Contents

Facelets Essentials: Guide to JavaServer™ Faces View Definition Framework

by Bruno Aranda and Zubin Wadia

Facelets is a templating language developed from the ground up with JavaServer™ Faces (JSF™) in mind. Because Facelets has come about as a result of many of the concerns with JavaServer™ Pages (JSP™) API when building JSF views, it steps outside of the JSP specification and provides a highly performant, JSF-centric view technology. Its top properties are templating, code reuse, and ease of development.

Focusing on these priorities allows Facelets to help make JSF suitable for large-scale projects. For example, one of the first things a Facelets developer finds is that the technology immediately leads to a reduction in user interface (UI) code. Take our advice: use Facelets in your applications instead of JSP. In this book, we will show you how to maximize your JSF productivity with Facelets by leveraging it the right contexts.

Facelets

When JavaServer Faces (JSF) was devised, the intention was to reuse JSP as the main technology to create pages, as it was already a standard in the web community. The idea was to simplify the adoption of JavaServer Faces by using a familiar tag language that already had a large adoption rate within the Java community.

Unfortunately, JSP and JSF don't naturally complement each other. JSP is used to create static or dynamic web content but not to create component trees. Its elements are processed in a page from top to bottom with the basic objective of creating a response to a request. JSF, however, has a more complex life cycle, and component generation and rendering happen in clearly separated phases. When used together, JSP and JSF both write output to the response, but they do so differently: the JSP container creates output as soon as it finds JSP content, whereas JSF components dictate its own rendering. This difference can cause some problems, as the following code snippet illustrates:

```
 <h:outputText value="I am first" />
I am second
```

In this snippet, we mix a JSF tag, the `h:outputText`, with free text (analyzed directly by the JSP container) just after it. As we expect, we get this output in the rendered page:

```
I am first
I am second
```

However, we can encapsulate our previous snippet in a panel (`h:panelGroup`) like so:

```
<h:panelGroup>
      <h:outputText value="I am first"/>
      I am second
</h:panelGroup>
```

If we do, the output is reversed:

```
I am second
I am first
```

We would naturally expect the second snippet to produce output like the first one. However, the JSP container adds the “I am second” text as soon as it encounters it, whereas the `h:panelGroup`, which is a component that renders its own children, won’t produce the “I am first” output until the closing tag is reached.

Problems similar to this can frustrate the developer using JSF for the first time who does not know the implementation details of both JSP and JSF. Facelets fills the gap between JSP and JSF. It is a view technology focused on building component trees and interweaving content with the complex JSF life cycle. Facelets replaces JSP with a very simple API that is a reflection of its simple principles, and it incorporates numerous developer-friendly features, which we describe in the next section.

Why Use Facelets

There are multiple reasons to use Facelets instead of JSP to create JSF pages. First, Facelets does not depend on a web container, so you can use JSF 1.2 without having to use JEE5 or a container that already has JSP 2.1. Facelets can work with any implementation and version of JSF.

Also, as described in the previous section, interweaving JSP and JSF can cause difficulties. In addition, JSTL cannot be used with JSF. Facelets provides a solution for this incompatibility while also providing a compilation process that is much faster than JSP’s, because no Java bytecode is actually generated and compiled behind the scenes when you first visit your page.

In addition, Facelets provides templating, so you can reuse your code extensively to simplify the development and maintenance of large-scale applications.

It allows the creation of lightweight components, which are quite trivial to develop compared to the JSF pure components. For example, you don't need to create tags for the UI components.

Then too, Facelets has Unified Expression Language (EL) support, including support for EL functions and compile-time EL validation. The Unified EL takes the features of the JSP EL and adds a few additional capabilities such as deferred expression evaluation, JSTL iteration tags, and method invocation within an expression definition. Facelets also provides precise error reporting, showing detailed information about the line where the exception occurs.

It is possible to use the `jsfc` attribute (which is the equivalent to the `jwcid` concept in Tapestry) to provide integration with existing HTML editors. In the following example, `jsfc` is used to tell the compiler to build a text field component (`h:inputText`) instead of just outputting the `input` tag:

```
<input id="myId" type="text"
jsfc="h:inputText"
value="#{bean.foo}"/>
```

Also, Facelets does not require any special render kits.

Creating an Application with Facelets

In this section, we are going to create a very simple application from scratch that uses Facelets. You will learn how to configure Facelets and get acquainted with some of the basic ideas behind the framework.

Facelets, like JSF, is based on standards and does not depend on any particular operating system or product.

Downloading Facelets

The Facelets project is hosted at Java.net (`http://facelets.dev.java.net/`). There, you can choose the download option that best meets your needs. You can go with the release binary and the sample applications bundled with it, or you can compile from source.

Release Binaries

You can download a release binary from `http://facelets.dev.java.net/servlets/ProjectDocumentList.` Once you've downloaded file, unzip the project into the folder of your choice.

The bundle includes the Facelets JAR at the root level (`jsf-facelets.jar`), and the dependencies can be found in the `lib` folder. Later, we will explain which dependencies Facelets needs.

A few demonstrations are also included; these can be used to start testing the framework. The WAR files for these demonstrations can be found at the root level and their sources in the `demo` folder. Some of the demonstrations follow:

- *starterkit:* A blank Facelets application that can be used when starting an empty project that uses Facelets
- *numberguess:* The traditional demonstration of JSF, where the user needs to guess a number, migrated to Facelets
- *hangman:* Demonstration of a migration from Tapestry to JSF and Facelets
- *portlet:* A small Facelets and MyFaces portlet example that supports the edit and help modes

CVS

You can also check out the source code using CVS if you are a member of Java.net. To do this, execute the following commands, replacing `USERNAME` with your Java.net login name:

```
cvs -d :pserver:USERNAME@cvs.dev.java.net:/cvs login
cvs -d :pserver:USERNAME@cvs.dev.java.net:/cvs \
    checkout facelets
```

You can find more information on how to check out Facelets from CVS at `http://facelets.dev.java.net/servlets/ProjectSource`.

Maven

The Facelets artifact can be found in the Maven Central Repository (`http://www.ibiblio.org/maven2/`) and in the Java.net Maven Repository (`https://maven-repository.dev.java.net/repository/`). You can include the artifact in your project by adding the following dependency to your Project Object Model (POM):

```
<dependency>
    <groupId>com.sun.facelets</groupId>
    <artifactId>jsf-facelets</artifactId>
    <version>1.1.13</version>
</dependency>
```

Version 1.1.13 is the current one at the time of this writing. You should use a newer version if possible.

Adding Dependencies

Facelets can work with any implementation and version of JSF. It also uses the recent EL API, and it can work with any version or implementation of this API as well. Table 1-1 summarizes the dependencies of Facelets.

Table 1-1. Facelets Dependencies

PROJECT	BUILD	INCLUDED	DESCRIPTION
Apache MyFaces	No	No	Implements JSF 1.1 and JSF 1.2.
JavaServer Faces RI	No	No	The reference implementations of JSF 1.1 and JSF 1.2 are available for use with your application.
JavaServer Faces API	Yes	Yes	JSF 1.2 API that works with the new EL specification. (Optionally, MyFaces Core API could be used.)
EL API	Yes	Yes	The stand-alone EL utilized by both JSP and JSF.
EL RI	No	Yes	The reference implementation that is used by Facelets for handling EL.
Servlet API	Yes	Yes	Servlet API
XML SAX	Yes	No	This dependency should not be an issue for most deployments, as it's a standard part of web containers and JREs.

Creating a Project Structure

Ensuring that your web application has the correct list of libraries included can be one of the trickiest parts when starting a project. Missing libraries or incompatibilities between its versions can lead to obscure exceptions and increase developer frustration. Moreover, the diversity of web containers, each of them with its own libraries and particularities, makes this task even more complex.

If we were to set up a project that uses Facelets and MyFaces, a common directory structure would look like this: