PRINTABLE VERSION

Getting Started with Java EE 6 Applications

This document provides a brief introduction to some of the features introduced as part of Java Enterprise Edition 6 (Java EE 6) specification. To illustrate the new features, this tutorial will demonstrate how to create a simple Java EE 6 web application that contains an EJB 3.1 stateless session bean facade for an entity class. You will use wizards in the IDE to generate the entity class and the session bean. The code generated by the wizard uses queries that are defined in the Criteria API that is part of JPA 2.0 and contained in the Java EE 6 specification. You will then create a named managed bean that accesses the session facade and a presentation layer that uses the Facelets view framework as specified in JSF 2.0.

This tutorial is based on the blog post Simplest Possible JSF 2 / EJB 3.1 / JPA Component - With WAR Deployment by Adam Bien. You can find additional Java EE examples at Adam Bien's Kenai project Java EE Patterns and Best Practices and in his book "Real World Java EE Patterns - Rethinking Best Practices", available at http://press.adam-bien.com.

This document uses the NetBeans IDE 6.8 Release.

Tutorial Exercises

- Creating the Web Application Project
- Creating the Entity Class and Session Facade
  - Creating the Entity Class
  - Creating the Session Facade
- Creating the JSF Managed Bean and JSF Pages
  - Creating the Managed Bean
  - Modifying the Index Page
  - Creating the Results Page
- Running the Project
- Downloading the Solution Project

To follow this tutorial, you need the following software and resources.

Software or Resource	Version Required
NetBeans IDE	6.8, Java version
Java Development Kit (JDK)	version 6
Sun GlassFish Enterprise Server	v3

Notes.

- GlassFish v3 requires Java Development Kit (JDK) 6.
- If you are using GlassFish v2 or an older version of NetBeans IDE, see Getting Started with Java EE Applications.

Prerequisites

This document assumes you have some basic knowledge of, or programming experience with, the following technologies:

- Java Programming
- NetBeans IDE

You can download a zip archive of the finished project.

Creating the Web Application Project

In this exercise you create a simple web application. When you create the web application you will specify the bundled GlassFish v3 application server as the target Java EE container. The GlassFish v3 application server is Java EE-compatible and includes the JSF 2.0 libraries that are required in this application.

In the New Project wizard you will choose Java EE 6 Web as the Java EE version. Java EE 6 Web is a lightweight Java EE 6 profile that contains a subset of the full Java EE 6 platform. The Java EE 6 Web profile is designed for web applications that do not require advanced Java EE 6 technologies such as support for remote interfaces, the full EJB 3.1 specification and the Java Message Service (JMS) API.

The Web profile supports the transaction processing and persistence management that are commonly used in enterprise web applications. You can use the Java EE Web profile for web applications that use session beans with a local interface or no interface. The full Java EE profile is required if the application uses a remote interface.

1. Choose File > New Project (Ctrl-Shift-N) from the main menu.
2. Select Web Application from the Java Web category and click Next.
3. Type **SimpleEE6App** for the project name and set the project location.
4. Deselect the Use Dedicated Folder option, if selected. Click Next.
   (For this tutorial there is little reason to copy project libraries to a dedicated folder because you will not need to share libraries with other users or projects.)
5. Set the server to GlassFish v3 and set the Java EE Version to Java EE 6 Web. Click Next.

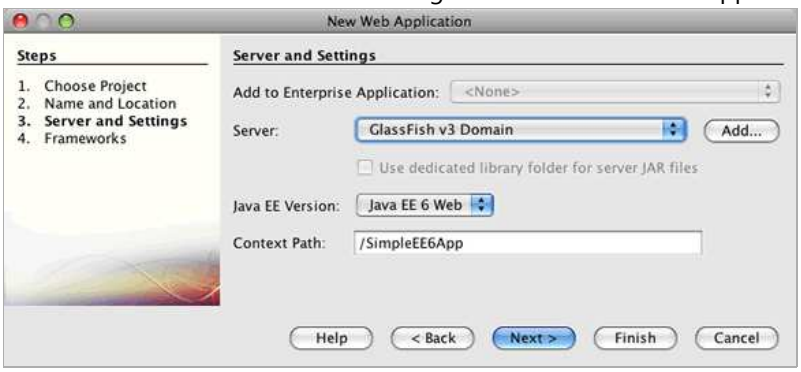
Download NetBeans IDE

Training
Java Programming Language
Developing Applications for the Java EE Platform

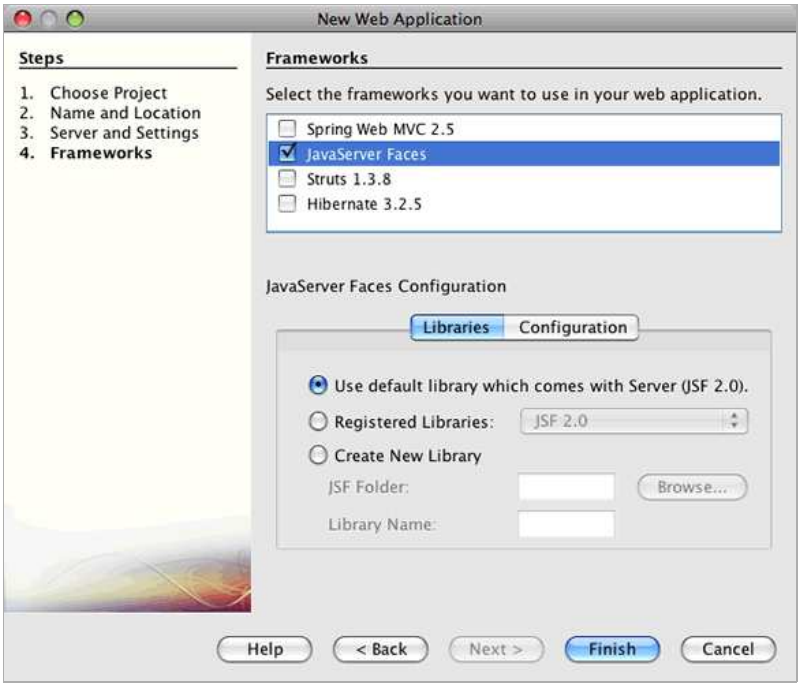
Support
Sun Developer Expert Assistance
Sun Software Service Plans for Developers

Documentation
General Java Development
External Tools and Services
Java and JavaFX GUIs
Java EE & Java Web Development
Web Services Applications
NetBeans Platform (RCP) and Module Development
PHP Applications
JavaScript and Dynamic Languages
C/C++ Applications
Mobile Applications
Sample Applications
Demos and Screencasts

More
FAQs
Contribute Documentation!
Docs for Earlier Releases



6. Select JavaServer Faces in the Frameworks pane. Click Finish.
   By default, the IDE will select the JSF 2.0 library if you are developing a Java EE 6 web application and deploying to GlassFish v3. The JSF 2.0 library enables you to use Facelets as the page language and also provides support for JSF 1.2 and JSP.



When you click Finish, the IDE creates the project and opens the project in the Projects window. The IDE automatically creates the JSF page `index.xhtml` and opens the page in the editor.

Creating the Entity Class and Session Facade

In this section you will create an entity class and a session facade for the entity class. An entity class is a plain old Java object (POJO), a simple Java class that is identified as an entity by the `@Entity` annotation. Starting with the Java EE 5 specification, you can use entity classes as persistent objects to represent tables in a database. The Java Persistence API enables you to use persistence in web applications without the need to create an EJB module.

The session facade for the entity class in this application is a stateless session bean. The Enterprise JavaBean (EJB) 3.1 architecture introduced as part of the Java EE 6 specification enables you to create session beans without the business interfaces that were required in EJB 3.0. The Java EE 6 specification also allows you to package EJB components directly in a WAR archive. This simplifies development of smaller web applications by eliminating the need to create separate EJB modules that are packaged as a JAR archive in an EAR archive. However, for larger enterprise applications that are distributed across different machines, you will still want to create EAR archives to separate your business logic from the presentation layer.

> For more about using EJB 3.1 in the IDE, see the tutorial Creating an Enterprise Application with EJB 3.1.
>
> For more details about entity classes, see the chapter Introduction to the Java Persistence API in the Java EE 6 Tutorial, Part I.
>
> For more information about session beans, see the chapter What is a Session Bean? in the Java EE 6 Tutorial, Part I.

Creating the Entity Class

In this exercise you will use the New Entity Class wizard to create a simple persistent entity class. You will also use the wizard to create a persistence unit that defines the data source and entity manager used in the application. You will add one field in the class to represent the data in your table and generate a getter and setter for the new field.

An entity class must have a primary key. When you create the entity class using the wizard, the IDE by default generates the field `id` and annotates the field with the `@Id` annotation to declare the field as the primary key. The IDE also adds the `@GeneratedValue` annotation and specifies the key generation strategy for the primary id field.

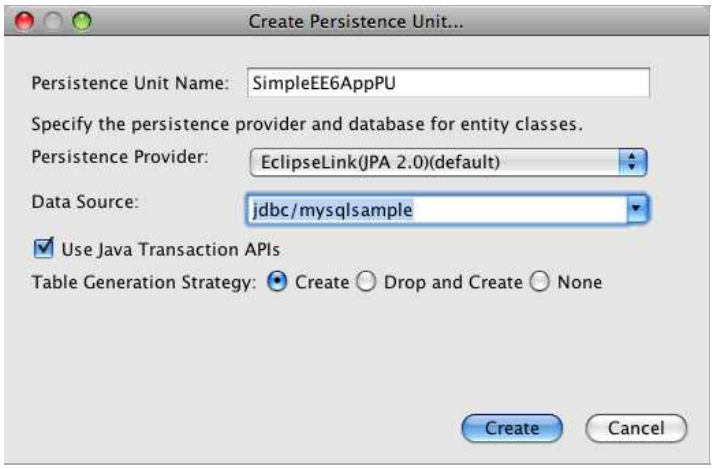
Using Java Persistence in your project greatly simplifies application development by removing the need for configuring deployment descriptors to provide object-relational mapping information for persistent fields or properties. Instead, you can use annotations to define these properties directly in a simple Java class.

Entity persistence is managed by the EntityManager API. The EntityManager API handles the persistence context, and each persistence context is a group of entity instances. When developing your application, you can use annotations in your class to

specify the persistent context instance of your entity instances. The life-cycle of the entity instances is then handled by the container.

To create the entity class, perform the following steps.

1. Right-click the project node and choose New > Other.
2. Select Entity Class from the Persistence category. Click Next.
3. Type **Message** for the Class Name.
4. Type **entities** for the Package.
5. Click Create Persistence Unit.
6. Select a datasource (for example, select `jdbc/sample` if you want to use JavaDB).
   The datasource for `jdbc/sample` is bundled with the IDE when you install the IDE and GlassFish, but you can specify a different datasource if you want to use a different database.
   You can keep the other default options (persistence unit name, EclipseLink persistence provider). Check that the persistence unit is using the Java Transaction API and that the Table Generation Strategy is set to Create so that the tables based on your entity classes are created when the application is deployed.
7. Click Create in the New Persistence Unit wizard.



8. Click Finish in the New Entity Class wizard.
   When you click Finish, the IDE creates the entity class and opens the class in the editor. You can see that the IDE generated the id field `private Long id;` and annotated the field with `@Id` and `@GeneratedValue(strategy = GenerationType.AUTO)`.
9. In the editor, add the `message` field (in bold) below the `id` field.

```
private Long id;
private String message;
```

10. Right-click in the editor and choose Insert Code (Ctrl+I) and then select Getter and Setter.
11. In the Generate Getters and Setters dialog box, select the `message` field and click Generate.
    The IDE generates getter and setter methods for the field `message`.

12. Save your changes.

The entity class represents a table in the database. When you run this application, a database table for Message will be automatically created. The table will contain the columns `id` and `message`.

