Using EJBs in Web Dynpro Applications

SAP NetWeaver 04

Copyright

Icons in Body Text

Icon	Meaning
	Caution
	Example
	Note
	Recommendation
	Syntax

Additional icons are used in SAP Library documentation to help you identify different types of information at a glance. For more information, see *Help on Help* → *General Information Classes and Information Classes for Business Information Warehouse* on the first page of any version of *SAP Library*.

Typographic Conventions

Type Style	Description
Example text	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.
Example text	Emphasized words or phrases in body text, graphic titles, and table titles.
EXAMPLE TEXT	Technical names of system objects. These include report names, program names, transaction codes, table names, and key concepts of a programming language when they are surrounded by body text, for example, SELECT and INCLUDE.
`Example text`	Output on the screen. This includes file and directory names and their paths, messages, names of variables and parameters, source text, and names of installation, upgrade and database tools.
`Example text`	Exact user entry. These are words or characters that you enter in the system exactly as they appear in the documentation.
`<Example text>`	Variable user entry. Angle brackets indicate that you replace these words and characters with appropriate entries to make entries in the system.
`EXAMPLE TEXT`	Keys on the keyboard, for example, `F2` or `ENTER`.

Using EJBs in Web Dynpro Applications 5
Importing the Initial Projects 7
Writing Data Access Command Beans 9
Checking the EJB Project 10
Creating a Java Proejct 11
Implementing the Command Bean 12
Creating the JAR 16
Defining the Web Dynpro Model 17
Importing the JavaBean Model 17
Creating the Context 19
Mapping the Component Context to the View Context 22
Binding UI Elements 23
Implementing the Web Dynpro Application 25
Deploying and Running the Sample 27

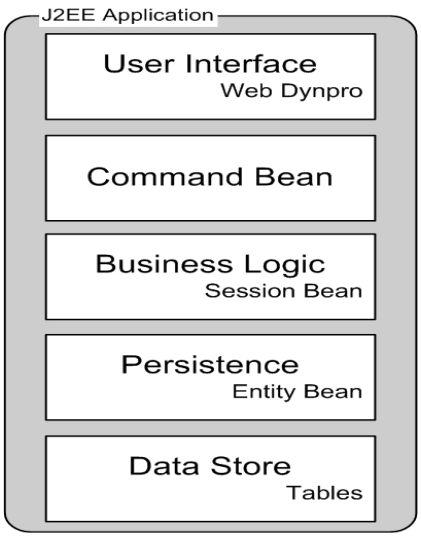
Using EJBs in Web Dynpro Applications

Introduction

This tutorial covers all the steps necessary for using existing business functions from within a Web Dynpro application. The business functions are provided in the form of an Enterprise JavaBean (EJB) application where the business logic is implemented through a stateless session bean and the persistence through a CMP-based entity bean.

In order to use the existing functions in the Web Dynpro application, we will import a model that facilitates their usage. The JavaBean importer is used to import the model. Strictly speaking, it would be possible to import a session bean using the JavaBean importer. The result would be a model class without any properties. Since you can only bind properties, not methods, to UI elements, this would not make sense. Thus, we will have to implement a command bean in the form of an intermediate layer, which serves as the input for the JavaBean import and provides the necessary properties.

Now the different layers of the application look as follows:

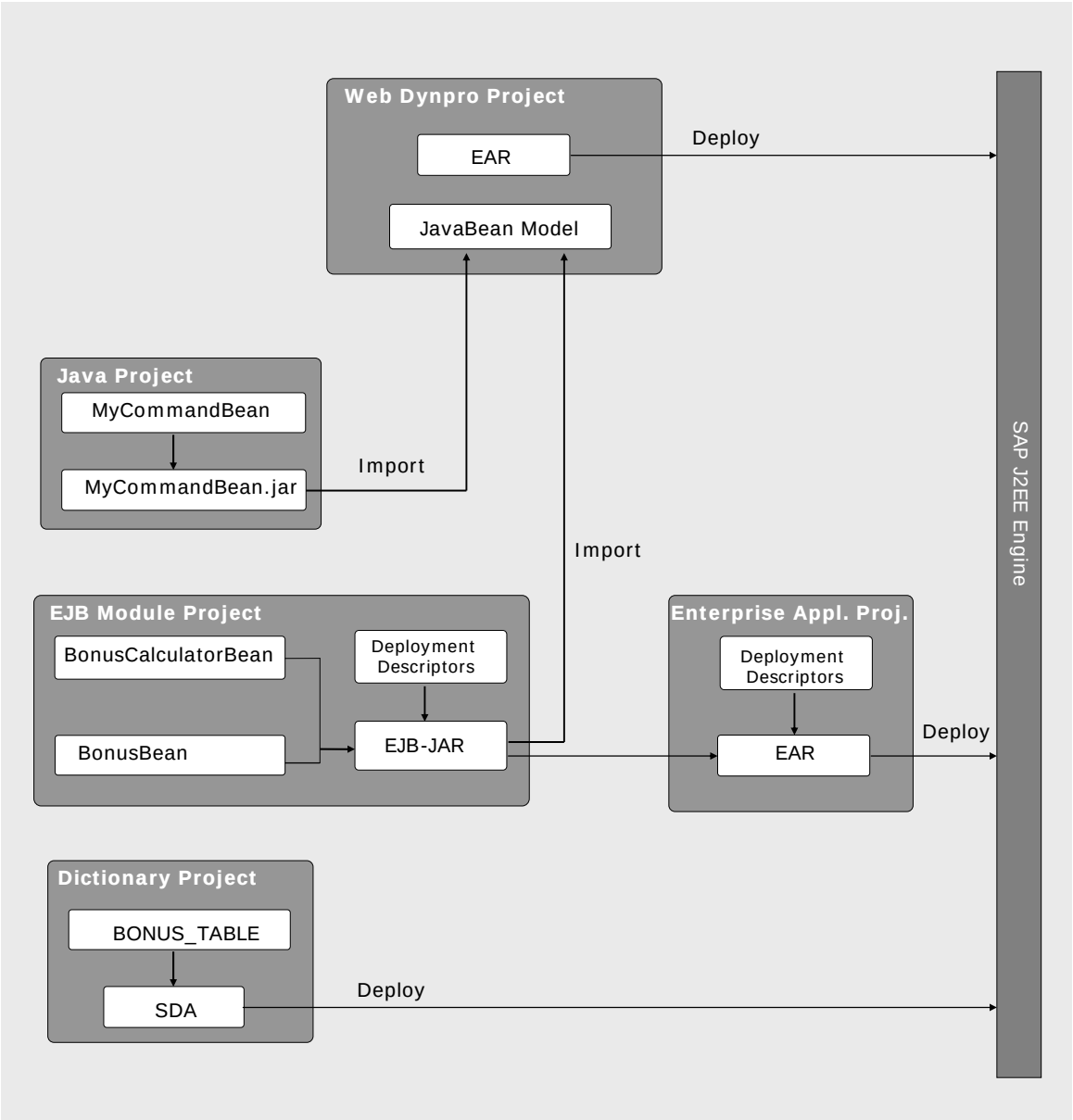


The example scenario in this tutorial consists of different components: a Dictionary table and Enterprise JavaBeans (EJBs), which are already provided as an application skeleton; a Data Access Command Bean (JavaBean), which will be developed in this tutorial; and a Web Dynpro Project. In the Web Dynpro, the command bean will be imported into the Web Dynpro project using the JavaBean importer wizard.

Sample Application

This tutorial aims at realizing a simple calculator Web application. It enables the user to calculate a bonus. The Web page accepts a social security number and a multiplier as the user's input and displays the calculated bonus amount in response. The calculated bonus must also be recorded in a database table. The bonus calculation is performed by the business method of a session bean. When the server receives the input, it passes the request to the *calcualateBonus ( )* method, which calculates the bonus. Then it returns the calculated value to the end user. The *storeData* business method also receives the calculated bonus and stores it in the database table. For accesses to table data, a container-managed persistence entity bean is used. Since the social security number is unique, it can only be entered once. If the same number is entered more than once, the storeData method returns a *Duplicate Key Exception* message.

The following picture illustrates the different projects and the enclosed components of our example application.



Task

Most of the components of the example application no longer need to be created and are, for the most part, ready for use – provided you start with a project base that already exists. In this tutorial, we will focus on all the development steps that are required in a Web Dynpro application in order to connect up to the business logic implemented by EJBs when using the JavaBean import.

The EJB application and the initial Web Dynpro application are already provided in the form of an initial example application, which can be imported from SDN. Then, the command bean is implemented in a separate Java project and imported into the Web Dynpro project. Finally, the context mappings are performed and the required methods are implemented to invoke the business methods form within the Web Dynpro. At runtime, the data entered by the user of the application is passed to the model through the data binding between the input fields and the context elements, and through the model binding of these context elements.

Objectives

By the end of this tutorial, you will be able to:

- ✓ Write a simple Data Access Command Bean based on predefined business logic.
- ✓ Generate a model to be used for linking up the business logic of the EJB project from within the Web Dynpro project.
- ✓ Declare a context node in the component controller in such a way so that a connection to the model can be created.
- ✓ Perform the implementation for using business methods in Web Dynpro components.
- ✓ Import all necessary libraries to be used by the model import wizard in the Web Dynpro project.

Prerequisites

Systems, installed applications, and authorizations

- ☐ You have installed the SAP NetWeaver Developer Studio.
- ☐ You have access to the SAP J2EE Engine.

Knowledge

- ☐ You have acquired some basic experience with Web Dynpro applications.
- ☐ You have basic knowledge of the EJB programming model.
- ☐ You are experienced in working with the SAP NetWeaver Developer Studio.

Importing the Initial Projects

To focus the development of this example application on the actual content covered, there is a predefined sample application template available in the SAP Developer Network (SDN) http://sdn.sap.com (*Web Application Server* area | *Web Dynpro* | *Samples and Tutorials Quicklink*).

Prerequisites

- You have access to the SAP Developer Network (http://sdn.sap.com) with a user ID and password.
- The SAP NetWeaver Developer Studio is installed on your computer.

Procedure

Importing the initial projects into the Developer Studio

1. Call the SAP NetWeaver Developer Network using the URL http://sdn.sap.com and log on with your user ID and the corresponding password. If you do not have a user ID, you must register before you can log on.
2. Navigate to the *Web Application Server* | *Web Dynpro* area and then to the *Samples and Tutorials* section.
3. Download the ZIP file *TutWD_EJBinWD_Init.zip*, which contains the Dictionary project *BonusCalculationDic*, the EJB Projects *BonusCalculationEJB*, and *BonusCalculationEar* as well as the Web Dynpro project *TutWD_BonusCalculation_Init*.

4. Unzip the contents of the ZIP file into the work area of the SAP NetWeaver Developer Studio or into the local directory.
5. Call the SAP NetWeaver Developer Studio.
    a. Import the projects. To do this, choose *File* → *Import* in the new menu. Choose *Multiple Existing Projects into Workspace* and then *Next* to confirm.
    b. Choose *Browse*, open the folder in which you unzipped the projects of the ZIP file *TutWD_EJBinWD_Init*, select those projects (*BonusCalculationDic*, *BonusCalculationEJB*, *BonusCalculationEar* and *TutWD_BonusCalculation_Init*), and choose *Finish* to confirm.

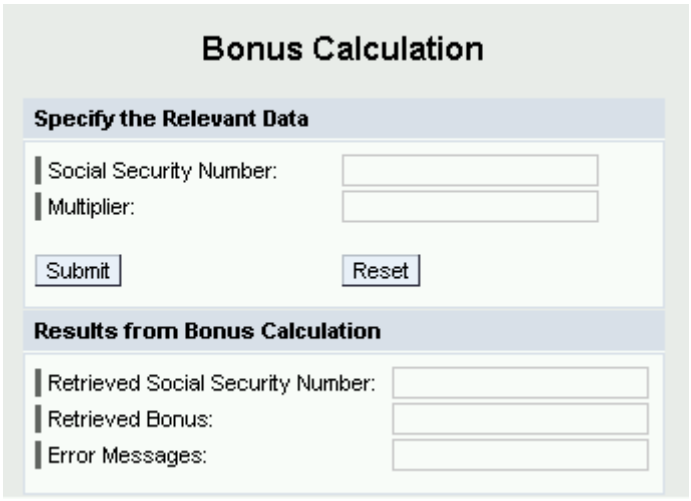
Initial Projects

After you have imported the project templates, the following projects are visible in the Developer Studio:

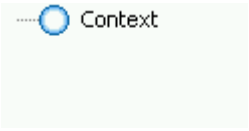
Dictionary Project
BonusCalculationDic that contains the table **BONUS_TABLE** of the bonus calculation data.
J2EE Projects
EJB project: **`BonusCalculationEJB`** The session bean of the EJB module project has two business methods – *calculateBonus( )* and *storeData( )*.
EAR project: **`BonusCalculationEar`**

Web Dynpro project
Web Dynpro project: **`TutWD_BonusCalculation_Init`** This predefined Web Dynpro project implements the UI layer.
 Web Dynpro application: **`BonusCalculationApp`**
 Web Dynpro component: **`BonusCalculationaComp`** This is the Web Dynpro component that contains our entire application.
 View: **`BonusCalculationView`** In this view, the user can enter the social security number and the multiplier into the appropriate input fields and trigger the bonus calculation. On the result area, some bonus calculation data will be displayed. Initially, some UI elements, context elements, and event-handlers are provided in the project.

Layout:



Context:



Methods:

T.	Name	Return type
	onActionReset	void
	onActionSubmit	void

Window: **`BonusCalculationComp`**

Writing Data Access Command Beans

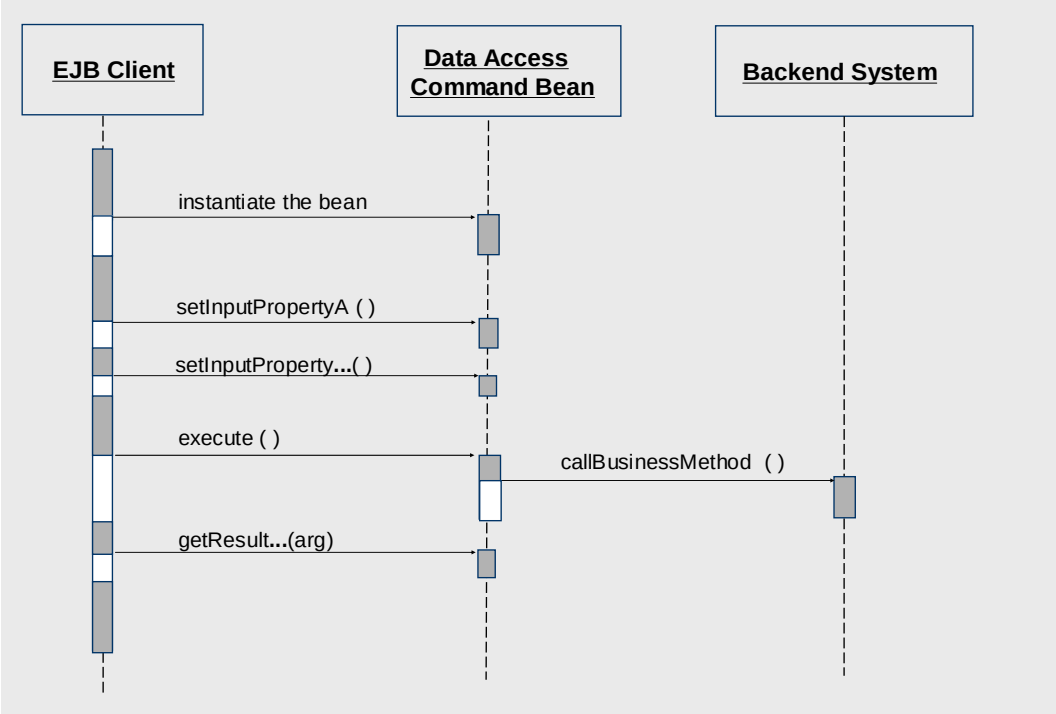
Applying the Data Access Command Bean Pattern

To be able to use the JavaBean Importer in the Web Dynpro, we require an access layer (intermediate layer) consisting of JavaBeans that encapsulate access to the back-end system. This kind of intermediate layer, therefore, forms a high-level component that lies between the business logic and a Web Dynpro application that serves as an EJB client.

It also serves to hide all low-level aspects of the application – such as data acquisition logic or details regarding the storage of persistent data – and to decouple them from the EJB layer. However, in the Developer Studio there is no way to simply generate such an intermediate layer – for example, using a wizard. It must be coded manually.

If you use this pattern, a Client application will program against the interface that is provided by the bean. A Data Access Command Bean (DACB) is a pure JavaBean object that represents a simple interface to Enterprise Java Bean clients. For the most part, the implementation of the DACBs consists of Set and Get methods, as well as an Execute method.

The following diagram displays what an EJB client needs to do. First of all, the JavaBean needs to be instantiated before all the required input parameters are set using the Set method. The actual call of the business method takes place using execute( ). The back-end files resulting from this call are the passed on to the client using Get methods.



Resulting Steps

If we transfer the DACB pattern to our example, this will have the following consequences:

The first work step will be that we create a DACB suitable for the corresponding business method in the session bean from the EJB project in a separate Java project and that we implement it according to DABC patterns. The resulting JAR is then used at a later time as input for the JavaBean importer in the Web Dynpro.

Checking the EJB Project

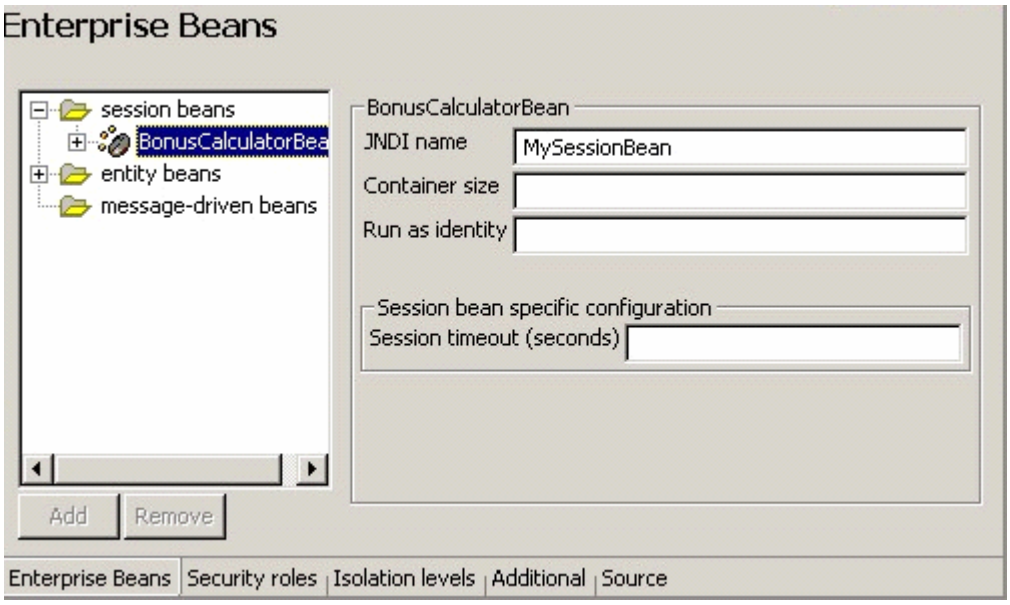
When you attempt to call an EJB, it is important that this EJB has a unique JNDI name. Since the Data Access Command Bean, which we are about to create, calls the session bean of the *BonusCalculationEJB* project, we need to ensure that the session bean is given such a JNDI name.

Prerequisites

- ☐ You have imported all the required projects.

Procedure

Switch to the *J2EE Perspective*. From the project structure select *BonusCalculationEjb*. Open the *ejb-j2ee-engine.xml* by double-clicking it. Select the *Enterprise Beans* tab if not already selected. Expand the *session beans* node and double-click on *BonusCalculatorBean*. In the field *JNDI name,* **MySessionBean** should be displayed. If not, enter **`MySessionBean`** and save the changes.



Creating a Java Project

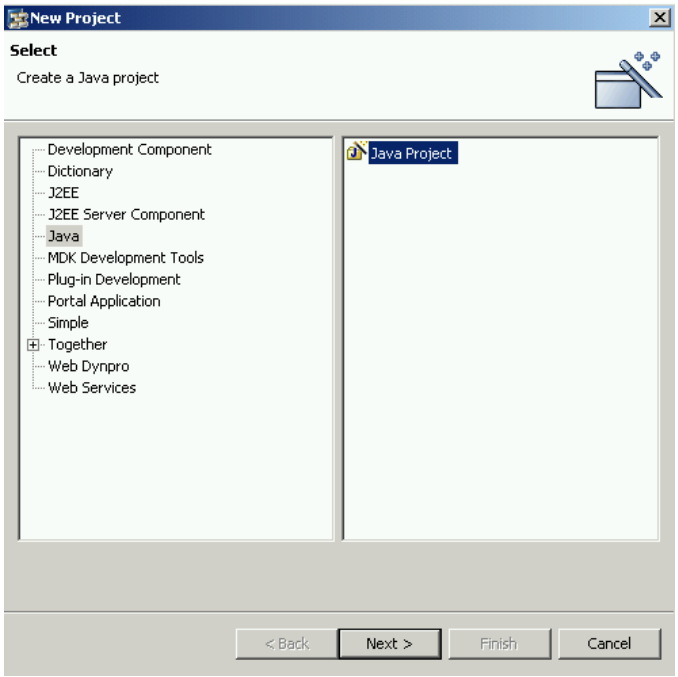
In this procedure you will create a separate Java project that serves as a container for the Data Access Command Beans and will be imported in the Web Dynpro model.

Prerequisites

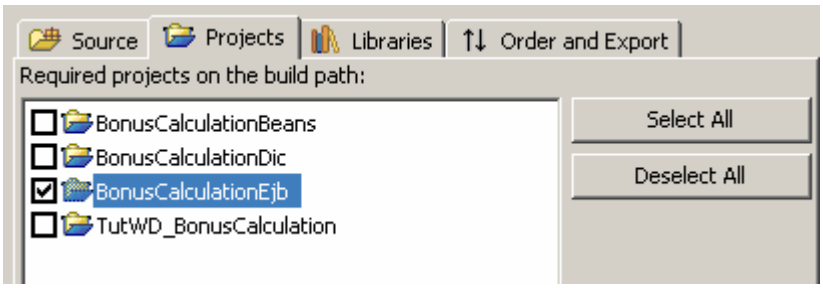
☐ You have imported all the required projects.

Procedure

1. Switch to the *Java Perspective* and choose *File* → *New* … → *Project* to start the *New Project* wizard. Select *Java* (in the left-hand pane) and *Java Project* (in the right-hand pane). Choose *Next*.



2. Give your Java Project the name **`BonusCalculationBeans`** and choose *Next*.
3. In the *Projects* tab, select the EJB project *BonusCalculationEjb* and choose *Finish*.



4. Confirm the upcoming message with *Yes*.

Result

The wizard generates a default structure for your new Java project.

Implementing the Command Bean

The JavaBean class `MyCommandBean`, which serves as a Command Bean, should be able to store all the data that is needed to calculate the bonus and to display the result (including possible error messages). `MyCommandBean` will be initialized from within the Web Dynpro and passes the bonus calculation data to the session bean (`BonusCalculatorBean`) of the EJB Project.

MyCommandBean will be imported for the JavaBean model into the WebDynpro and supplies the properties to be bound to the controller context. For this purpose, **MyCommandBean** must contain a field for each relevant value. The fields will be named **multiplier**, **ssn**, **bonusAmount**, and **message**.

Procedure

Creating the JavaBean MyCommandBean

1. In the *Java Perspective,* select the *BonusCalculationBeans* project and choose *New* → *Class* from the context menu.
2. Enter *MyCommandBean* for the name and *com.sap.bonus.calculation.bean* for the package. Choose *Finish.*

Implementing the Command Bean Class

Now that the new class has been created, the bean properties *multiplier*, *ssn*, *bonusAmount,* and *message* will be declared.

1. Add the following attributes to the class:

```
Public class  MyCommandBean {
   private int multiplier;
   private String ssn;
   private double bonusAmount;
   private String message;
...
}
```

2. Since we want to call the session bean from within the command bean, we now need to declare variables for the local and the local home interface (**`BonusCalculatorLocal`**

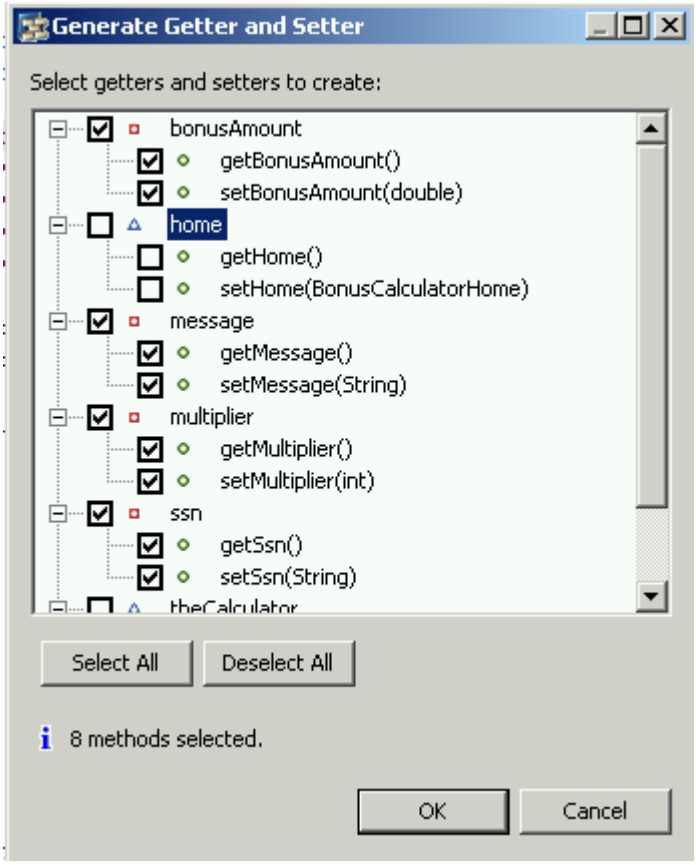
and **BonusCalculatorLocalHome**) of the session bean. Insert the following lines of code:

```
public class  MyCommandBean {
   private int multiplier;
   private String ssn;
   private double bonusAmount;
   private String message;

   BonusCalculatorLocal theCalculator = null;
   BonusCalculatorLocalHome home = null;
}
```

Do not worry if errors occur. From the context menu of the editor, choose *Source* → *Organize Imports*. Now there should be no errors left.

3. Since this Java class needs to comply with the JavaBean specification, we have to write Get and Set methods for the properties we declared before. From the context of the editor, choose *Source* → *Generate Getter and Setter Methods*. From the screen that appears select *bonusAmount*, *message*, *multiplier*, and *ssn*.



4. Now we have to write a constructor that will be executed each time the command bean is instantiated. In this constructor, the look-up for the session bean will be executed. Consider, that we need to add **localejbs** in the lookup string while using local interface for EJB access.
   Add the following lines of code:

```
public class  MyCommandBean {
 private int multiplier;
 private String ssn;
 private double bonusAmount;
 private String message;

 BonusCalculatorLocal theCalculator = null;
 BonusCalculatorLocalHome home = null;

 public MyCommandBean() throws CreateException {
   //looks up the session bean and creates the Home interface
   try {
    InitialContext ctx = new InitialContext();
    home = (BonusCalculatorLocalHome)ctx.lookup("localejbs/MySessionBean");
    theCalculator = home.create();
    } catch (Exception namingException) {
    namingException.printStackTrace();
    }
 }

}
```

Again, from the context menu of the editor, choose *Source* → *Organize imports*.

5. The next step is to create a method called execute, which calls the 2 business methods (*calculateBonus* and *storeData*) of the session bean. Insert the following lines of code:

```
public void execute() {
   //Calls the calculateBonus method of the session bean
   //which calculates and returns the bonus

   try {
      this.bonusAmount = theCalculator.calculateBonus(multiplier);
   } catch (Exception e) {
      e.printStackTrace();
      return;
   }

   // Calls the storeData method which tries to store the calculated bonus
   // and the corresponding Social Security Number in the entity bean.
   // In case the Social Security Number has been entered already the
   // method returns “Duplicate Key Exception”, if not it returns “”.

   try {
      this.message = theCalculator.storeData(this.getBonusAmount(),
                                             this.getSsn());
   } catch (Exception e1) {
      e1.printStackTrace();
   }
}
```

In this simple example we do not make any use of explicit transactions calls. However, in general case we recommend to handle transactions using JTA.

Result

You have successfully implemented the command bean, which will serve as an input for the Web Dynpro model importer.

Creating the JAR

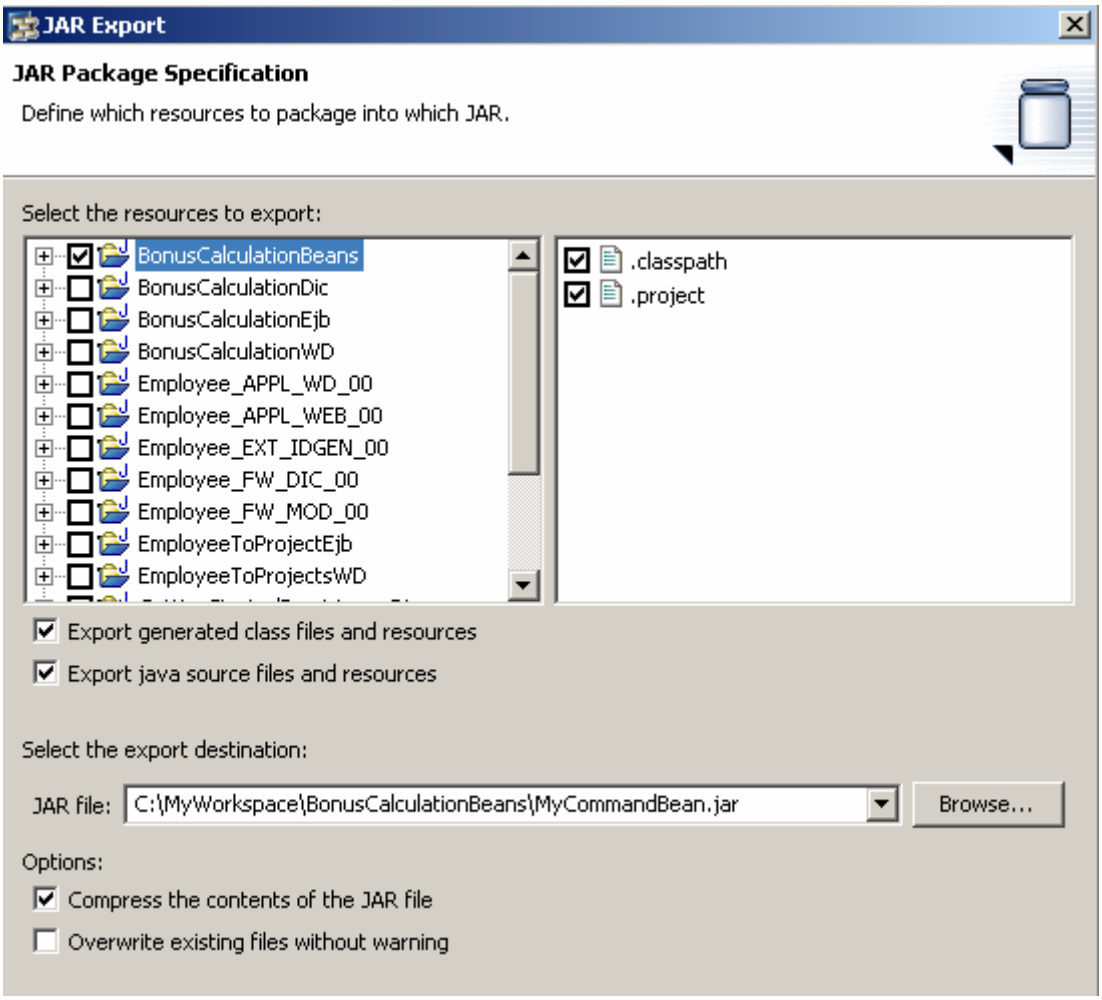
In order to be able to import the command bean through the JavaBean import, a JAR file needs to be exported.

Prerequisites

- ☐ The structure of the Java Project **`BonusCalculationBeans`** is currently displayed in the *Package Explorer*.

Procedure

1. Switch to the *Navigator* view and select *BonusCalculationBeans*. From the context menu, choose *Export* → *Jar File* and choose *Next*.
2. Choose the *Browse* button, navigate to *BonusCalculationBeans*, and enter **`MyCommandBean.jar`** in the *Name* field. Choose *Save* and then *Finish*.



Result

You have created the JAR file and are now ready to import the JavaBean model.

Defining the Web Dynpro Model

In compliance with the MVC paradigm, a model in Web Dynpro enables the user to access business data that is stored outside the Web Dynpro application. In our case, however, the business data that belongs to our bonus sample application is made accessible through Data Access Command Beans. These beans offer exactly the input we need for the JavaBean model importer in Web Dynpro.

Within the Web Dynpro project, you will obtain such a model as a result of proxy generation. It is represented by a set of model classes, including their relations. However, in our quite simple case we will see only one model class generated and there is no relation defined here. In the following step, we only need to bind the model object to the context. Each context, however, has a node that represents the corresponding model object. The model node, in turn, defines a set of attributes that corresponds with the properties of the JavaBean. Finally, we only need to take care of the link from the UI elements to the business data referenced in the view controller context (data binding for the Web Dynpro view's UI elements).

Importing the JavaBean Model

A *model* enables the user to access business data that is outside the Web Dynpro application from within the application. In this tutorial, the business data that belongs to the bonus calculation is made accessible through a JavaBean import.

Within a Web Dynpro component, business data is stored in separate context structures (consisting of context nodes and context attributes). The link between the context elements contained there and the business data in an existing model can be set up using the communication classes and auxiliary classes required for this purpose. You can use the Web Dynpro tools to generate such a model for a command bean. The model mainly consists of special model classes that you can use to link context structures to the model.

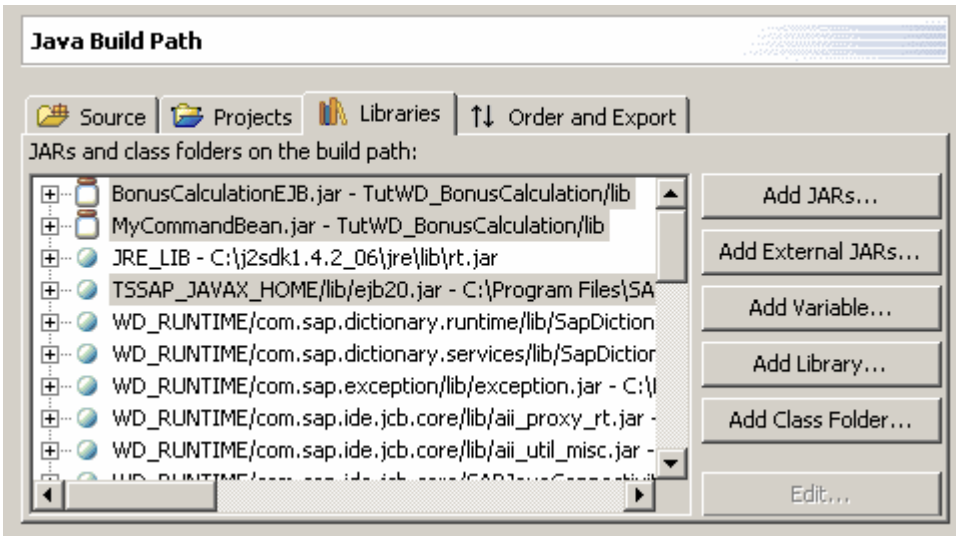
Below you will learn how you can generate such a model.

Procedure

Adding JARs to the Web Dynpro Project Classpath

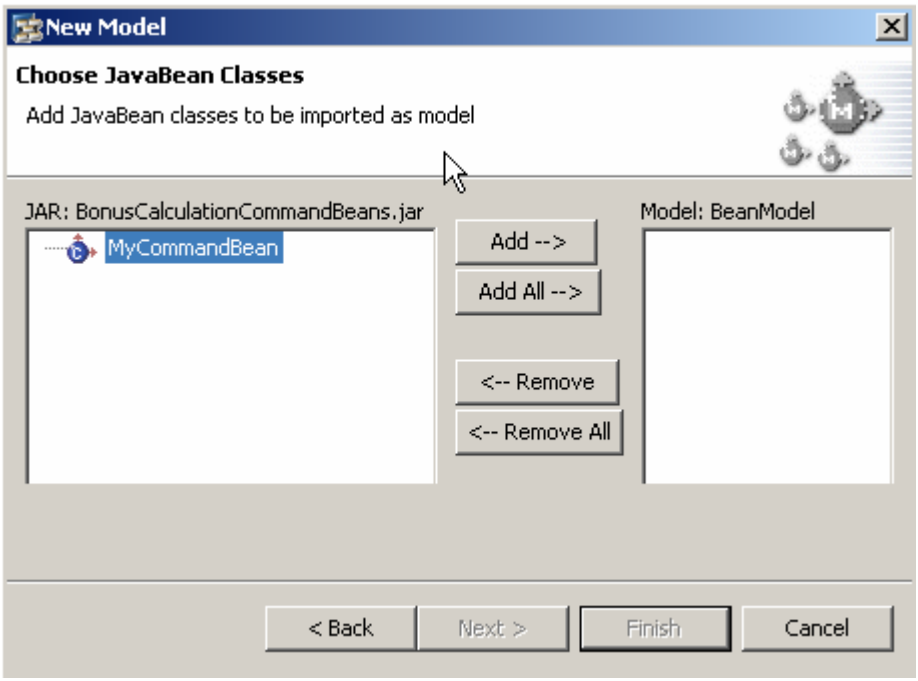
Now you have to put the .jar file of the EJB Module Project and the jar file Java Project in the library folder of the Web Dynpro application.

1. Switch to the *Navigator* View and select *BonusCalculationEJB* → *BonusCalculationEJB.jar*. From the context menu, select *Copy* and navigate to *TutWD_BonusCalculation_Init* → *lib*. Select *Paste* from the context menu.
2. Proceed in the same way with the JAR file *MyCommandBean.jar* from your Java project.
3. Add the two JAR Files to the classpath of your Web Dynpro project *TutWD_BonusCalculation_Init.*
4. Add also the *ejb20.jar* to the classpath. Use the classpath variable TSSAP_JAVAX_HOME.



Importing the JavaBean Model Implementation

1. In the project structure, expand the node *Web Dynpro* → *Models*. From the context menu, choose *Create Model* to start the appropriate wizard.
2. Choose the *Import JavaBean Model* option, followed by *Next*. Enter the name **`BeanModel`** as the model name and **`com.sap.bonus.calculation.model`** as the package name.
3. Select *local JAR file* and browse for the *MyCommandBean.jar*. Select *Deploy Time* and click *Next*.
4. In the next screen, select *MyCommandBean* and choose*Add* -->.



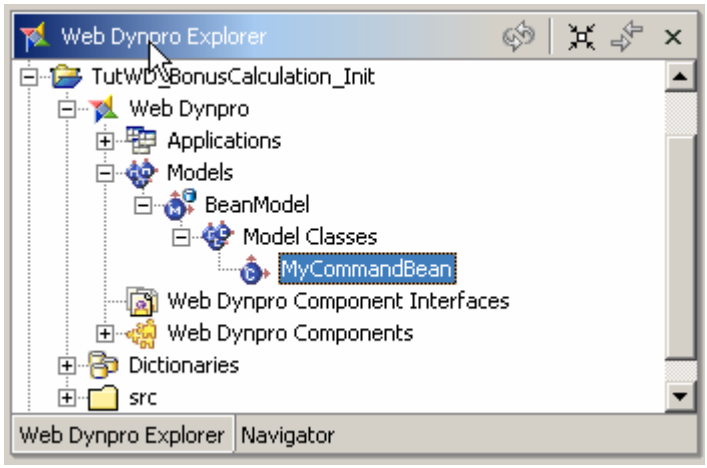
5. Choose *Finish*.

Result

You have now created a model named *BeanModel* in your Web Dynpro project. In accordance with the MVC paradigm, the model was not simply generated as part of the Web

Dynpro component, but as an independent development object. Accordingly, you will have to explicitly create this model for the Web Dynpro component before it can be used there.

The generated model class is now visible within the project structure in the *Web Dynpro Explorer* under the *Models* node.



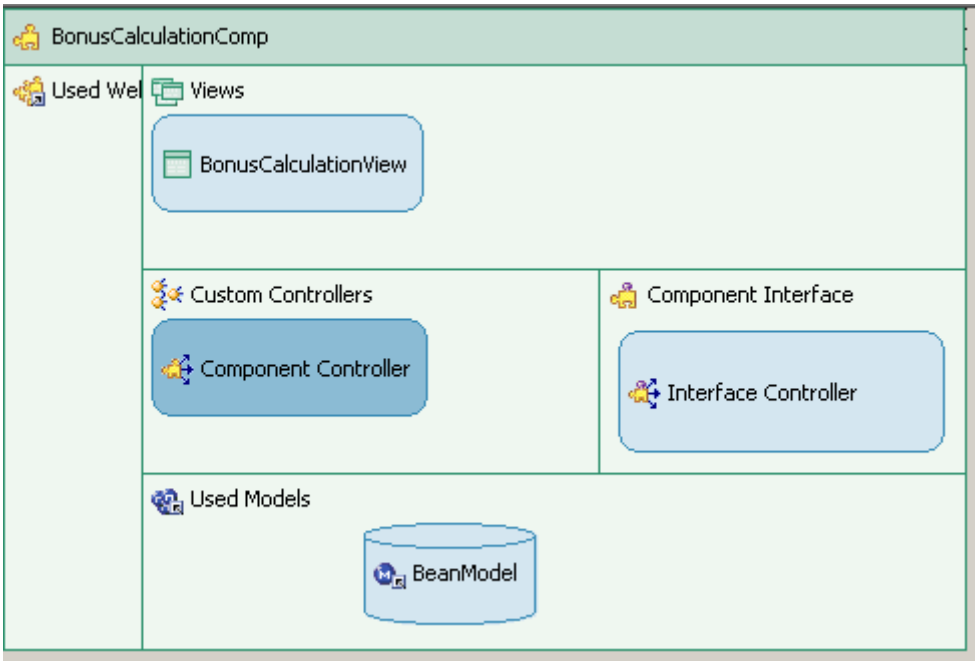
Creating the Context

Each Web Dynpro component is supplied with a corresponding *Component Controller*. This controller is responsible for the acquisition of the data required by the command bean for the bonus calculation. Accordingly, it must be able to depict the corresponding input and output structures of the *SessionBean* model. In order to create a connection between the *Component Controller* and the model created in the previous step, you will bind the context of the *Component Controller* to the created model structure using the *Data Modeler*.

Procedure

Adding a Model to the Web Dynpro Component

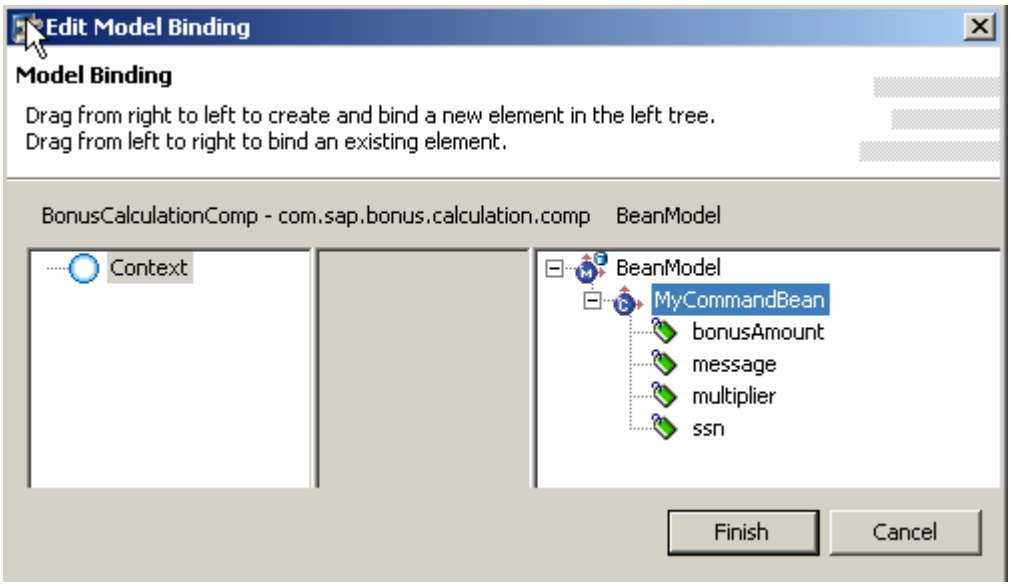
1. Open the Data Modeler through the context menu entry *Open Data Modeler* of the node *BonusCalculationComp*. In the toolbar on the left, choose the *Add an existing model to the component* icon. The icon will turn gray. Place the cursor on the *Used Models* area and left-click.
2. Select the *SessionBeanModel* checkbox and confirm by pressing *OK*. The structure of the Web Dynpro component *BonusCalculationComp* will look like this in the *Data Modeler*:



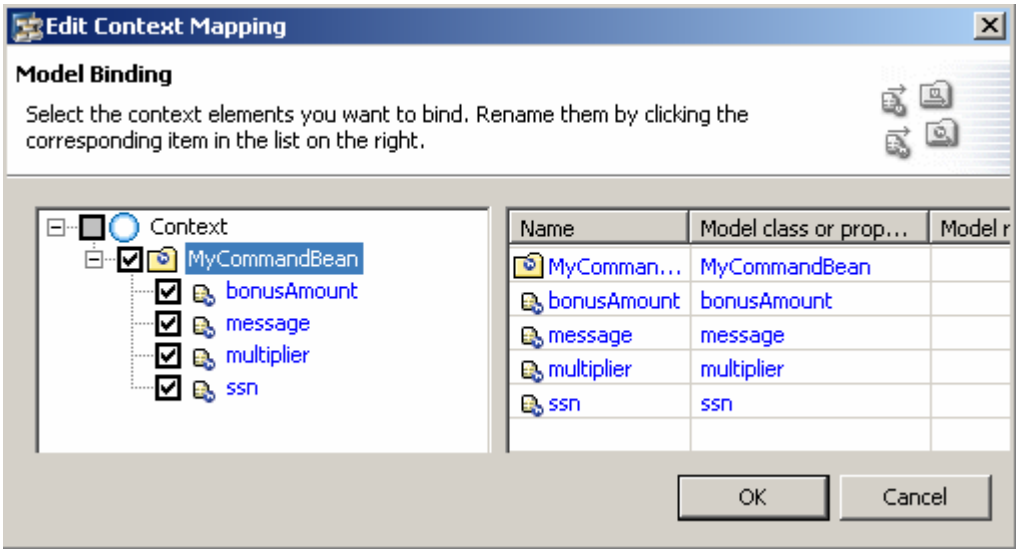
Binding the Component Controller Context to the Command Bean

In the *Data Modeler,* you can easily declare the connection between the context of the *Component Controller* and the *BeanModel* you have created.

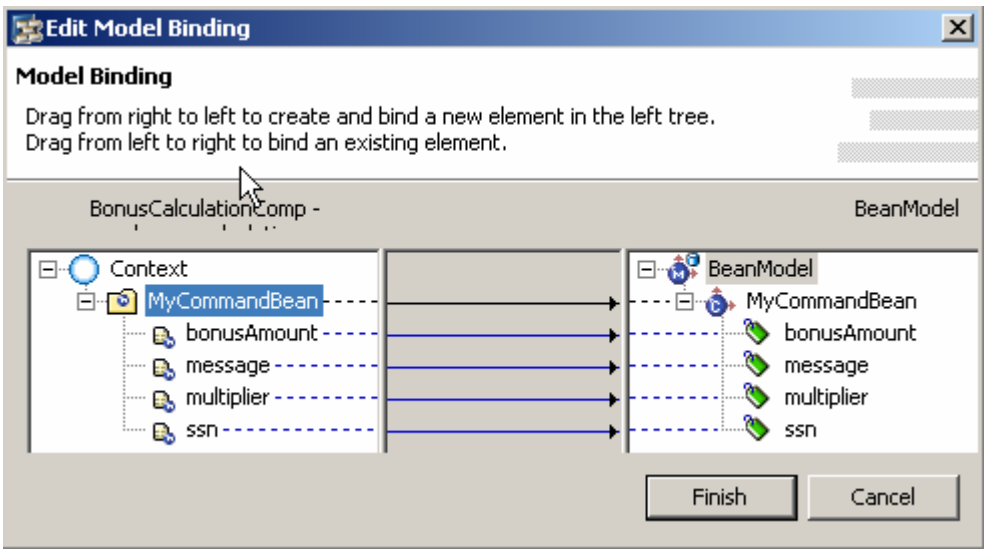
1. In the *Data Modeler* toolbar, choose the *Create a data link* icon. Draw a line beginning at *Component Controller* and ending at *BeanModel.* The *Model Binding Wizard* starts automatically.
2. Using Drag & Drop, drag the node of the model class **`MyCommandBean`** in the *BeanModel* to the root node of the component controller context, and drop it.



3. In the following input dialog, select the model node and all the model attributes.



If you now expand the context tree in the following dialog boxes, the declared model link between the *Component Context* node and the model classes, including their relations, will be displayed graphically through link lines. The relations are expanded in the model tree (right window) in the associated classes.



4. Close the *Model Binding Wizard* by choosing *Finish*.

Result

From the model definition, you have now defined a structure of context model elements (consisting of model nodes and model attributes) in the component controller **`BonusCalculationComponent`**, and you have also bound these to the corresponding model class.

Mapping Component Context to View Context

In the last step, you created a structure for context model elements in the context of the component controller and also bound this structure to a generated model class. The component controller is a central point in Web Dynpro. From here, it is possible to control the

functions of a Web Dynpro component. In this way, its context serves as a storage area for all the data received from the model. However, a component controller cannot be linked with a view. For this reason, model data from the context of the component controller must be able to be passed to the context of a view controller. Only then can model data be displayed in a view.

It would be technically possible for a view context to access the model, but this would not be good design style. To use the MVC paradigms in a consistent manner, you require the component controller as a binding link between the view and the model. In this step of the tutorial, you will map the context structure of a component controller onto the context of the view controller. Using this context mapping, you enable context elements of the view to be referenced – that is, elements that have data stored in the component context and themselves represent a copy of the actual model data.

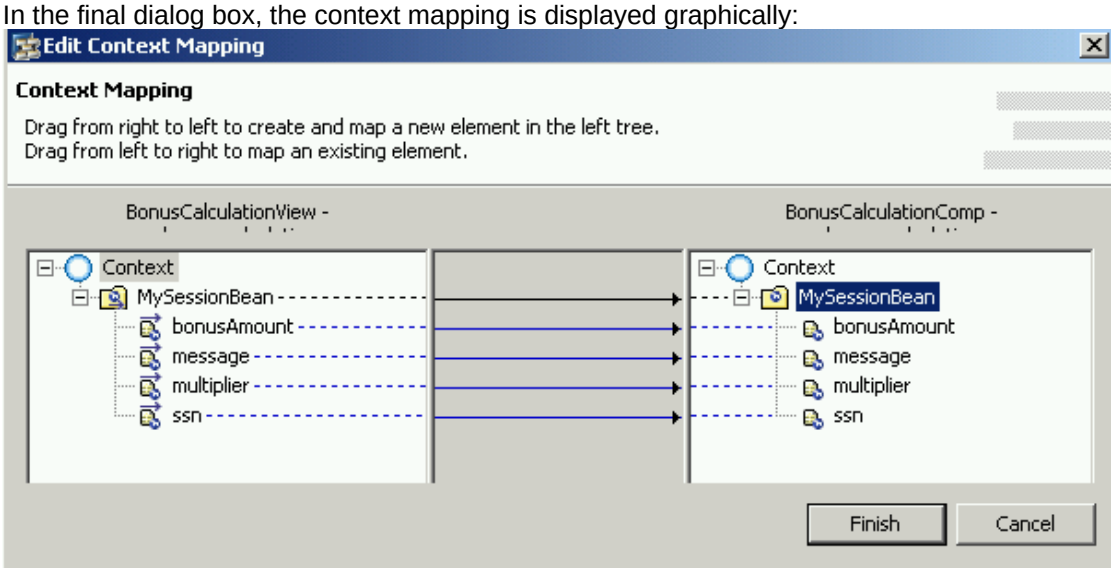
Prerequisites

☐ Open the *Data Modeler* for the Web Dynpro component *BonusCalculationComp*.

Procedure

1. In the toolbar, choose *Create a data link.* Draw a line, beginning with the *BonusCalculationView* view and ending at the *Component Controller*. The *Model Binding Wizard* starts automatically.
2. Using Drag & Drop, drag the node of the model class **`MyCommandBean`** in the *Component Controller* to the root node of the view controller context. In the following input dialog, select the model node and model attributes. Then choose *OK*.

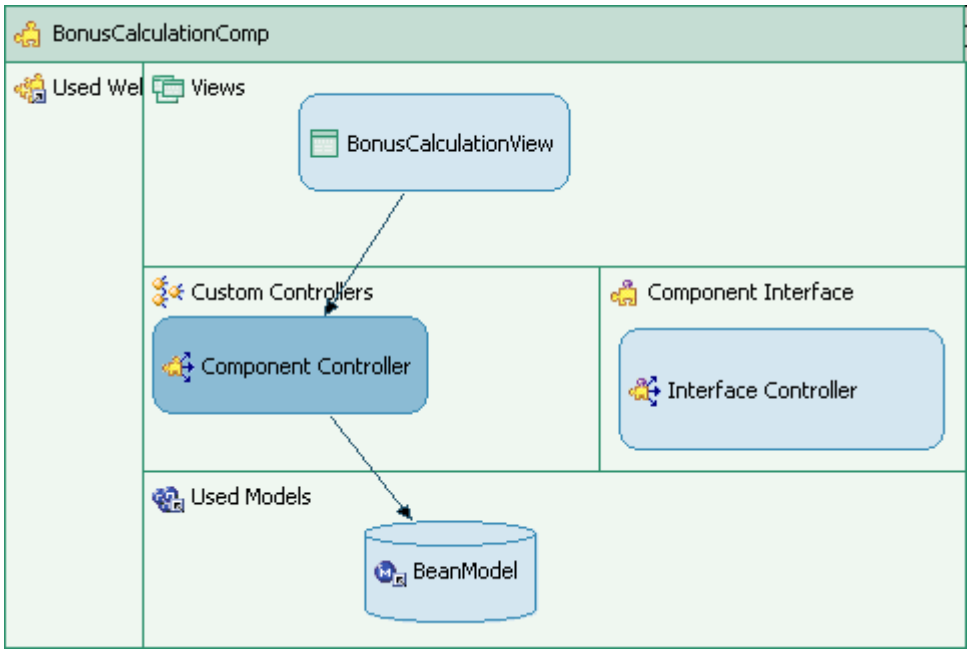
   In the final dialog box, the context mapping is displayed graphically:



3. Close the *Model Binding Wizard* by choosing Finish.

Result

You have created the necessary view context and mapped it to the component context you created previously. The *Data Modeler* shows this through the appropriate arrow links.



You are now in a position to bind UI elements such as input fields to the corresponding view context elements.

Binding UI Elements

With the *BonusCalculationView*, the imported project template has a predefined form that allows input of relevant bonus calculation data. It returns the social security number, the calculated bonus, and – if necessary – an error message on the basis of the results data returned by the command bean. Therefore, you do not need to worry about the layout of your example application.

You only need to take care of the link from the interface elements to the business data referenced in the view controller context. You can do this easily through **Data Binding**.

Prerequisites

☐ You have set up the view context for the FormView and mapped it to the component controller context.

Procedure

1. Start the layout editor for the *BonusCalculationView*.
2. Set up the data bindings between the UI element properties and the respective context model attributes in accordance with the table below.

   You can start the wizard for data binding of a UI element property to a context attribute by pressing the … button at the right margin of the *Value* column in the *Properties* view.

Property	Value
Input field **`SSNInputField`** of type *InputField*	

Properties of InputField – value	MyCommadBean.ssn
Input field **`MultiplierInputField`** of type *InputField*	
Properties of InputField – value	MyCommandBean.multiplier
Input field **`RetSsnOutputField`** of type *InputField*	
Properties of InputField – value	MyCommandBean.ssn
Input field **`RetBonusOutputField`** of type *InputField*	
Properties of InputField – value	MyCommandBean.bonusAmount
Input field **`MessageOutputField`** of type *InputField*	
Properties of InputField – value	MyCommamdBean.message

3. Save the current status of your project.

Implementing the Web Dynpro Application

After you have successfully executed all the required declarative development steps, you now need to add some individual lines of Java code for the view controller.

Within each controller, there are some predefined places where you can add source code. These include, in particular, the standard methods **`wdDoInit()`** and **`wdDoExit(),`** and also the Action Event-Handler in the view controllers.

`wdDoInit()` is always controlled if the controller is instanced. For this reason, you must remember that at this point there is an object that – at runtime – represents the entire input, including all the input parameters for calling the command bean business method.

In the action event handler **`onActionSubmit(),`** the actual Command Bean call must be implemented, based on the data entered by the user and stored in the context concerned.

In the action event handler **`onActionReset(),`** the clearing of the fields in the UI has to be defined, calling setter methods.

Prerequisites

- ☐ You have created the JavaBean import model.
- ☐ You have mapped the view context onto the Component Controller context.

Procedure

Implementing the Component Controller

Here you define an object that – at runtime – represents the input page of the command bean method. This object must also be bound to the model node **`MySessionBean`**, which was declared at design time.

1. Choose the *Implementation* tab for the component controller **`BonusCalculationComp`**.

   After the generation routines have been run once again, the updated source code of the view controller implementation is displayed.

2. Between `//@@begin wdDoInit()` and `//@@end` of the method `wdDoInit()`, enter the following Java code:

```
/** Hook method called to initialize controller. */
public void wdDoInit()
{
  //@@begin wdDoInit()
   try {
      wdContext.nodeMyCommandBean().bind(new MyCommandBean());
   } catch (Exception ex) {
      IWDMessageManager msgMgr = wdComponentAPI.getMessageManager();
      msgMgr.reportException(ex.getLocalizedMessage(), true);
   }
  //@@end
}
```

3. In the context menu of the source code editor, choose the function *Source → Organize Imports* in order to add the missing import line.

The **`MessageManager`** class is imported into the view controller and you can use the generic UI service to display message texts in the user interface.

Implementing the Action Event Handler of the View Controller

The actual command bean is now called through the **`execute()`** method call of the model object currently stored in the context model node. This already contains the bonus calculation data entered by the user – through *data binding* and *context mapping*.

The returned results – in the example application, solely the social security number, the bonus amount, and the message – are then displayed automatically in the UI through data binding.

1. Choose the *Implementation* tab for the view controller **`BonusCalculationView.`**
2. In the **`onActionSubmit()`** method, add the following source code:

```
/** declared validating event handler */
public void onActionSubmit(com.sap.tc.webdynpro.progmodel.api.IWDCustomEvent wdEvent
)
{
  //@@begin onActionSubmit(ServerEvent)
     wdContext.currentMyCommandBeanElement().modelObject().execute();
  //@@end
}
```

3. In the **`onActionReset()`** method, add the following source code:

```
/** declared validating event handler */
public void onActionReset(com.sap.tc.webdynpro.progmodel.api.IWDCustomEvent wdEvent )
{
//@@begin onActionReset(ServerEvent)
wdContext.currentMyCommandBeanElement().setSsn("");
wdContext.currentMyCommandBeanElement().setMultiplier(0);
wdContext.currentMyCommandBeanElement().modelObject().setSsn("");
wdContext.currentMyCommandBeanElement().modelObject().setBonusAmount(0.0);
wdContext.currentMyCommandBeanElement().modelObject().setMessage("");
//@@end
}
```

Result

The Developer Studio updates and compiles the Java classes belonging to your project. (Note: Compilation only occurs if you are using the Workbench standard settings.) After you have done this, no more error messages should appear in your tasks view.

Deploying and Running the Sample

Now that you have reached this stage, you can start the fully developed example application in the Web Browser as described below.

Prerequisites

- ☐ You have made sure that the SAP J2EE Engine has been launched and that you are connected to an appropriate database instance of the SAP DB.

  To do this, refer to Starting and Stopping the SAP J2EE Engine.

- ☐ You have checked that the configuration settings for the J2EE server are entered correctly in the Developer Studio.

  To check the server settings, choose the menu path *Window* → *Preferences* → *SAP J2EE Engine.*

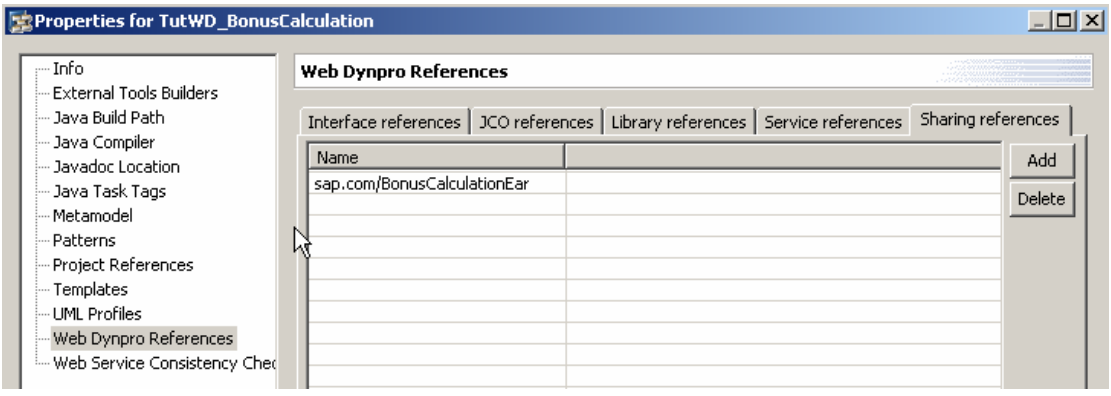
Procedure

Deploying the Tables and the J2EE Application

1. First of all, you need to deploy the SDA file of the *BonusDictionary* Project. To do so, switch to the *Dictionary Perspective*. In the project structure, select *BonusDictionary* and choose *Deploy* from the context menu.
2. Now you need to deploy the EAR file of the *BonusCalculationEJB* Project. Switch to the *J2EE Perspective*. In the project structure select *BonusCalculationEar* → *BonusCalculationEar.ear*. Select *Deploy to J2EE Engine* from the context menu.

Adding Sharing References for the Web Dynpro Project

3. In oder to access EJBs from the Web Dynpro application at runtime, we also need to add a sharing reference to the deployed Ear from within the Web Dynpro project. To do so, select the project properties and choose *Web Dynpro References* → *Sharing references*.
4. Choose the *Add* button and enter the fully qualified name (vendor name/Ear name) for the Ear required.



Building the Web Dynpro Project

5. Switch back to the *Web Dynpro Perspective*. Save the current status of the metadata for the project using the button in the upper application toolbar of your Developer Studio, if you have not already done so.
6. Open the context menu for the project node (**`TutWD_BonusCalculation_Init`**) in the Web Dynpro Explorer and choose *Rebuild Project.* Make sure that the *Tasks* view does not display any errors for your project. You can ignore any warning messages for *labelFor* properties that have not been set.

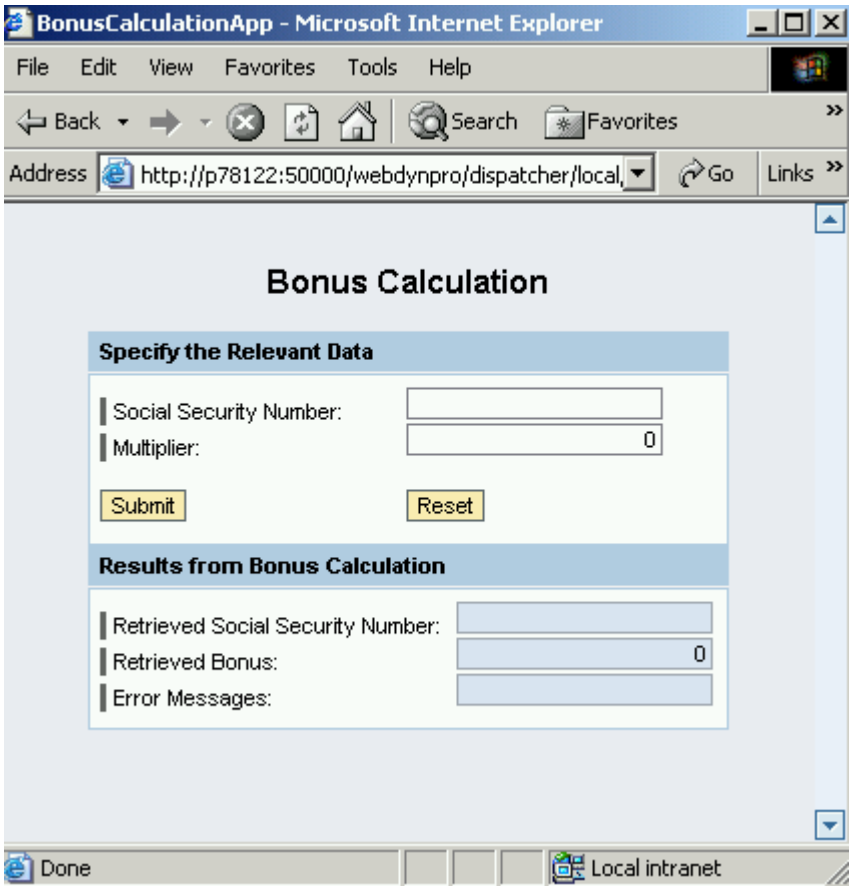
Deploying and Launching the Web Dynpro Application

7. In the Web Dynpro Explorer, open the context menu for the application object **`BonusCalculationApp`**. Choose *Deploy new archive and run*.

Result

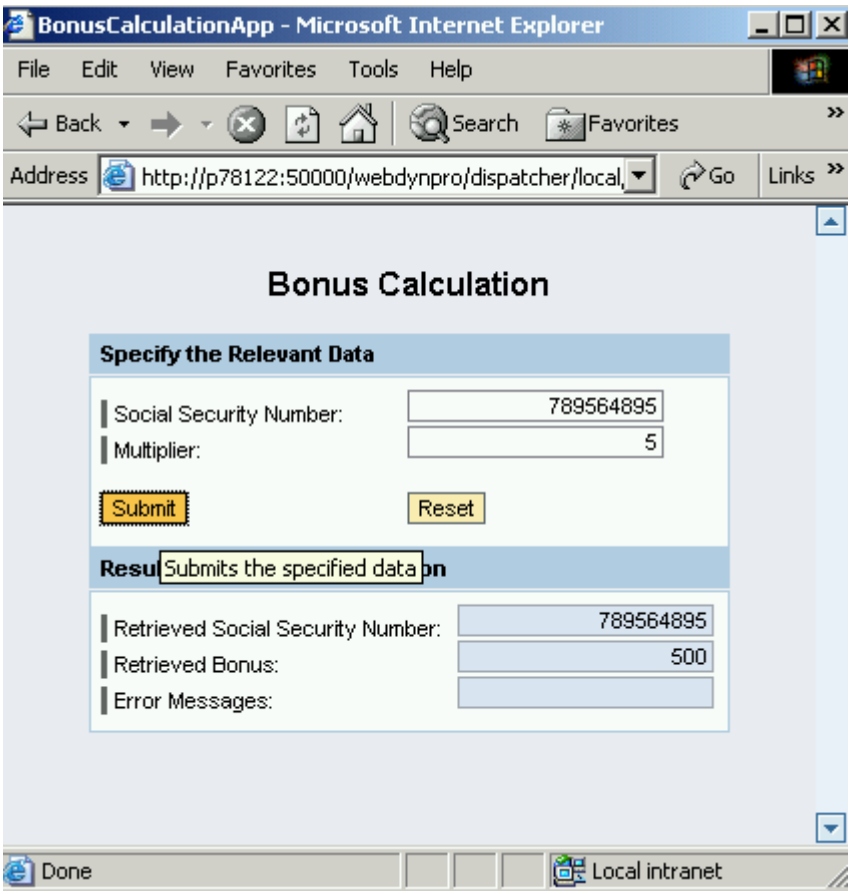
The Developer Studio performs the deployment process in one single step, based on an automatically generated Enterprise Archive file, and then automatically launches your application in the Web Browser.

Test your newly developed Web Dynpro application by entering data into the input fields and then clicking the button *Submit*.



After you have triggered a server roundtrip (here you communicate using *MyCommandBean*), the returned result is displayed on the user interface in the Web Browser.

If the entered social security number has not been entered, no error message is displayed in the message field yet.



If you enter a social security number that already exists, a *Duplicate Key Exception* message is displayed in the message field.