EJB and Java Bean Model Usage in Web Dynpro Java

Applies to:

SAP NetWeaver 7.0 SP13 and onwards. For more information, visit the [Web Dynpro Java]() homepage.

Summary

Majority of the Web Dynpro applications will be database driven and requires integration with any RDBMS like Oracle database, DB2, etc. This document covers all the steps required to implement the business logic in Enterprise Java Beans (EJB), persistence using Java Database Connectivity (JDBC) and how to implement the SAP recommended approach of importing Java Bean Model to a Web Dynpro application.

Author: Subash Mathews Thomas

Company: Qatar Petroleum, Doha, Qatar.

Created on: 06/07/2009

Author Bio

Subash Mathews Thomas is currently working as Systems Engineer in Qatar Petroleum and has over 10 years of extensive software development experience including more than nine years developing large distributed applications using J2EE technologies. Subash is a SUN certified and IBM certified Java professional and he has been closely involved with the estimation, design, and development of various large and medium Java EE-based enterprise application development projects for large client organizations across the world. During the last few years, Subash has experience working in the SAP NetWeaver technologies like WebDynpro Java, Guided Procedures, etc.

Table of Contents

Scope of this Document

This document serves as a guideline cum best practices for developing database driven e-Applications using Web Dynpro and EJB. This document does not cover all the theories of Enterprise Java Beans (EJB) and assumes that you have a basic knowledge of EJB programming model and basic experience with webdynpro application.

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

- Write a simple Data Access Command Bean (Java Bean) based on predefined business logic.
- Write an EJB (Stateless Session Bean) and implement the persistence with JDBC.
- 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

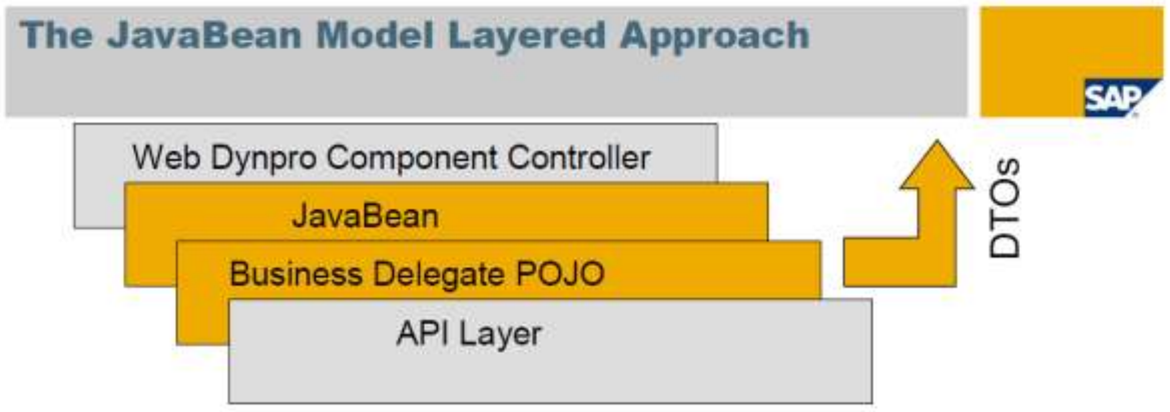
Systems, installed applications, and authorizations

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

Knowledge

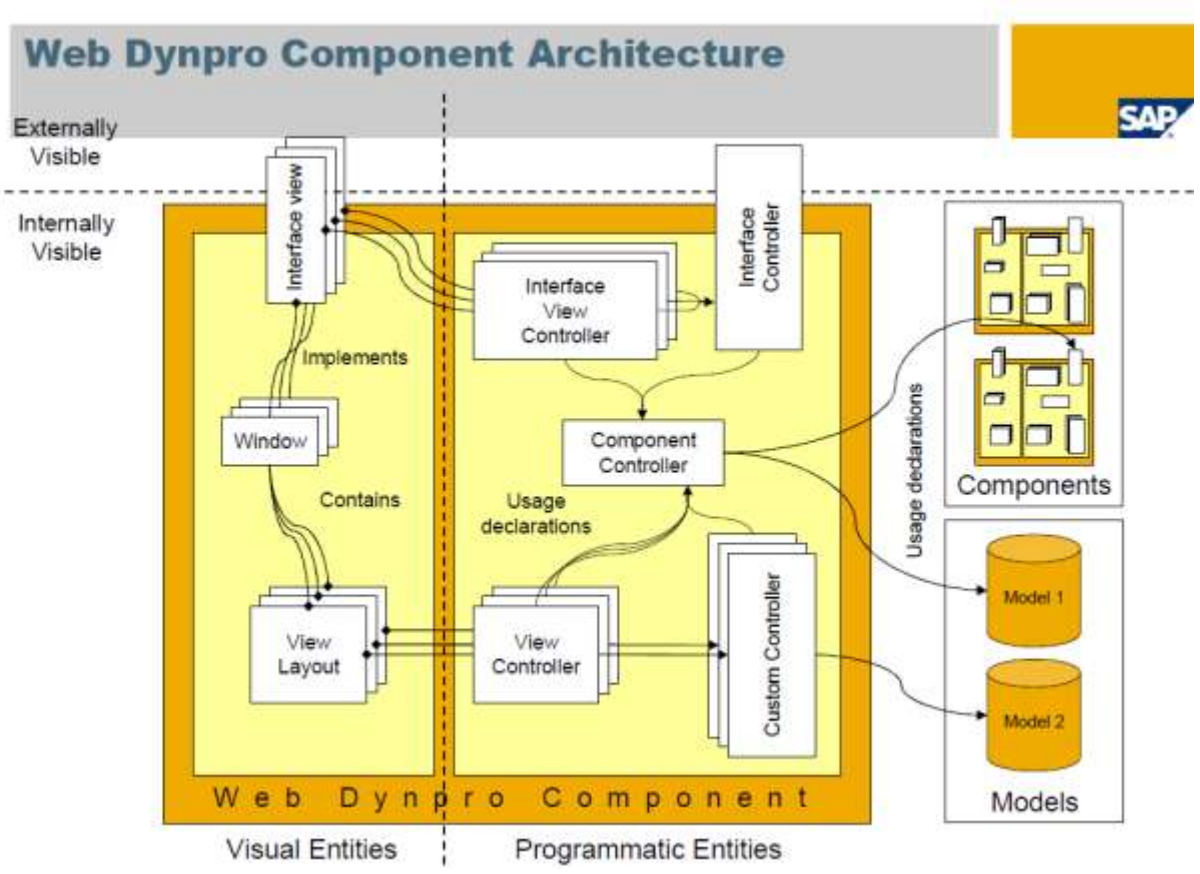
- You have acquired some basic experience with Web Dynpro applications.
- You have basic knowledge of the EJB programming model.
- You are experience working with the SAP NetWeaver Developer Studio.

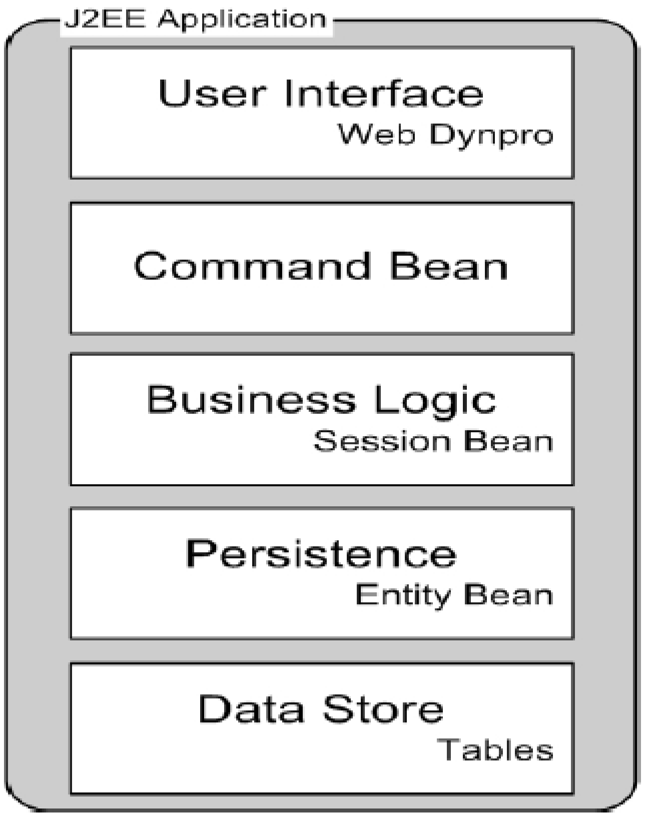
Benefits of Java Bean Model Layered Approach


The JavaBean Model Layered Approach
SAP
Web Dynpro Component Controller
JavaBean
Business Delegate POJO
API Layer
DTOs

- Clear segregation of responsibilities.
- Clear rules of behavior.
- Reduces coupling of layers – presentation vs. business layer.
- Hides details of underlying layer, such as EJB lookup logic, business implementation, etc.
- Using Command Bean Pattern exposes commands in addition to properties (Java Bean). Methods take no arguments, context is JavaBean state.
- Using Java Bean Models mapped to context variables in Web Dynpro will make the life of the developer much easier as he does not have to populate manually the context variables in Web Dynpro Component and View controllers.