If you look at the persistence unit in the XML editor, you can see that the application will use the Java Transaction API (JTA) (`transaction-type="JTA"`). This specifies that the responsibility for managing the lifecycle of entities in the persistence context is assigned to the container. This results in less code because the entity lifecycle is managed by the container and not by the application. For more about using JTA to manage transactions, see the Java Transaction API documentation.

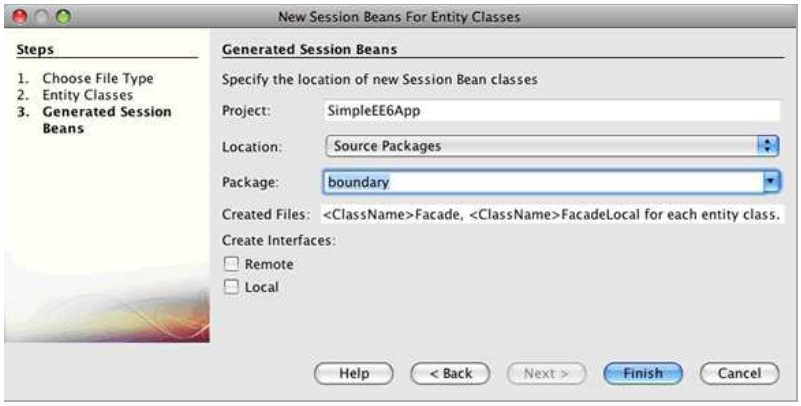
Creating the Session Facade

In this exercise you will use a wizard to create a stateless session facade for the Message entity. The EJB 3.1. specification states that business interfaces for session beans are now optional. In this application where the client accessing the bean is a local client, you have the option to use a local interface or a no-interface view to expose the bean.

To create the session bean, perform the following steps.

1. Right-click the project node and choose New > Other.
2. Select Session Beans for Entity Classes from the Java EE category. Click Next.

3. Select the `Message` entity and click Add. Click Next.
4. Type **boundary** for the package. Click Finish.
   Notice that you did not need to create a business interface for the session bean. Instead, in this application the bean will be exposed to a local managed bean using a no-interface view.



When you click Finish, the session facade class `MessageFacade.java` is created and opens in the Source Editor. The bean class contains the business logic and manages the EntityManager. As you can see in the generated code, the annotation `@Stateless` is used to declare the class as a stateless session bean component.

```
@Stateless
public class MessageFacade {
    @PersistenceContext(unitName = "SimpleEE6AppPU")
    private EntityManager em;
```

When you create the facade for the entity using the wizard, by default the IDE adds the `PersistenceContext` annotation (`@PersistenceContext(unitName = "SimpleEE6AppPU")`) to inject the entity manager resource into the session bean component and to specify the name of the persistence unit. In this example the name of the persistence unit is declared explicitly, but the name is optional if the application has only one persistence unit.

The IDE also generates methods in the facade to create, edit, remove and find entities. The EntityManager API defines the methods that are used to interact with the persistence context. You can see that IDE generates some commonly used default query methods that can be used to find entity objects.

```
public List<Message> findAll() {
    CriteriaQuery cq = em.getCriteriaBuilder().createQuery();
    cq.select(cq.from(Message.class));
    return em.createQuery(cq).getResultList();
}

public List<Message> findRange(int[] range) {
    CriteriaQuery cq = em.getCriteriaBuilder().createQuery();
    cq.select(cq.from(Message.class));
    Query q = em.createQuery(cq);
    q.setMaxResults(range[1] - range[0]);
    q.setFirstResult(range[0]);
    return q.getResultList();
}

public int count() {
    CriteriaQuery cq = em.getCriteriaBuilder().createQuery();
    Root<Message> rt = cq.from(Message.class);
    cq.select(em.getCriteriaBuilder().count(rt));
    Query q = em.createQuery(cq);
    return ((Long) q.getSingleResult()).intValue();
}
```

The `findAll`, `findRange` and `count` methods use methods defined in the Criteria API for creating queries. The Criteria API is part of the JPA 2.0 specification that is included in the Java EE 6 specification.

Creating the JSF Managed Bean and JSF Pages

In this section you will create the presentation layer for the application using JavaServer Faces (JSF) 2.0 and a managed backing bean that is used by the JSF pages. The JSF 2.0 specification adds support for Facelets as the preferred view technology for JSF-based applications. Starting with JSF 2.0, you can also use the `@ManagedBean` annotation in your source code to declare a class a managed bean. You are no longer required to add entries in the `faces-config.xml` file to declare JSF managed beans. You can use bean names in JSF pages to access methods in the managed bean.

> For more about IDE support for the JavaServer Faces 2.0 specification, see JSF 2.0 Support in NetBeans IDE 6.8.
>
> For more about the JavaServer Faces 2.0 specification, see the JavaServer Faces Technology chapter in the Java EE 6 Tutorial, Volume I.

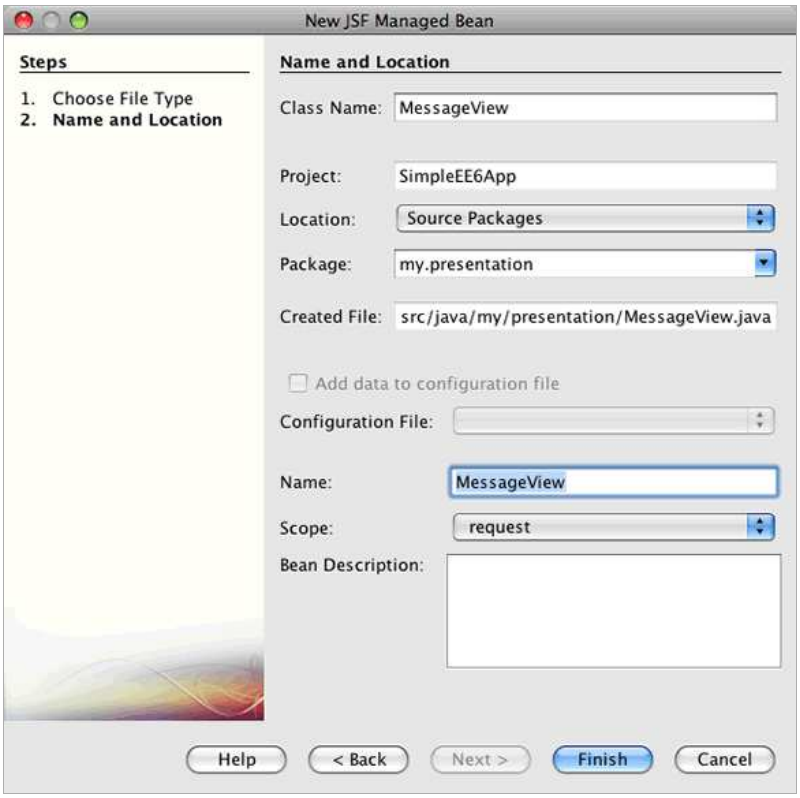
Creating the Managed Bean

In this exercise you will create a simple JSF managed bean that is used to access the session facade. The JSF 2.0 specification that is part of Java EE 6 enables you to use annotations in the bean class to identify the class as a JSF managed bean, to specify

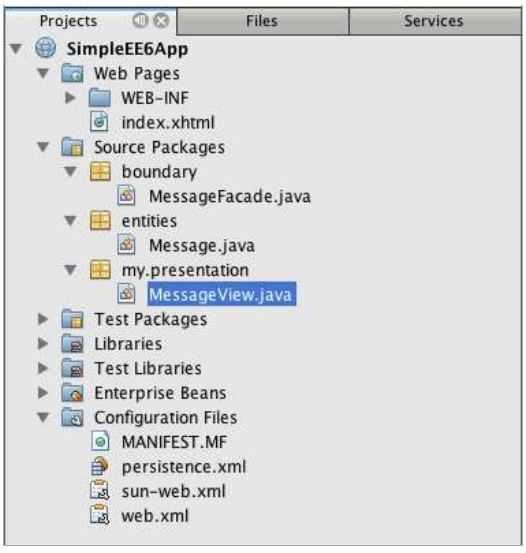
the scope and to specify a name for the bean.

To create the managed bean, perform the following steps.

1. Right-click the project node and choose New > Other.
2. Select JSF Managed Bean from the JavaServer Faces category. Click Next.
3. Type **MessageView** for the Class Name.
   You will use the Managed Bean name `MessageView` as the value for the `inputText` and `commandButton` in the JSF page `index.xhtml` when calling methods in the bean.
4. Type **my.presentation** for the Package.
5. Type **MessageView** for the Name that will be used for the managed bean.
   When you create the managed bean using the wizard, the IDE will by default assign a name to the bean based on the name of the bean class.
6. Set Scope to Request. Click Finish.



When you click Finish, the IDE creates the bean class and opens the class in the editor. In the Projects window you will see the following files.



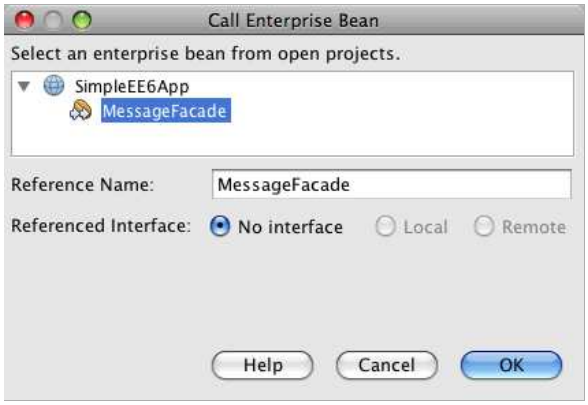
In the editor, you can see that the IDE added the `@ManagedBean` and `@RequestScoped` annotations and the name of the bean.

```
@ManagedBean(name="MessageView")
@RequestScoped
public class MessageView {

    /** Creates a new instance of MessageView */
    public MessageView() {
```

```
    }

}
```

You will now add an `@EJB` annotation to use dependency injection to obtain a reference to the MessageFacade session bean. You will also call the `findAll` and `create` methods that are exposed in the session facade. The IDE's code completion can help you when typing the methods.

1. Right-click in the editor and choose Insert Code (Ctrl+I) to open the popup menu.
2. Choose Call Enterprise Bean in the popup menu.
3. Select MessageFacade in the Call Enterprise Bean dialog box. Click OK.



When you click OK, the IDE adds the following code (in bold) to inject the bean.

```
public class MessageView {

    /** Creates a new instance of MessageView */
    public MessageView() {
    }

    // Injects the MessageFacade session bean using the @EJB annotation
    @EJB
    MessageFacade messageFacade;
}
```

4. Add the following code to the class.

```
    // Creates a new field
    private Message message;

    // Creates a new instance of Message
    public MessageView() {
       this.message = new Message();
    }

    // Calls getMessage to retrieve the message
    public Message getMessage() {
       return message;
    }

    // Returns the total number of messages
    public int getNumberOfMessages(){
       return messageFacade.findAll().size();
    }

    // Saves the message and then returns the string "theend"
    public String postMessage(){
       this.messageFacade.create(message);
       return "theend";
    }
```

5. Fix your imports (Ctrl-Shift-I) and save your changes.

   You can use the code completion in the editor to help you type your code.

Notice that the `postMessage` method returns the string "theend". The JSF 2.0 specification enables the use of implicit navigation rules in applications that use Facelets technology. In this application, no navigation rules are configured in `faces-config.xml`. Instead, the navigation handler will try to locate a suitable page in the application. In this case, the navigation handler will try to locate a page named `theend.xhtml` when the `postMessage` method is invoked.

Modifying the Index Page

In this exercise you will make some simple changes to the `index.xhtml` page to add some UI components. You will add a form with an input text field and a button.

1. Open `index.xhtml` in the editor.
2. Modify the file to add the following simple form between the `<h:body>` tags.

```
<h:body>
    <f:view>
        <h:form>
            <h:outputLabel value="Message:"/><h:inputText value="#{MessageView.message.message}"/>
            <h:commandButton action="#{MessageView.postMessage}" value="Post Message"/>
        </h:form>
    </f:view>
</h:body>
```

The JSF code completion can help you when you type the code.

