How To... Call An EJB From JSF Using Data Tables

Applicable Releases:
SAP NetWeaver Composition Environment 7.1

Topic Area:
User Productivity
Development and Composition

Capability:
User Interface Technology
Java

Version 1.0
October 2008
## Document History

<table>
<thead>
<tr>
<th>Document Version</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.00</td>
<td>First official release of this guide</td>
</tr>
</tbody>
</table>
### Typographic Conventions

<table>
<thead>
<tr>
<th>Type Style</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>Example Text</em></td>
<td>Words or characters quoted from the screen. These include field names, screen titles, pushbuttons labels, menu names, menu paths, and menu options. Cross-references to other documentation</td>
</tr>
<tr>
<td><em>Example text</em></td>
<td>Emphasized words or phrases in body text, graphic titles, and table titles</td>
</tr>
<tr>
<td><em>Example text</em></td>
<td>File and directory names and their paths, messages, names of variables and parameters, source text, and names of installation, upgrade and database tools.</td>
</tr>
<tr>
<td><em>Example text</em></td>
<td>User entry texts. These are words or characters that you enter in the system exactly as they appear in the documentation.</td>
</tr>
<tr>
<td><code>&lt;Example text&gt;</code></td>
<td>Variable user entry. Angle brackets indicate that you replace these words and characters with appropriate entries to make entries in the system.</td>
</tr>
<tr>
<td><strong>EXAMPLE TEXT</strong></td>
<td>Keys on the keyboard, for example, F2 or ENTER.---------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>

### Icons

<table>
<thead>
<tr>
<th>Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>🔄</td>
<td>Caution</td>
</tr>
<tr>
<td>📘</td>
<td>Note or Important</td>
</tr>
<tr>
<td>📖</td>
<td>Example</td>
</tr>
<tr>
<td>⚡️</td>
<td>Recommendation or Tip</td>
</tr>
</tbody>
</table>
# Table of Contents

1. Business Scenario ............................................................................................................... 1  
2. Background Information ..................................................................................................... 1  
3. Prerequisites ........................................................................................................................ 1  
4. Step-by-Step Procedure ...................................................................................................... 2  
   4.1 Tutorial Setup ............................................................................................................... 2  
   4.2 Implement the EJB Module .......................................................................................... 3  
      4.2.1 Create the JavaBean ....................................................................................... 3  
      4.2.2 Create the SessionBean .................................................................................. 4  
      4.2.3 Add SessionBean methods to local interface .............................................. 7  
   4.3 Implement the Web Module .......................................................................................... 8  
      4.3.1 Create a JSF Managed bean ........................................................................... 8  
      4.3.2 Create the ResourceBundle ............................................................................ 9  
      4.3.3 Create the Style file ........................................................................................ 10  
      4.3.4 Create the JSF view ....................................................................................... 11  
   4.4 Build, Deploy and Run your application ..................................................................... 18
1. Business Scenario

The following guide will explain how to create a JSF application that calls an EJB Module. The EJB Module will contain the Student JavaBean and the StudentList Session Bean where the business logic of the application is defined. In addition, this example demonstrates the use of Java EE annotation that gives developers an easy and natural way for declaring resource dependencies simplifying the existing development practices.

It will also explain how to use the DataTable UI element in your JSF view to easily add a new student in the student list.

2. Background Information

Enterprise JavaBeans (EJB) Technology 3.0 has new updates features in the Java Platform, Enterprise Edition 5 (Java EE 5). JavaBeans has been defined by SUN as “reusable software programs that you can develop and assemble easily to create sophisticated applications”. To learn more about Java EE 5 Technologies, please visit Java EE Technologies at a Glance

Java EE Annotations are Java modifiers, similar to public and private, that simplify the application development process by allowing developers to specify within the Java class itself how the application component behaves in the container and requests for dependency injection. To learn more about Java EE Annotations you can visit Introduction to Java EE 5 Technology

The DataTable UI element renders an HTML 4.01 compliant table element and contains one or more column tags that define the columns of the table. The look and feel of the table can be manipulated with CSS classes. To learn more about the DataTable UI element, please visit the Tag Library Documentation

3. Prerequisites

The following is a list of all you need for developing JSF applications and invoking Enterprise Services.

- AS Java 7.1 (CE 7.1 or NW 7.1)
- NWDS 7.1 (SP3 or higher with latest patch level).

**Note**

While this tutorial is geared towards the SAP AS Java (the build/deploy steps of the guide), it wouldn’t be hard to replace the build/deploy portions with similar steps for any other Java EE 5 platform

Knowledge

- You have a basic knowledge of Java Enterprise Edition
- You have acquired some basic experience with JSF applications, for example by working through the JSF tutorials (Create a Hello World Application using JavaServer Faces [Extern] and Create Your First JSF Application [Extern])
4. Step-by-Step Procedure

The following guide will step you thru the creation of a simple Java EE 5 application comprising a java bean, a stateless session bean and a JSF view. You will create an EJB Module to manage the student information, a Web Module which contains the JSF view and uses resources from the EJB Module and an Enterprise Application needed to deploy the web module.

The user interface for this Web application consists of one JSF view with a Form UI element and a DataTable UI element. Once the user provides the student information and press the Add New Student Button, the student will be added to the table dynamically.

4.1 Tutorial Setup

1. Create a EJB Module Development Component named beanjsf/ejb.
2. Create a Web Module Development Component named beanjsf/web.
3. Create the dependency between the Web Module DC and the EJB Module DC. Switch to the Development Infrastructure perspective and select the Web Module DC (beanjsf/web). In the Component Properties view, select the Dependencies tab and add the EJB Module DC (beanjsf/ejb) in the required DCs section.
4. Switch again to the Java EE perspective and create an Enterprise Application Development Component named `beanjsf/ear`. Select the EJB Module and the Web Module in the Referenced Projects window.

### 4.2 Implement the EJB Module

In this step you will implement a `Student` Java bean and a `StudentsList` stateless session bean. The `Student` bean describes a single student and the `StudentsList` session bean describes a list that consists of a number of students and it is responsible for new students to the list.

#### 4.2.1 Create the JavaBean

1. From the context menu of the `ejbmodule` folder in the `EJB Module` project, create a Java class. Enter `Student` in the Name field, `com.sap.tutorial.jsf.ejb.beans` in the Package field, declare the attributes and generate a Constructor using fields and the corresponding Getters and Setters methods shown in the following code.

```java
public class Student {
    private String lastName;
    private String firstName;
    private int age;

    public Student() {
        super();
    }

    public Student(String lastName, String firstName, int age) {
        super();
        this.lastName = lastName;
    }
}
```
```java
    this.firstName = firstName;
    this.age = age;
}

public String getLastName() {
    return lastName;
}

public void setLastName(String lastName) {
    this.lastName = lastName;
}

public String getFirstName() {
    return firstName;
}

public void setFirstName(String firstName) {
    this.firstName = firstName;
}

public int getAge() {
    return age;
}

public void setAge(int age) {
    this.age = age;
}
```

### 4.2.2 Create the SessionBean

1. Create the stateless session bean. From the context menu of the `com.sap.tutorial.jsf.ejb.beans` package folder, select `New → Other`
2. Select EJB → EJB 3.0 → EJB Session Bean 3.0 on the New screen. Click the “Next” button.

3. Enter StudentsList in the EJB Class Name field and com.sap.tutorial.jsf.ejb.beans in the Default EJBPackage field.

4. Select Stateless in the Session Type field and Container in the Transaction Type field.

5. Deselect the Remote checkbox for Create Interface and press the Finish button.
6. The stateless StudentsListBean session bean was created. There are two types of session beans, stateless and stateful. The @Stateless annotation indicates to the EJB container that this class is a stateless session bean.

![Note]

Notice that StudentsListBean implements an interface called StudentsListLocal. This interface is the bean's business interface.

```java
@Stateless
public class StudentsListBean implements StudentsListLocal {

7. Declare two attributes: students and model

//ArrayList<Student> describes the list of students
private ArrayList<Student> students;

//ListDataModel represents the data over which the JSF UI element
//dataTable will iterates
private DataModel model = null;
```

8. Implement the Constructor to initialize the list of students and the DataModel that will be used by the h:dataTable JSF UI element, when the JSF view is displayed for the first time

```java
public StudentsListBean() {
    students = new ArrayList<Student>();
    students.add(new Student("William", "Wong", 14));
    students.add(new Student("John", "Smith", 12));
    students.add(new Student("Mari", "Beckley", 12));
    model = new ListDataModel(students);
}
```

9. Generate the Get method for the model attribute

```java
public DataModel getModel() {
    return model;
}
```

10. Create the addStudent method as shown in the following code

![Important]

To add or delete rows in the h:dataTable JSF UI element, you have to use the methods getWrappedData() and setWrappedData(). The getWrappedData() method gets a reference to the student's list to add the new student. Once students' list is modified, the method setWrappedData() resets the model with the new list, so it can be displayed on the screen.
```java
public void addStudent(Student student) {
    ArrayList<Student> currentStudents =
        (ArrayList<Student>) model.getWrappedData();
    currentStudents.add(student);
    model.setWrappedData(currentStudents);
}
```

### 4.2.3 Add SessionBean methods to local interface

1. Add the `addStudent` method to local interfaces. Locate the Outline view and from the context menu of the `addStudent` method choose EJB Methods → Add to Local Interfaces.

2. Repeat step 12 and add the `getModel` method to local interfaces.

3. The methods are included in the `StudentsListLocal` interface.

**Note**

The `@Local` annotation indicates that the business interface is a local business interface. Local business interface implementations must be in the same JVM as the client application invoking their methods.
4. Save the changes

### 4.3 Implement the Web Module

#### 4.3.1 Create a JSF Managed bean

1. Create a new package `com.sap.tutorial.jsf.ejb.util` from the context menu of the `Java Resources: source` folder in the `Web Module` project.

2. Create a Java class that will handle the user interface logic. From the context menu of the newly created package, select `new → class`. Enter `StudentListClient` in the `Name` field.

3. Declare an instance variable of type `StudentsListLocal`, which is the business interface for the `StudentsListSession` session bean using the `@EJB` annotation as follows:

```java
package com.sap.tutorial.jsf.ejb.beans;
import javax.ejb.Local;

public interface StudentsListLocal {
    public void addStudent(Student student);
    public DataModel getModel();
}
```

4. Declare the `currentStudent` attribute that describes the new student that will be added on the list and generate the corresponding `Get` and `Set` methods:

```java
private StudentsListLocal studentList;

private Student currentStudent = new Student();

public Student getCurrentStudent() {
    return currentStudent;
}

public void setCurrentStudent(Student currentStudent) {
    this.currentStudent = currentStudent;
}
```

5. Complete the User Interface logic with the following methods
public DataModel getStudents() {
    return this.studentList.getModel();
}

public void addStudent() {
    Student student =
        new Student(this.currentStudent.getFirstName(),
                    this.currentStudent.getLastName(),
                    this.currentStudent.getAge());
    this.studentList.addStudent(student);
}

6. Configure the StudentListClient bean in the application configuration resource file faces-config.xml using the managed-bean XML element in the Web Module project. Enter studentList in the Name field to reference the Student java bean and select session in the Scope field. The following XML code will be added in the Source tab:

   <managed-bean>
       <managed-bean-name>studentsList</managed-bean-name>
       <managed-bean-class>
           com.sap.tutorial.jsf.ejb.util.StudentListClient
       </managed-bean-class>
       <managed-bean-scope>session</managed-bean-scope>
   </managed-bean>

7. Save the changes

**4.3.2 Create the ResourceBundle**

1. Create the ResourceBundle choosing the com.sap.tutorial.jsf.ejb.util package and entering messages.properties in the File Name.

2. Enter the following keys and values for the English version of the localized messages:

   windowTitle=Students List
   pageTitle=Add New Student:
   addButtonText=Add New Student
   firstColumnHeader=First Name
   lastColumnHeader=Last Name
   ageColumnHeader=Age

3. For simplicity, only the English version of the localized message is created. Optionally you can create other versions of the localized messages and specify which languages are supported for
this application as indicated in the Product Offer tutorial Part 3 (International JSF application [extern]).

4. Expose the ResourceBundles by adding the following XML code in the Source tab of the faces-config.xml file

```xml
<application>
  <resource-bundle>
    <base-name>com.sap.tutorial.jsf.ejb.util.messages</base-name>
    <var>msgs</var>
  </resource-bundle>
</application>
```

### 4.3.3 Create the Style file

1. Create a Style file and define the following CSS classes

```css
.title {
  font-family: Verdana, Arial, Sans-Serif;
  font-weight: bold;
  font-size: 12px;
  color: #0000A0;
  font-style: normal;
}

.label {
  font-family: Verdana, Arial, Sans-Serif;
  font-weight: bold;
  font-size: 12px;
  color: #606060;
  font-style: normal;
}

.columnHeader {
  text-align: center;
  font-style: italic;
  font-weight: bold;
  color: #FFFFFF;
  background: #0000A0;
}

.students {
  border: thin solid black;
}
```
4.3.4 Create the JSF view

1. Drill into the Web Module project and right click on the WebContent folder and in the context menu select New → JSP.

2. Enter the file name index.jsp and click the Finish button. The JSP page will be created. The index.jsp page should be opened in the Web Page Editor.

3. Include the style sheet by adding a link element inside the head element as shown in the following code:

```html
<head>
  <link href="styles.css" rel="stylesheet" type="text/css"/>
  ...
</head>
```

4. The main UI elements contained in the index.jsp view are a Form and a DataTable. The following table contains the hierarchy of the Form UI elements:

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ViewRoot UI element</td>
<td></td>
</tr>
<tr>
<td>Form UI element in the UI-element ViewRoot</td>
<td></td>
</tr>
<tr>
<td>OutputText UI element in the UI-element Form</td>
<td>value #{msgs.pageTitle} styleClass title</td>
</tr>
<tr>
<td>PanelGrid UI element in the UI-element Form</td>
<td>Border 0 Columns 2</td>
</tr>
<tr>
<td>OutputText UI element in the UI-element PanelGrid</td>
<td>value #{msgs.lastColumnHeader} styleClass label</td>
</tr>
<tr>
<td>InputText UI element in the UI-element PanelGrid</td>
<td>value #{studentsList.currentStudent.lastName}</td>
</tr>
<tr>
<td>OutputText UI element in the UI-element PanelGrid</td>
<td></td>
</tr>
</tbody>
</table>
5. Result of the Form UI element added in the index.jsp view

```html
<h:form>
  <h:panelGrid border="0" columns="2">
    <h:outputText value="#{msgs.pageTitle}" styleClass="title"></h:outputText>
    <h:outputText value="#{msgs.lastColumnHeader}" styleClass="label">
      <h:inputText value="#{studentsList.currentStudent.lastName}"/>
    </h:outputText>
    <h:outputText value="#{msgs.firstColumnHeader}" styleClass="label">
      <h:inputText value="#{studentsList.currentStudent.firstName}"/>
    </h:outputText>
    <h:outputText value="#{msgs.ageColumnHeader}" styleClass="label">
      <h:inputText value="#{studentsList.currentStudent.ageName}"/>
    </h:outputText>
    <h:commandButton value="#{msgs.addButtonText}" rendered="true" action="#{studentsList.addStudent}"/>
  </h:panelGrid>
</h:form>
```
6. Now you are going to add the Data Table UI element that represents the student’s list. Click the JSF HTML toolset in the Palette, this will show all the UI elements available within it.

7. Drag and drop a Data Table element (found in the JSF HTML elements) to the Web Page Editor.

8. Take a look at the tags that were inserted into the JSP page.

Note

The body of h: dataTable tag typically contains one or more h: column tags that define the columns of the table. A column component is rendered as a single <td> element.

The body of h: dataTable tag can also contain header and footer facets. The header facets f: facet name="header" are rendered as a single <th> element in a row at the top of the table and the footer facets are rendered as a single <td> element in a row at the bottom of the table.

9. Select the Data Table UI element and then select the Properties view in the bottom window pane. Enter the text #{studentsList.students} in the Value property and student in the Var property.
Important

The Value property is bound to properties of the StudentListClient java bean. When this page is displayed, the getStudents method is called to obtain the element over which the DataTable will iterate (in this case an instance of the javax.faces.model.DataModel).

The Var property indicates the object in the array, list, result set, etc. that will be available for each iteration (in this case the Student JavaBean).

Note

The Value property has to be bound to a property of the following types:
- A Java object
- An array
- An instance of java.util.List
- An instance of java.sql.ResultSet
- An instance of javax.servlet.jsp.jstl.sql.Result
- An instance of javax.faces.model.DataModel.

10. The DataTable UI element can be customized extensively using the styles classes. Click the Attributes tab on your left in the Properties view and enter last, first, last in the columnClasses property, columnHeader in the headerClass property and students in the styleClass property.
Note

The columnClasses property receives a comma-delimited list of style classes to be applied to the columns of the table.

The headerClass property receives the style class to be applied to the headers of the table.

The styleClass attribute sets the style class to apply to the DataTable UI element when it is rendered.

11. By default the DataTable UI element is created with two columns, create another column using the Quick Edit tab on your left in the Properties view. Push the Add button and add the third column column3.
12. Change the header of the first column, by entering \#(msgs.lastColumnHeader) in the Value property of the OutputText element placed inside the header facets.

13. Drag and drop a OutputText element (found in the JSF HTML elements) inside the h:column tag and enter \#(student.lastName) in the Value property.
14. Repeat steps 20-21 to update columns 2 and 3 of the DataTable UI element. The properties are listed in the following table:

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Column2</strong> UI element in the UI-element DataTable</td>
<td>#{msgs.firstColumnHeader}</td>
</tr>
<tr>
<td><strong>Facets2</strong> UI element in the UI-element Column2</td>
<td>#{student.firstName}</td>
</tr>
<tr>
<td><strong>OutputText</strong> UI element in the UI-element Facets2</td>
<td>value: #{msgs.lastColumnHeader}</td>
</tr>
<tr>
<td><strong>OutputText</strong> UI element in the UI-element Column2</td>
<td>#{student.lastName}</td>
</tr>
<tr>
<td><strong>Column3</strong> UI element in the UI-element DataTable</td>
<td>#{msgs.ageColumnHeader}</td>
</tr>
<tr>
<td><strong>Facets3</strong> UI element in the UI-element Column3</td>
<td>#{student.age}</td>
</tr>
<tr>
<td><strong>OutputText</strong> UI element in the UI-element Facets3</td>
<td>value: #{msgs.lastColumnHeader}</td>
</tr>
</tbody>
</table>

15. Result of the DataTable UI element added in the index.jsp view
1. Create the application.xml deployment descriptor, sets the WAR file to “demo.sap.com~beanjsf~web.war” and the context root to “beanjsf” as indicated in the Hello World JSF tutorial (Create a Hello World Application using JavaServer Faces [Extern]).

2. Save changes.

3. Build and deploy the application.

4. Run the application using the following simplified URL:
   http://<servername>:<httpport>/beanjsf/faces/index.jsp

5. Results: