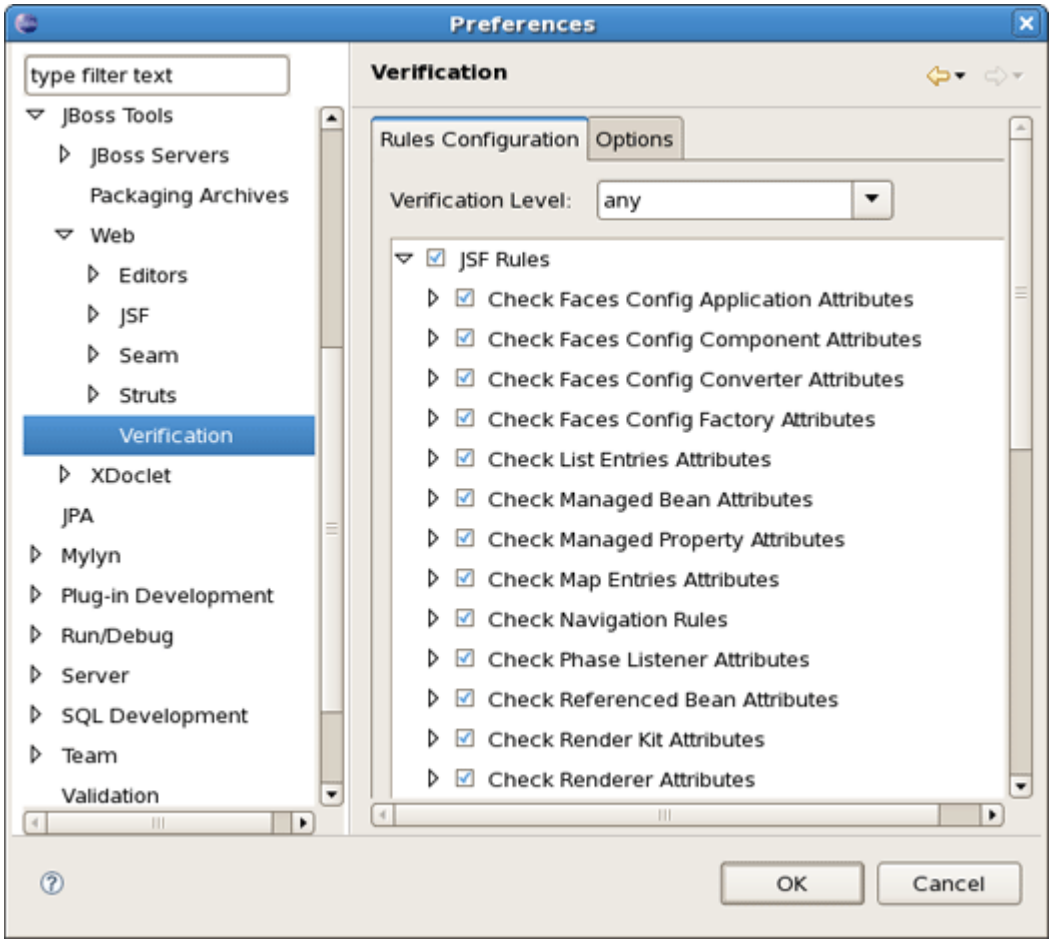

Figure 8.1. JSF Rules

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

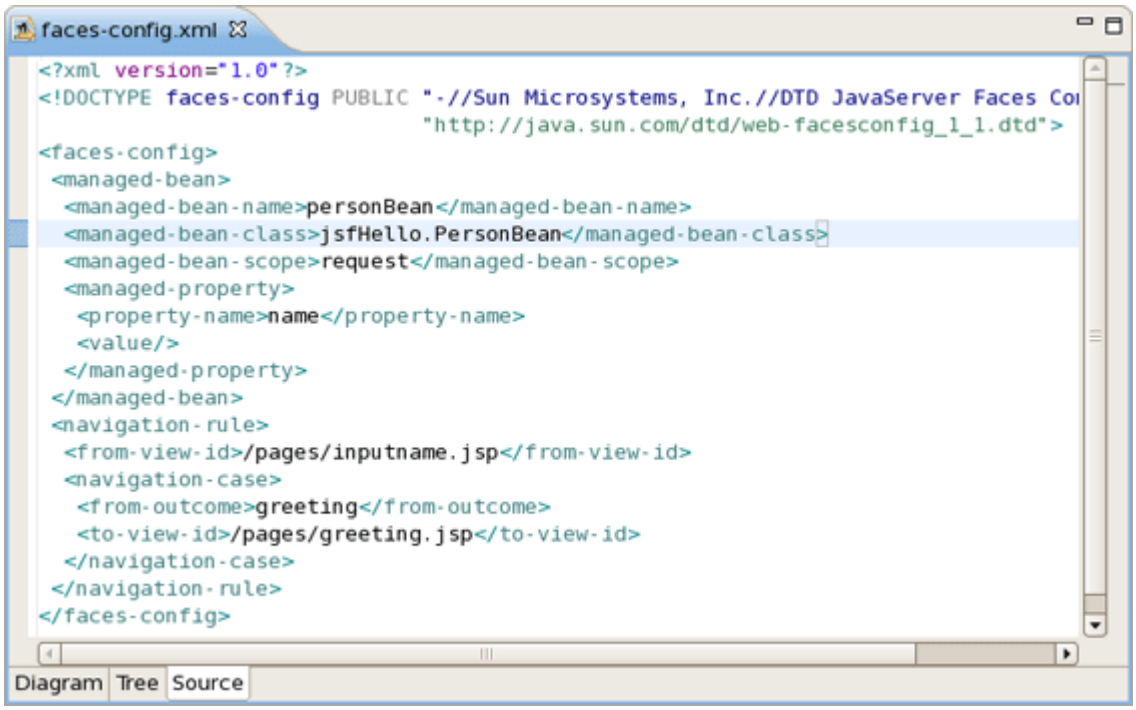

Figure 8.2. Faces-config.xml File

While typing a class name, you might make a minor typo (like *"jsfHello.PersonBean9"* instead of *"jsfHello.PersonBean"*). 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:

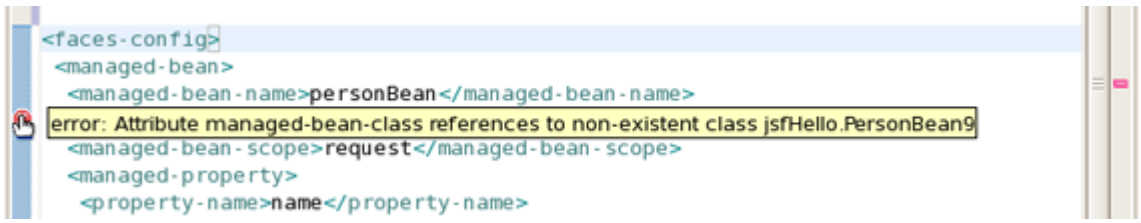

Figure 8.4. Error Message

Verification also checks navigation rules:

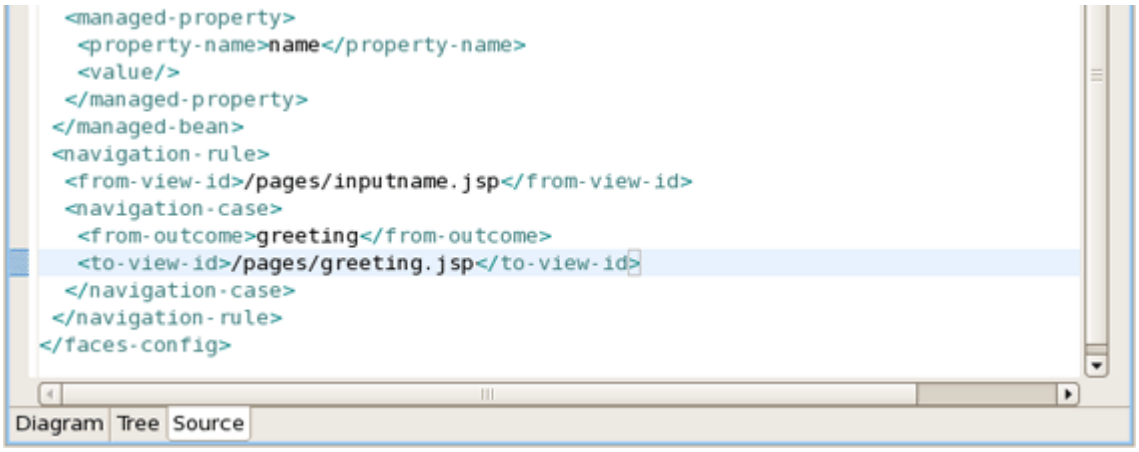

Figure 8.5. Checking Navigation Rules

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

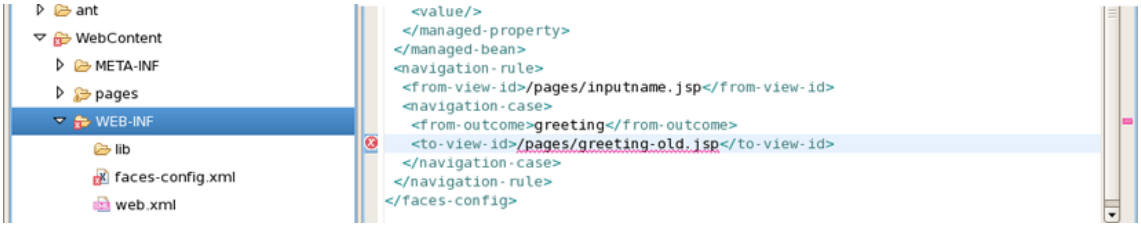

Figure 8.6. Page Name Verification

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.

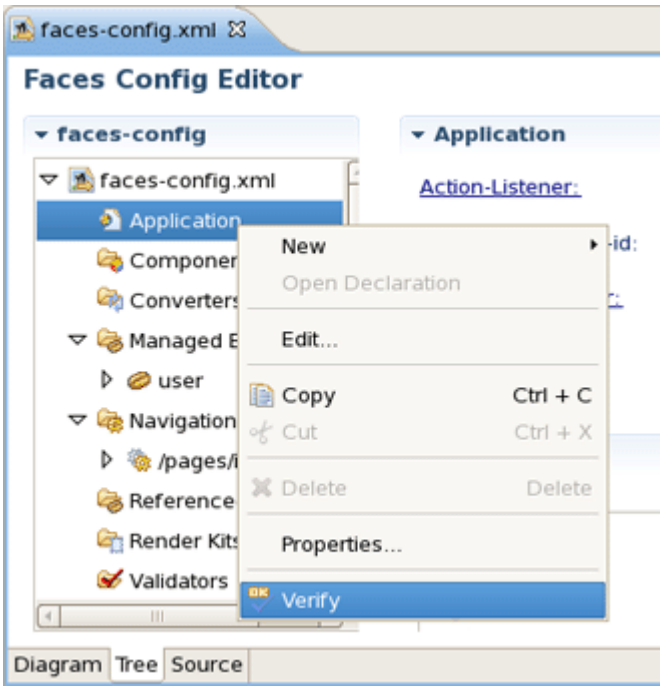

Figure 8.7. Verify Command

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].