3. Save your changes.

The `inputText` and `commandButton` components will invoke the methods in the named JSF managed bean `MessageView`. The `postMessage` method will return "theend", and the navigation handler will look for a page named `theend.xhtml`.

Creating the Results Page

In this exercise you will create the JSF page `theend.xhtml`. The page will be displayed when the user clicks the Post Message button in `index.xhtml` and invokes the `postMessage` method in the JSF managed bean.

1. Right-click the project node and choose New > Other.
2. Select JSF Page from the JavaServer Faces category. Click Next.
3. Type **theend** as the File Name.
4. Make sure that the Facelets option is selected. Click Finish.
5. Modify the file by typing the following between the <h:body> tags.

```
<h:body>
    <h:outputLabel value="Thanks! There are "/>
    <h:outputText value="#{MessageView.numberOfMessages}"/>
    <h:outputLabel value=" messages!"/>
</h:body>
```

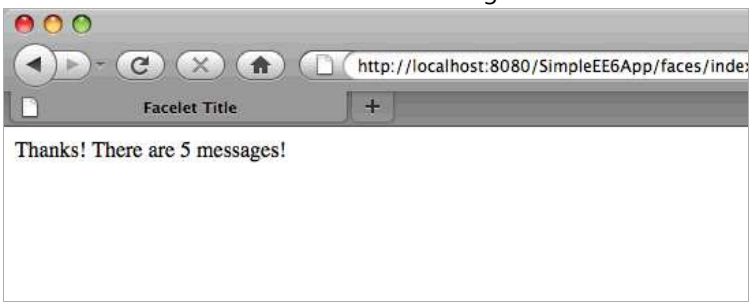
When you start typing, the IDE automatically adds the `xmlns:h="http://java.sun.com/jsf/html` tag library definition to the file for the JSF elements.

Running the Application

You are now finished coding the application. You can now test the application in your browser.

1. Right-click the project node in the Projects window and choose Run.
   When you choose Run, the IDE builds and deploys the application and opens `index.xhtml` in your browser.
2. Type a message in the text field. Click Post Message.

When you click Post Message, the message is saved to the database and the number of messages is retrieved and displayed.



Downloading the Solution Project

You can download the sample projects used in this tutorial in the following ways.

- Download a zip archive of the finished project.
- Checkout the project sources from Kenai by performing the following steps:
    1. Choose Team > Kenai > Get Sources from Kenai from the main menu.
    2. In the Get Sources from Kenai dialog box, locate the Kenai Repository by clicking Browse to open the Browse Kenai Projects dialog box.
    3. Search for the **NetBeans IDE Samples Catalog**.
    4. Select the NetBeans IDE Samples Catalog and click OK.
    5. Click Browse to specify the Folder to Get and select **Samples/JavaEE/SimpleEE6App**. Click OK.
    6. Specify the Local Folder for the sources (the local folder must be empty).
    7. Click Get From Kenai.
    When you click Get From Kenai, the IDE initializes the local folder as a Subversion repository and checks out the project sources.
    8. Click Open Project in the dialog that appears when checkout is complete.

    Notes.

    - Steps for checking out sources from Kenai only apply to NetBeans IDE 6.8.
    - You need a Subversion client to checkout the sources from Kenai. For more about installing Subversion, see the section on Setting up Subversion in the Guide to Subversion in NetBeans IDE.

Send Us Your Feedback

See Also

For more information about using NetBeans IDE to develop Java EE applications, see the following resources:

- Introduction to Java EE Technology
- JSF 2.0 Support in NetBeans IDE 6.8
- Java EE & Java Web Learning Trail

You can find more information about using Java EE 6 technologies to develop applications in the Java EE 6 Tutorial.

To send comments and suggestions, get support, and keep informed on the latest developments on the NetBeans IDE Java EE development features, join the nbj2ee mailing list.