Implementation Procedure

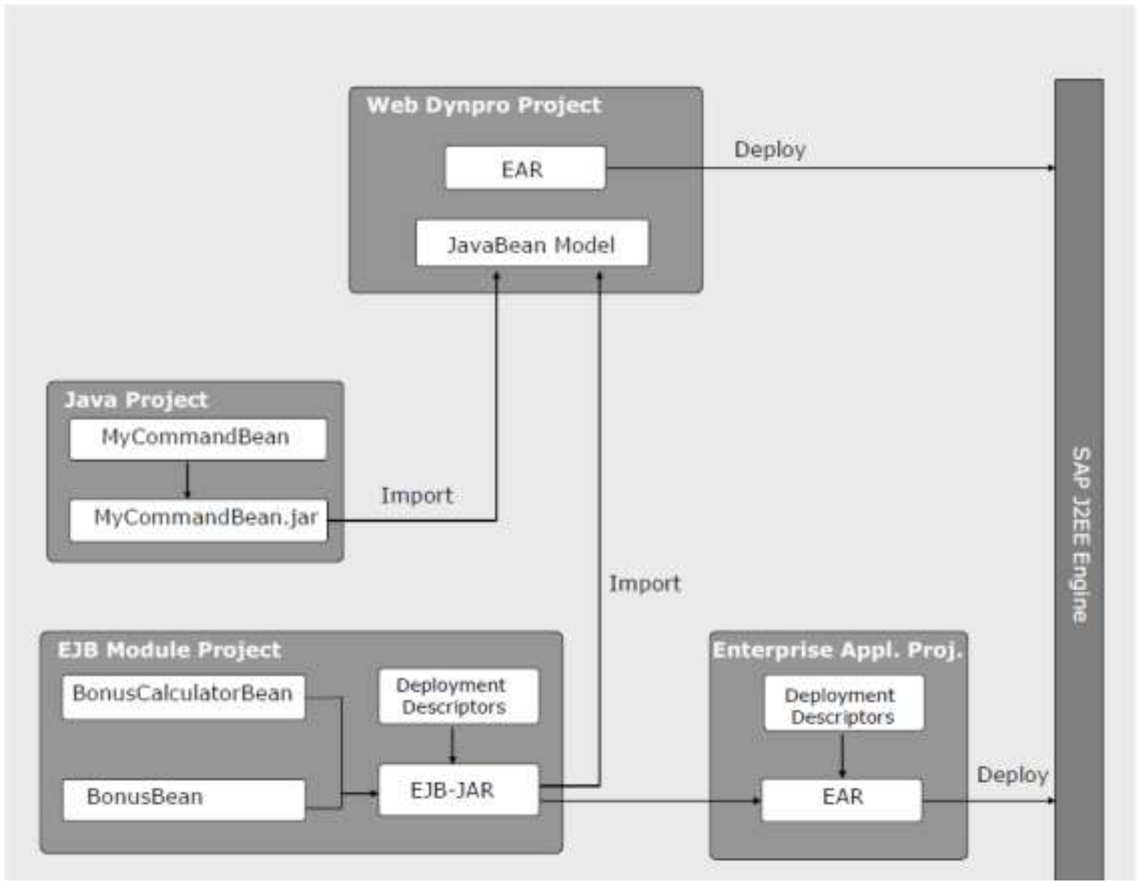




The example scenario in this document consists of different components: Enterprise JavaBeans (EJBs), where you implement the business logic; a Data Access Command Bean (JavaBean), which acts as the java bean model; and a Web Dynpro Project. In the Web Dynpro, the command bean will be imported into the Web Dynpro project using the JavaBean importer wizard. This will be more explained further in this document.

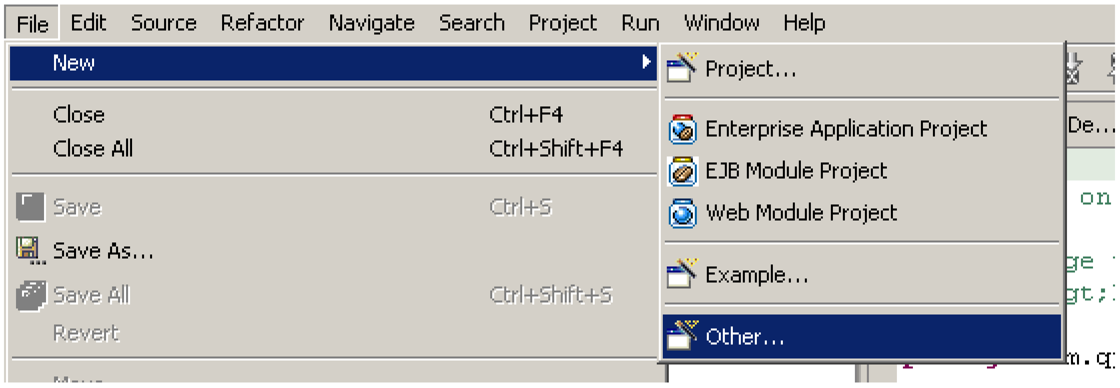
Different Types of Projects.

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

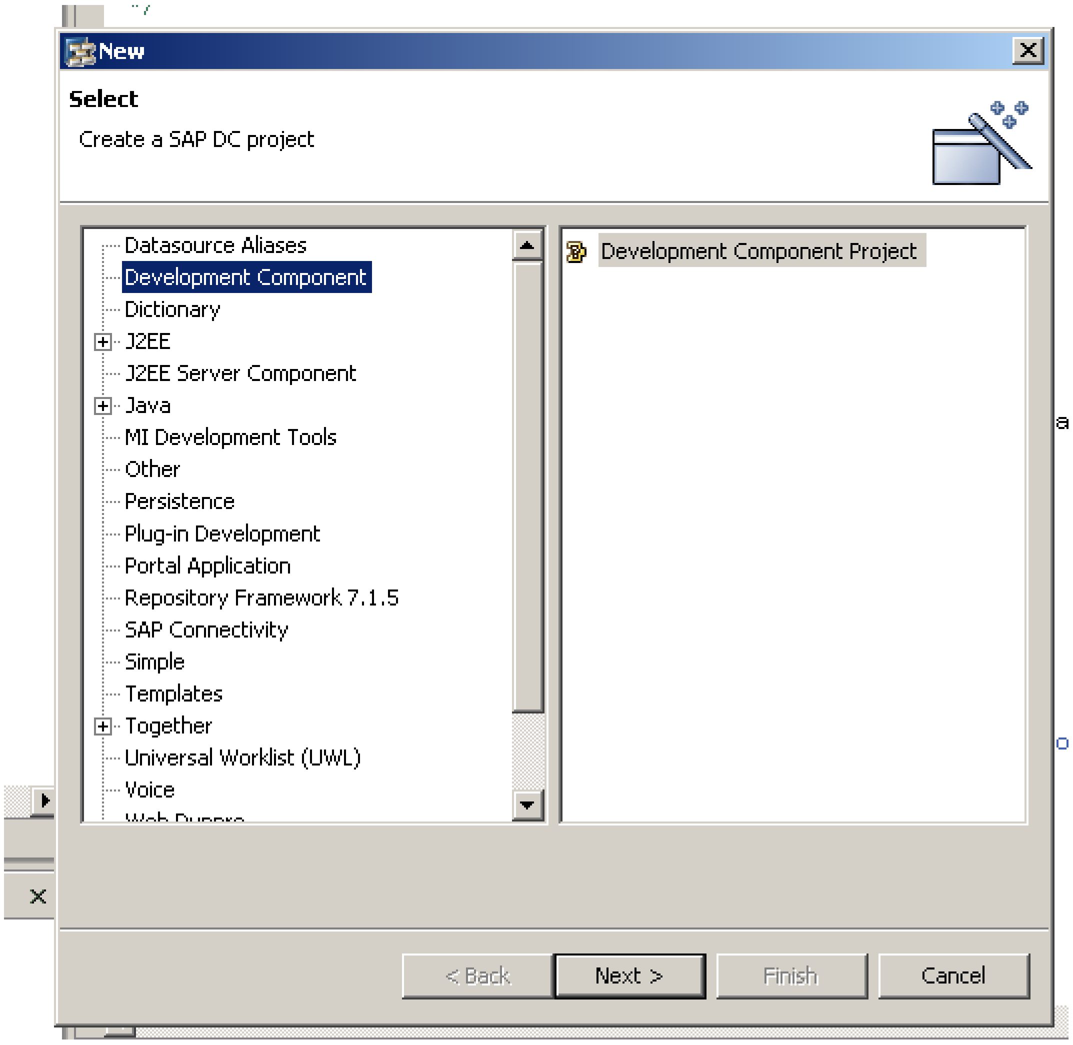


Create EJB Project

Create a new development component of type “Development Component” and select the next button:

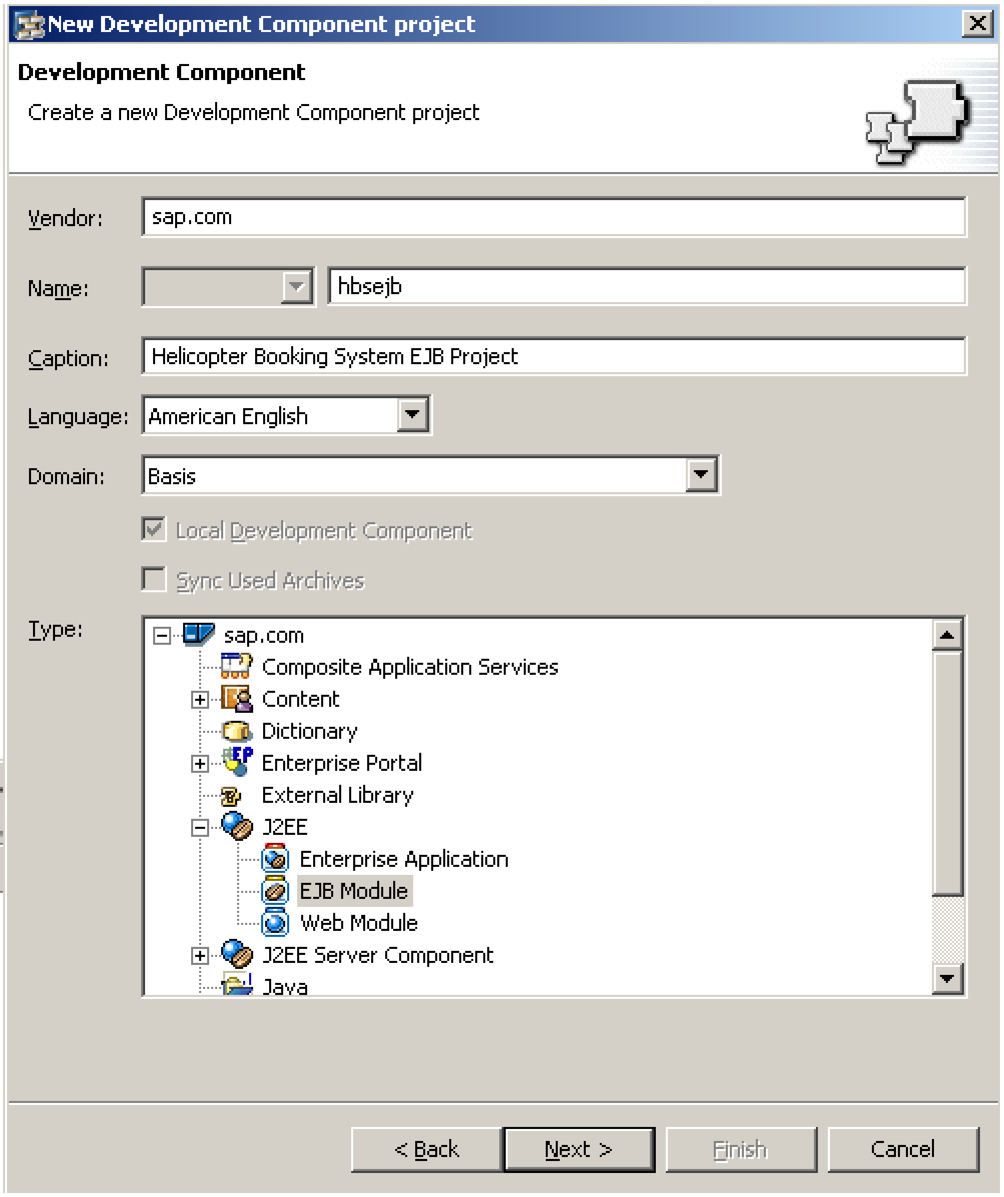


Select “Other” from the File Menu.



Create a DC Project.

Select J2EE ->EJB Module for the type of development component and select the Next button:



New Development Component project

Development Component

Create a new Development Component project

Vendor: sap.com

Name: hbsejb

Caption: Helicopter Booking System EJB Project

Language: American English

Domain: Basis

☑ Local Development Component

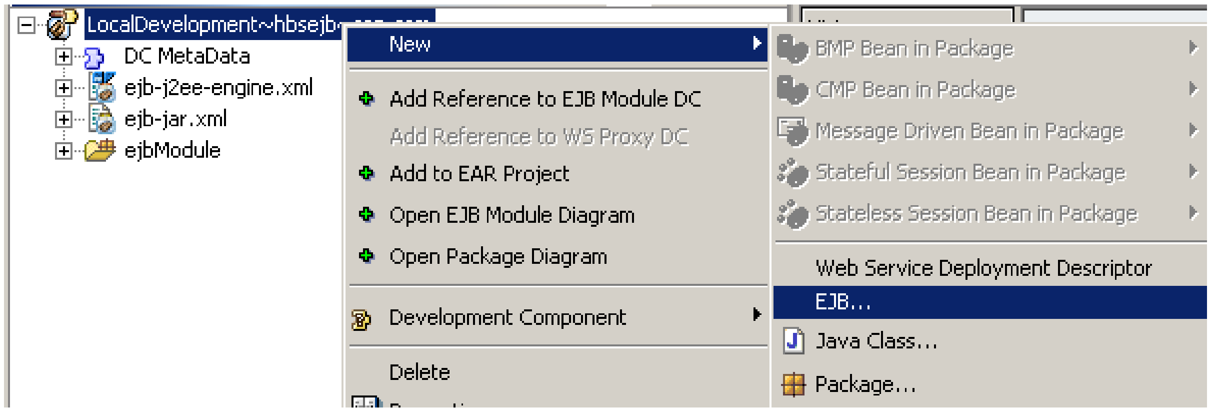
☐ Sync Used Archives

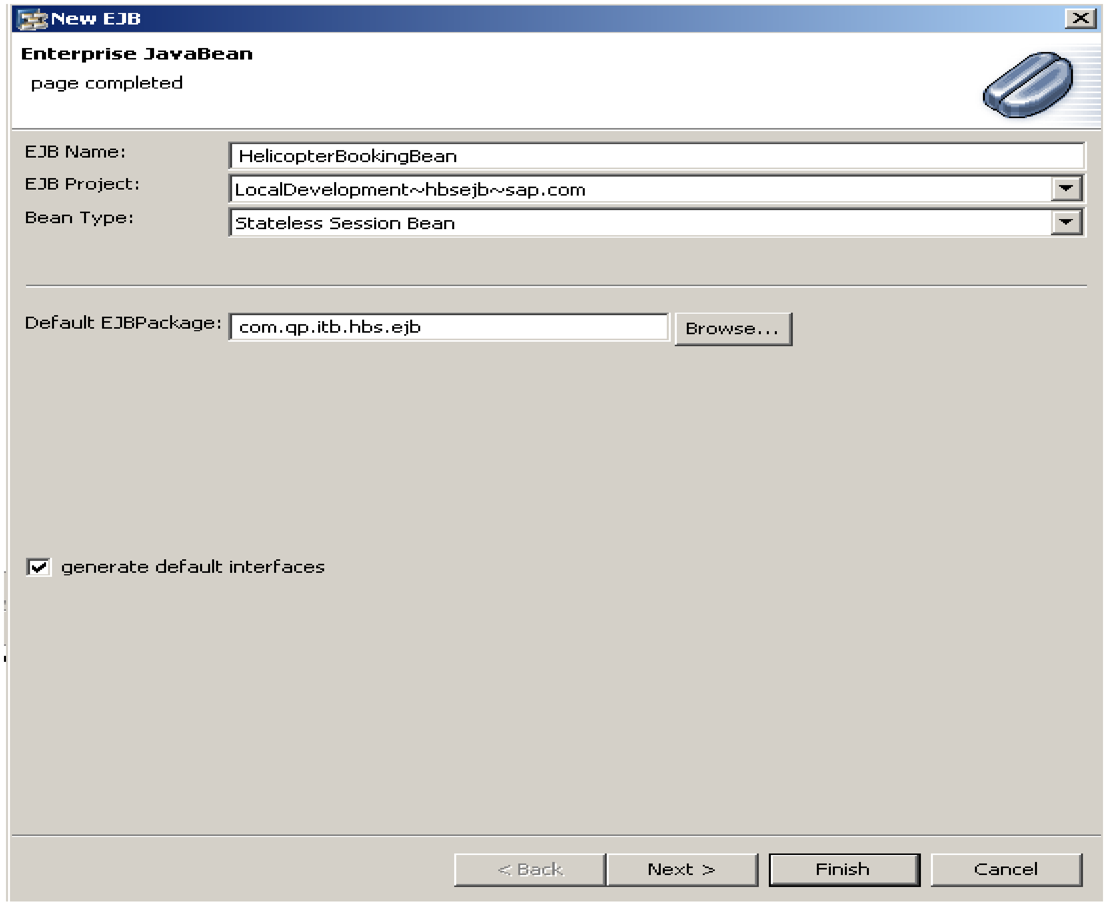
Type:

- sap.com
  - Composite Application Services
  - Content
  - Dictionary
  - Enterprise Portal
  - External Library
  - J2EE
    - Enterprise Application
    - EJB Module
    - Web Module
  - J2EE Server Component
  - Java

Name the component <project-acronym>ejb and select the Next button.

Create a new EJB class as follows:




New EJB

Enterprise JavaBean

page completed

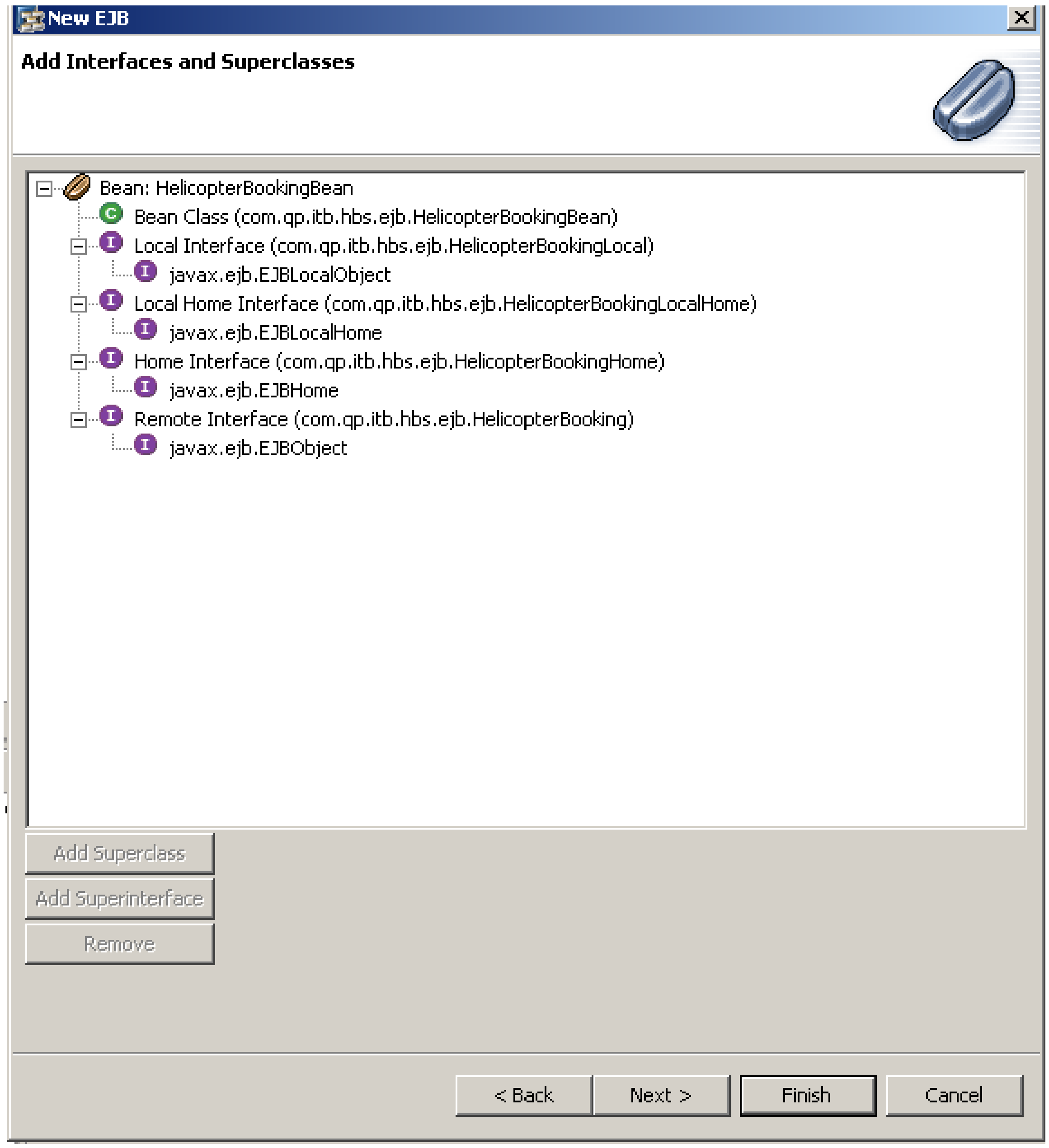
EJB Name: HelicopterBookingBean

EJB Project: LocalDevelopment~hbsejb~sap.com

Bean Type: Stateless Session Bean

Default EJBPackage: com.qp.itb.hbs.ejb

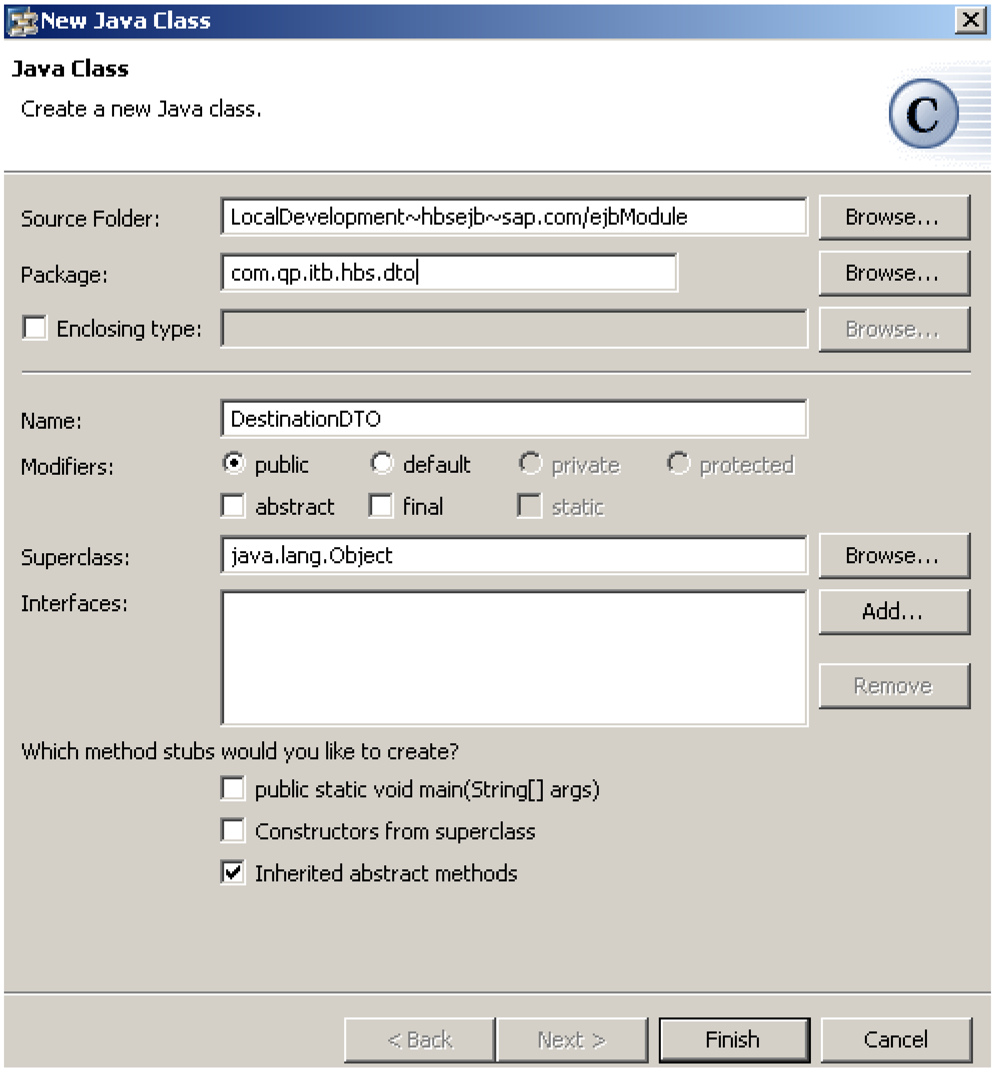
[✓] generate default interfaces



Bean: HelicopterBookingBean
- Bean Class (com.qp.itb.hbs.ejb.HelicopterBookingBean)
- Local Interface (com.qp.itb.hbs.ejb.HelicopterBookingLocal)
  - javax.ejb.EJBLocalObject
- Local Home Interface (com.qp.itb.hbs.ejb.HelicopterBookingLocalHome)
  - javax.ejb.EJBLocalHome
- Home Interface (com.qp.itb.hbs.ejb.HelicopterBookingHome)
  - javax.ejb.EJBHome
- Remote Interface (com.qp.itb.hbs.ejb.HelicopterBooking)
  - javax.ejb.EJBObject

Click on Finish button.

Create Data Transfer Object (DTO).



Source Folder: LocalDevelopment~hbsejb~sap.com/ejbModule

Package: com.qp.itb.hbs.dto

☐ Enclosing type:

Name: DestinationDTO

Modifiers: ● public ○ default ○ private ○ protected

☐ abstract ☐ final ☐ static

Superclass: java.lang.Object

Interfaces:

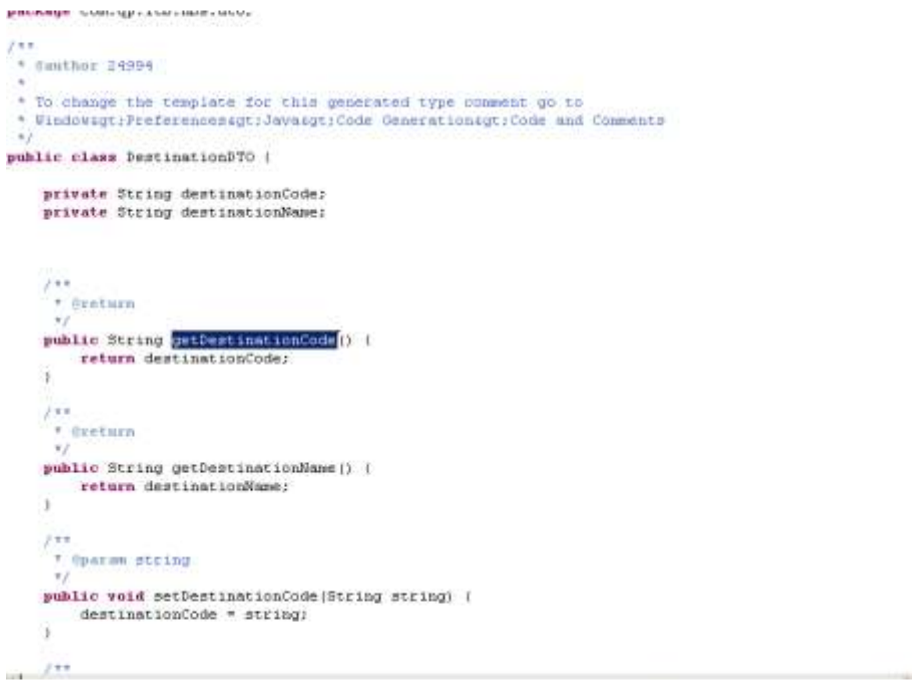
Which method stubs would you like to create?

☐ public static void main(String[] args)

☐ Constructors from superclass

☑ Inherited abstract methods

Click on Finish button.

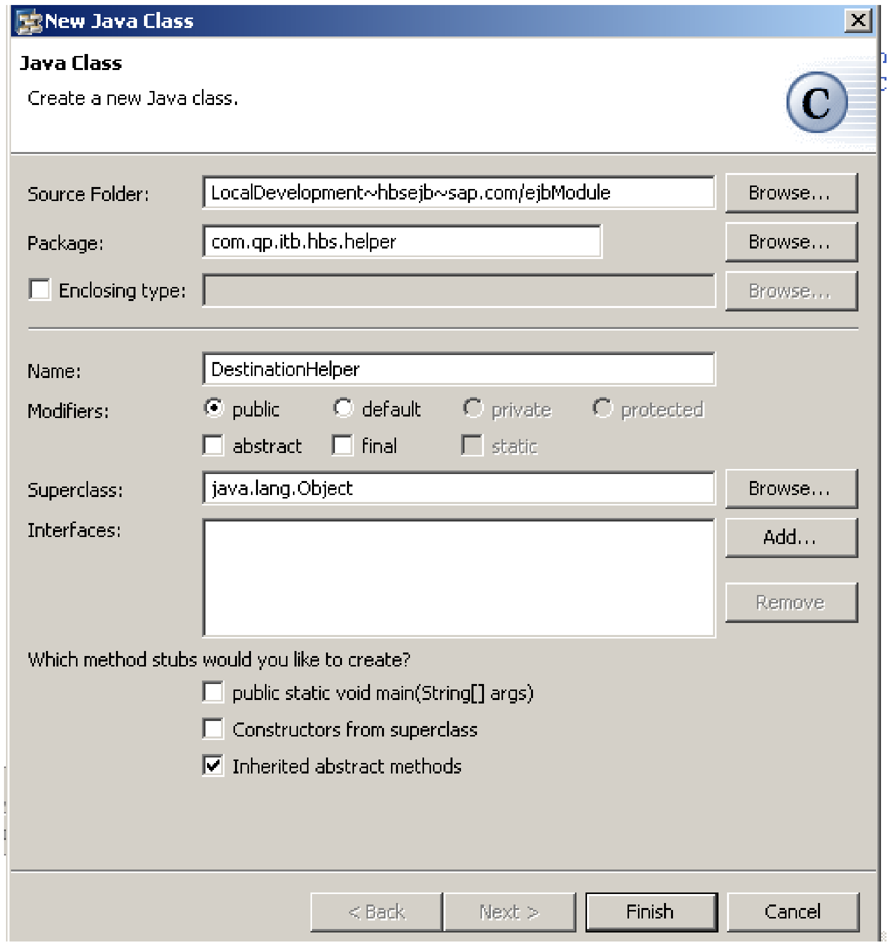


```
package com.qp.itb.hbs.dto;

/**
 * @author 24994
 *
 * To change the template for this generated type comment go to
 * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
 */
public class DestinationDTO {

    private String destinationCode;
    private String destinationName;


    /**
     * @return
     */
    public String getDestinationCode() {
        return destinationCode;
    }

    /**
     * @return
     */
    public String getDestinationName() {
        return destinationName;
    }

    /**
     * @param string
     */
    public void setDestinationCode(String string) {
        destinationCode = string;
    }

    /**
```

Code snippet of DestinationDTO class.

Create a new Java Class called DestinationHelper.

This is the class where you will connect to the database and write JDBC codes.



Source Folder: LocalDevelopment~hbsejb~sap.com/ejbModule

Package: com.qp.itb.hbs.helper

☐ Enclosing type:

Name: DestinationHelper

Modifiers: [✓ public] default private protected

☐ abstract ☐ final ☐ static

Superclass: java.lang.Object

Interfaces:

Which method stubs would you like to create?

☐ public static void main(String[] args)

☐ Constructors from superclass

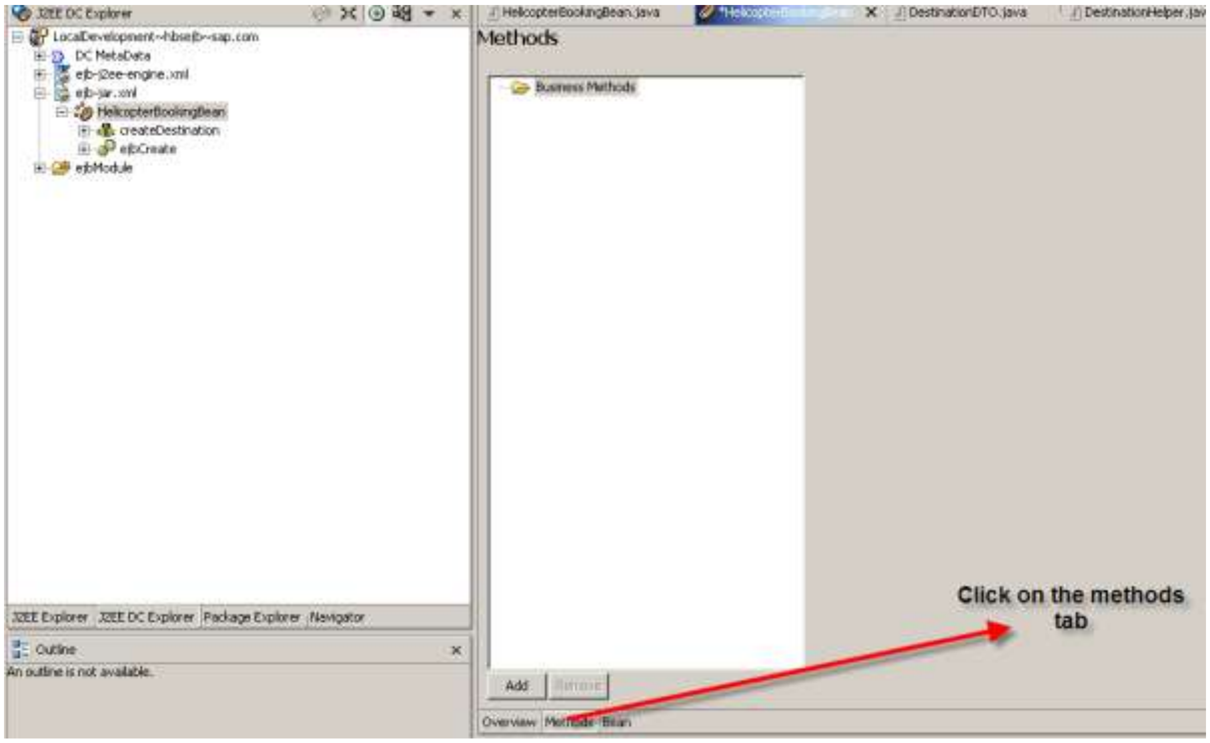
☑ Inherited abstract methods

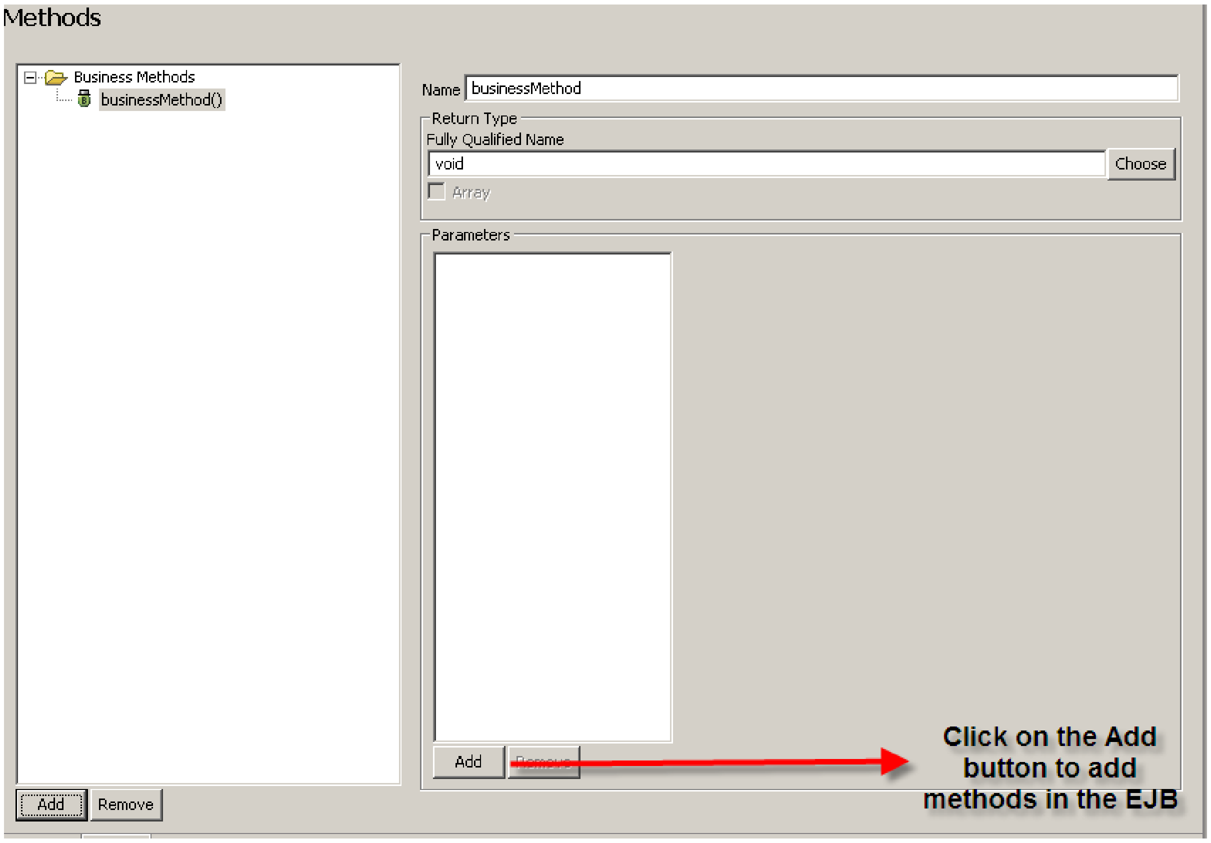
Click on Finish button.

```
// Sample Code for Helper class

    public class DestinationHelper {
       public void createNewDestination (String destinationCode,
                                                             String destinationName)
                                                             throws EJBException{
              Connection conn = null;
              String destination = null;
              Statement stmt = null;
              try {
                     //Getting the datasource connection
                     InitialContext ctx = new InitialContext();
                     DataSource dataSource = (DataSource) ctx.lookup("jdbc/HBIMS");
                     conn = dataSource.getConnection();
                     if (conn == null) {
                            destination = "connection is null";
                            throw new EJBException(destination);
                     }
                     conn.setAutoCommit(false);
                     stmt = conn.createStatement();
                     String sql ="INSERT INTO hbs_destinations (destination_code,
                     destination_name) VALUES ('"
                                   + destinationCode + "','"+ destinationName+ "')";
                     int k = stmt.executeUpdate(sql);
                     conn.commit();

              } catch (SQLException sex) {
                     sex.printStackTrace();
                     throw new EJBException(sex.getMessage());
              } catch (NamingException ex) {
                     ex.printStackTrace();
                     throw new EJBException(ex.getMessage());
                           conn.rollBack();
              } finally {
                     try {
                            if (stmt != null)
                                   stmt.close();
                            if (conn != null)
                                   conn.close();
                     } catch (SQLException se) {
                            se.printStackTrace();
                     }
              }
       }
    }
```

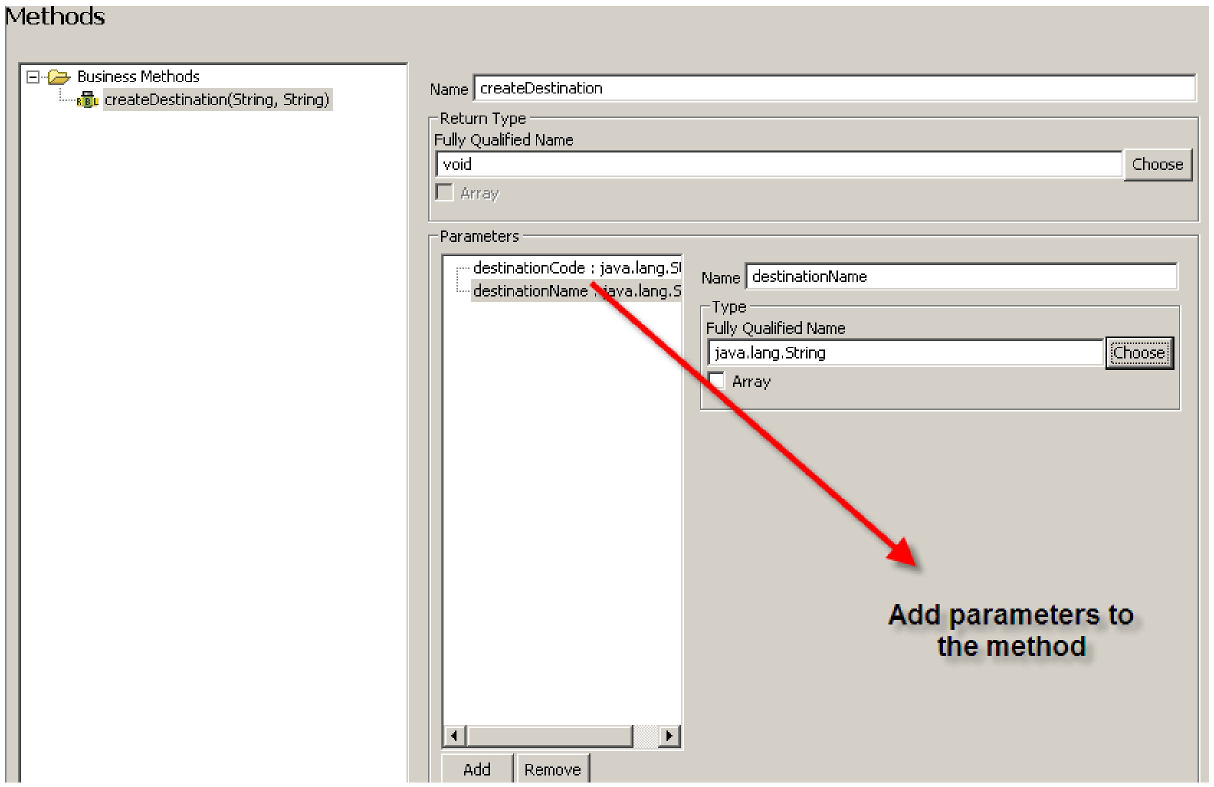
Above is the code snippet for DestinationHelper class.

Implement EJB Business method.





Name: businessMethod
Fully Qualified Name: void


Methods
Business Methods
createDestination(String, String)
Return Type
Choose
Array
Parameters
destinationCode : java.lang.St
destinationName : java.lang.S
Type
Choose
Array
Add
Remove
Add parameters to the method

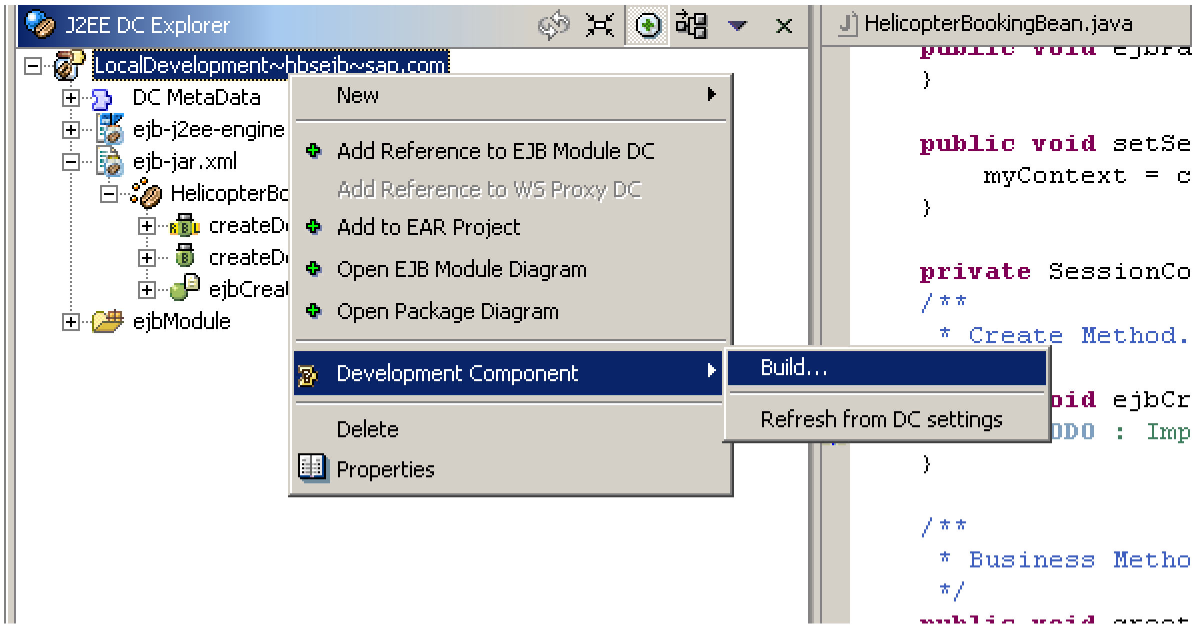
Name: createDestination
Fully Qualified Name: void

Name: destinationName
Fully Qualified Name: java.lang.String

Implement the business method in the EJB class.

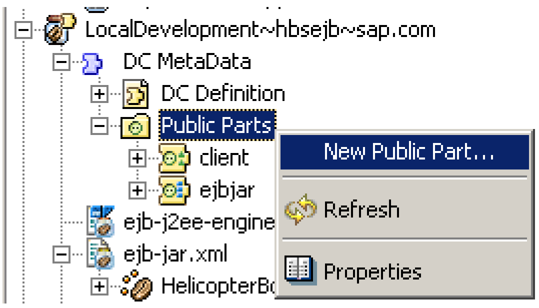
```
/**
 * Business Method.
 */
public void createDestination(DestinationDTO destination) {
    // TODO : Implement

    DestinationHelper helper = new DestinationHelper();
    try {
        helper.createNewDestination(destination.getDestinationCode(),
                                    destination.getDestinationName());


    }
    catch(EJBException e) {
        throw e;
    }
}
```

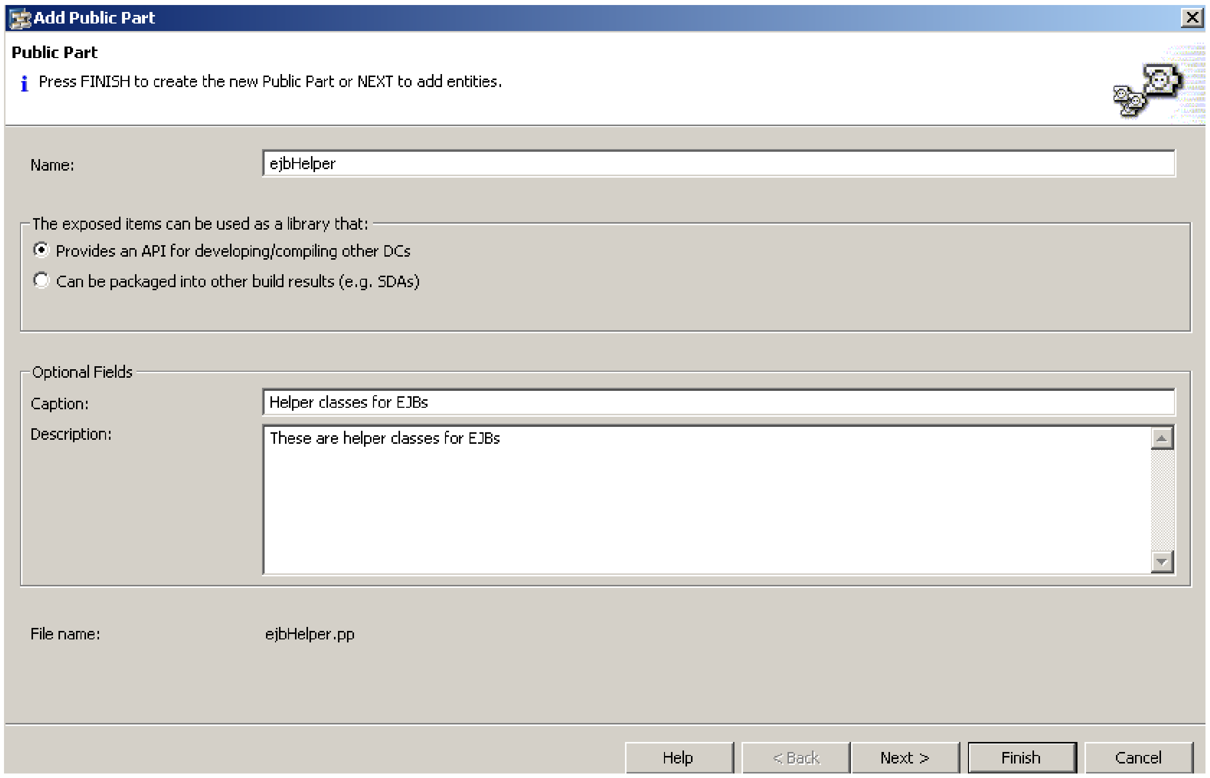
Build the EJB Project



```
public void setSe
    myContext = c
}

private SessionCo
/**
 * Create Method.
```

Add the helper classes to the public parts, so that other DC projects can access the helper class.





Public Part

Press FINISH to create the new Public Part or NEXT to add entities.

Name: ejbHelper

The exposed items can be used as a library that:

[✓ Provides an API for developing/compiling other DCs]

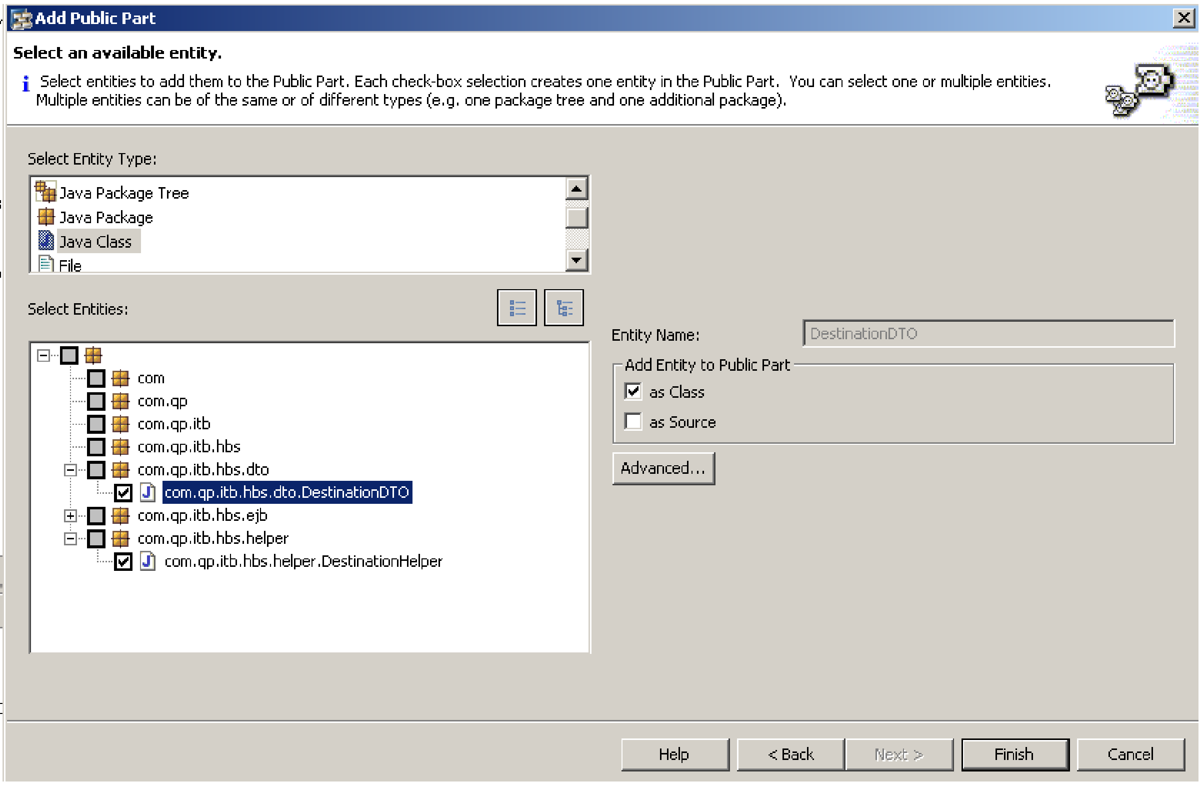
Can be packaged into other build results (e.g. SDAs)

Optional Fields

Caption: Helper classes for EJBs

Description: These are helper classes for EJBs

File name: ejbHelper.pp



Select an available entity.

Select entities to add them to the Public Part. Each check-box selection creates one entity in the Public Part. You can select one or multiple entities. Multiple entities can be of the same or of different types (e.g. one package tree and one additional package).

Select Entity Type:

- Java Package Tree
- Java Package
- [✓ Java Class]
- File

Select Entities:

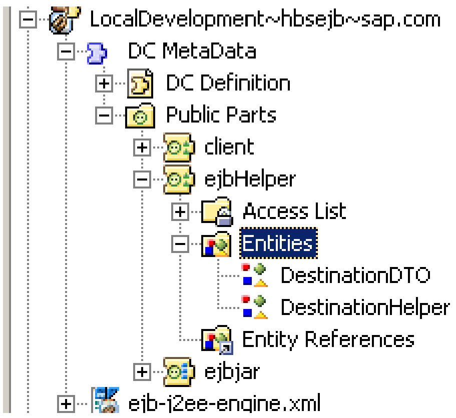
- com
- com.qp
- com.qp.itb
- com.qp.itb.hbs
- com.qp.itb.hbs.dto
  - [✓] com.qp.itb.hbs.dto.DestinationDTO
- com.qp.itb.hbs.ejb
- com.qp.itb.hbs.helper
  - [✓] com.qp.itb.hbs.helper.DestinationHelper

Entity Name: DestinationDTO

Add Entity to Public Part

- [✓] as Class
- [ ] as Source

Click on Finish button.

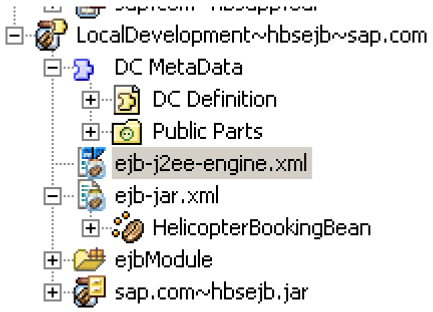


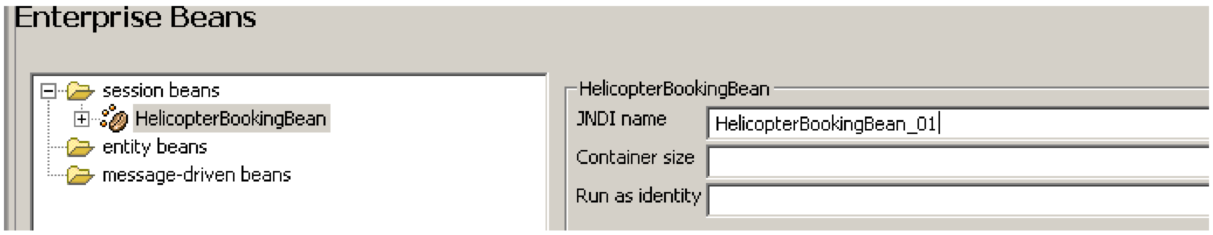
So now you have a public part called ejbHelper which has all the helper classes.

EJBs need not be exposed like this as they are public by default.

Create a JNDI name for the EJB.

Click on the ejb-j2ee-engine.xml as shown below.

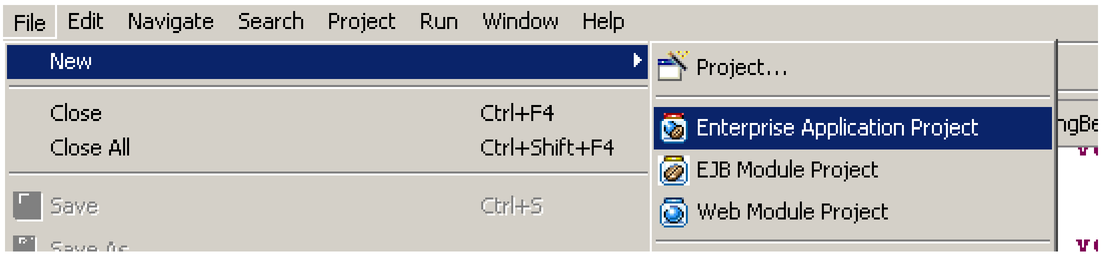


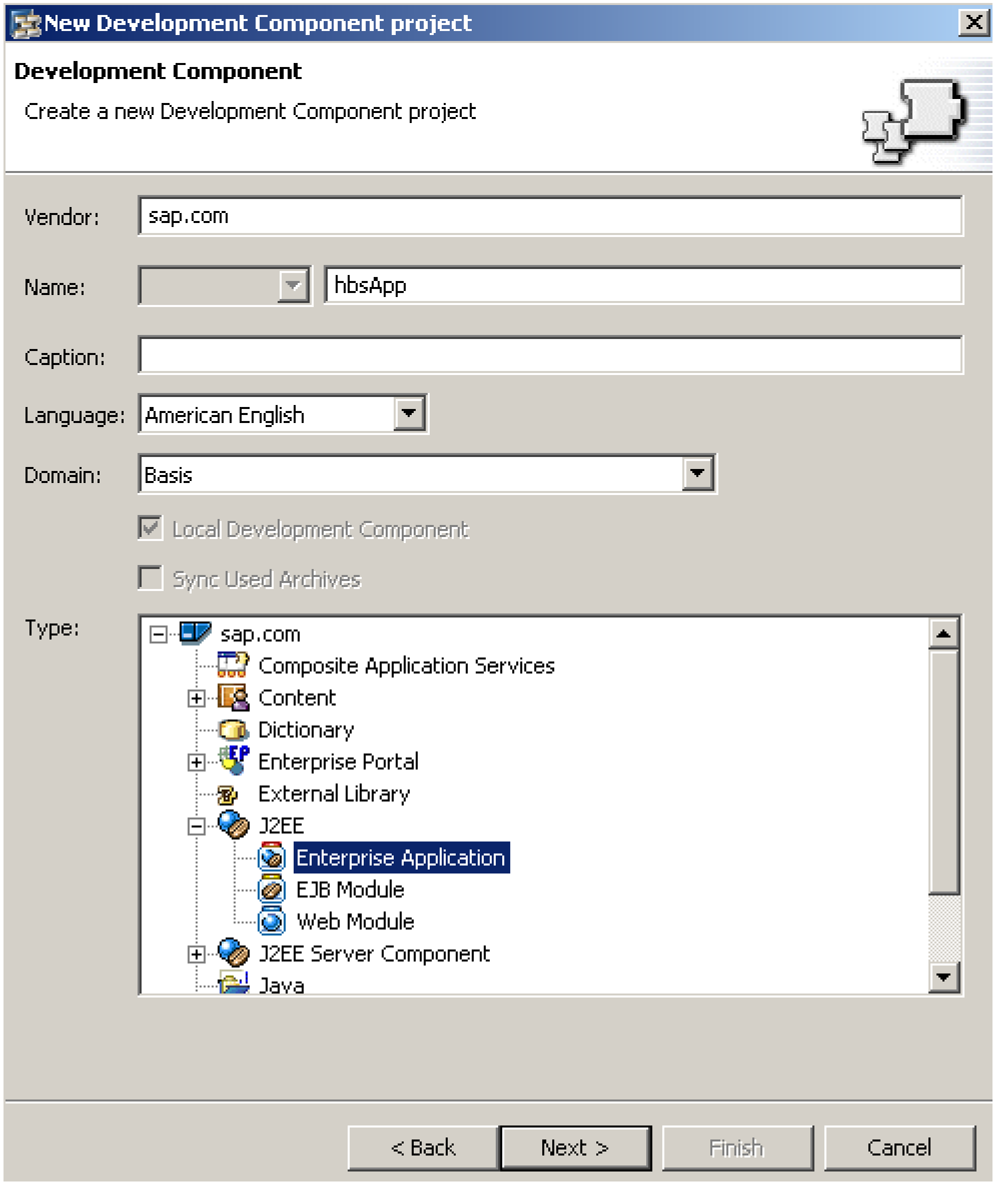


JNDI name: HelicopterBookingBean_01
Container size:
Run as identity:

Save the changes.

Create an Enterprise application project to deploy the EJB to J2EE server.





New Development Component project

Development Component

Create a new Development Component project

Vendor: sap.com

Name: hbsApp

Caption:

Language: American English

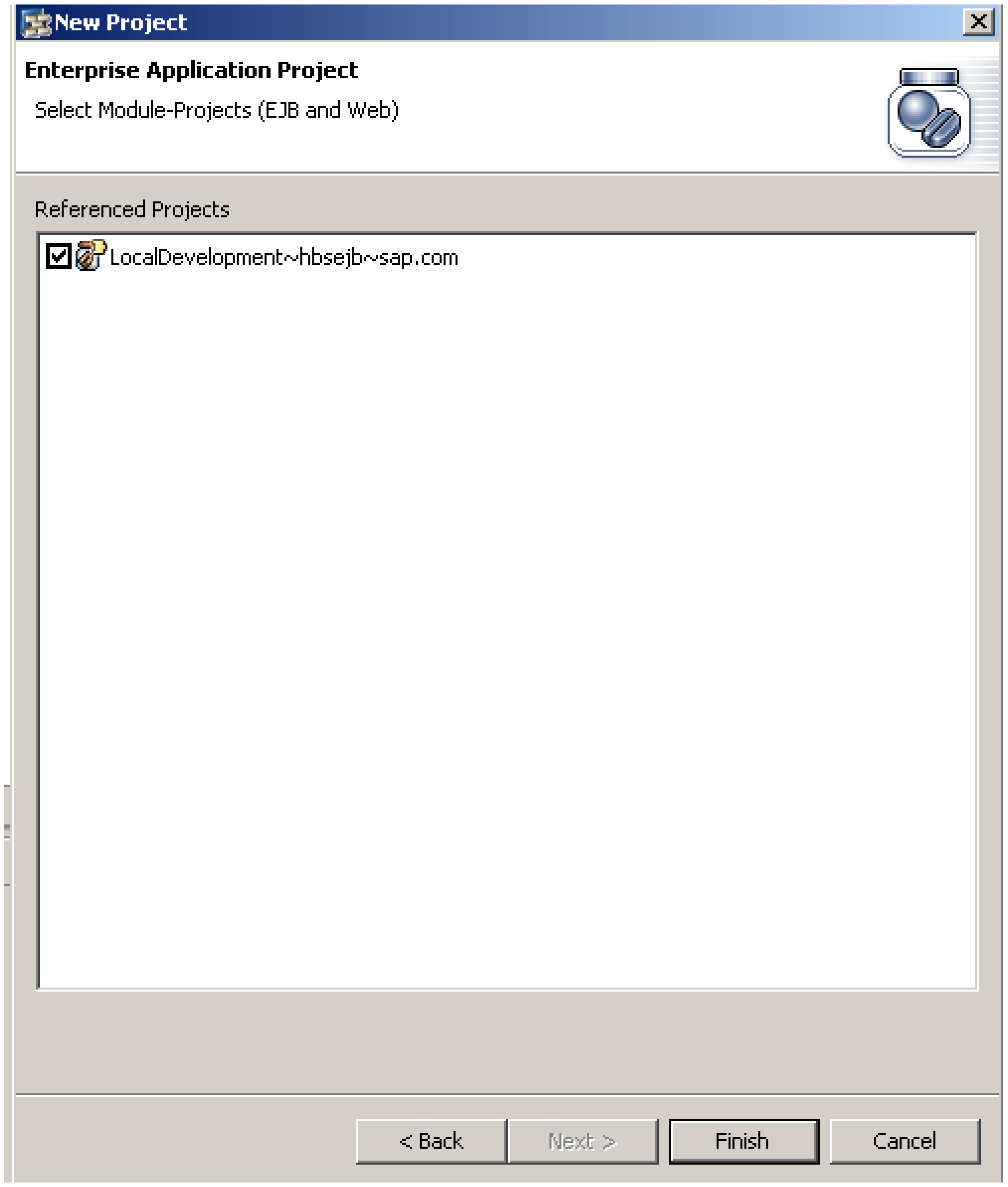
Domain: Basis

[✓] Local Development Component

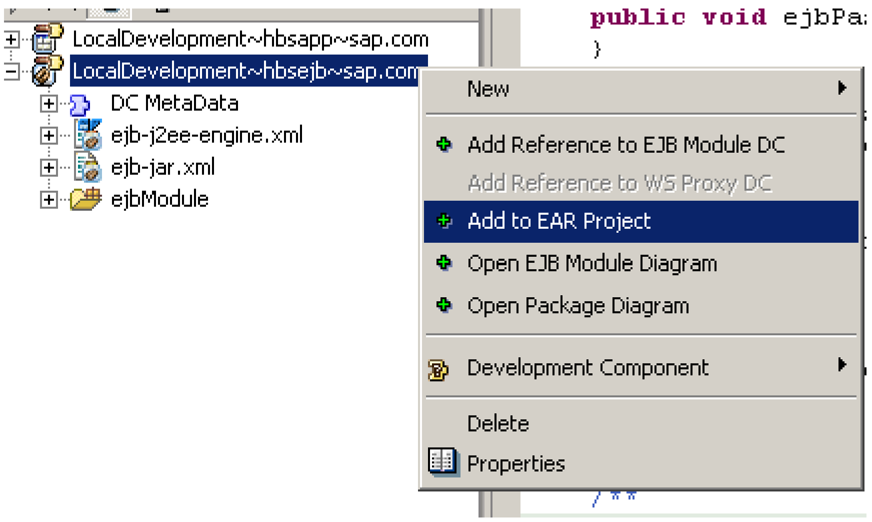
[ ] Sync Used Archives

Type:

- sap.com
  - Composite Application Services
  - Content
  - Dictionary
  - Enterprise Portal
  - External Library
  - J2EE
    - Enterprise Application
    - EJB Module
    - Web Module
  - J2EE Server Component
  - Java

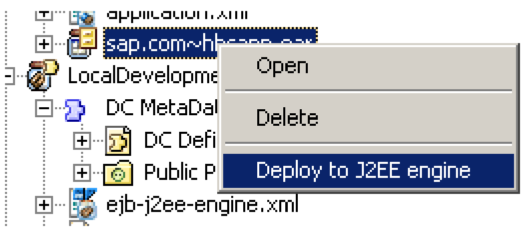

New Project
Enterprise Application Project
Select Module-Projects (EJB and Web)
Referenced Projects
LocalDevelopment~hbsejb~sap.com
< Back
Next >
Finish
Cancel

Click on Finish button.



Build and deploy the EAR file to J2EE Server.

Right click on the ear file and deploy it to the server.

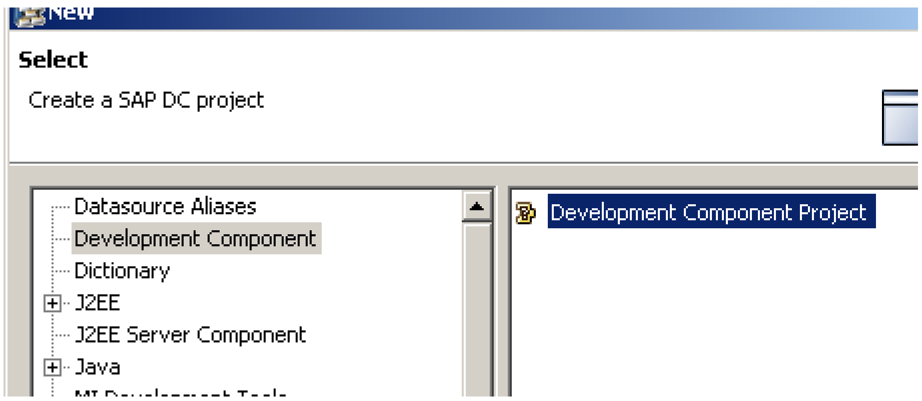


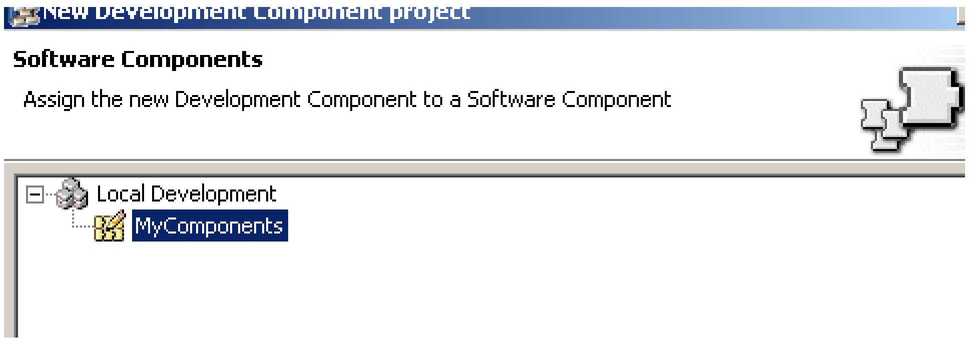
Create Model Project (Java Project)

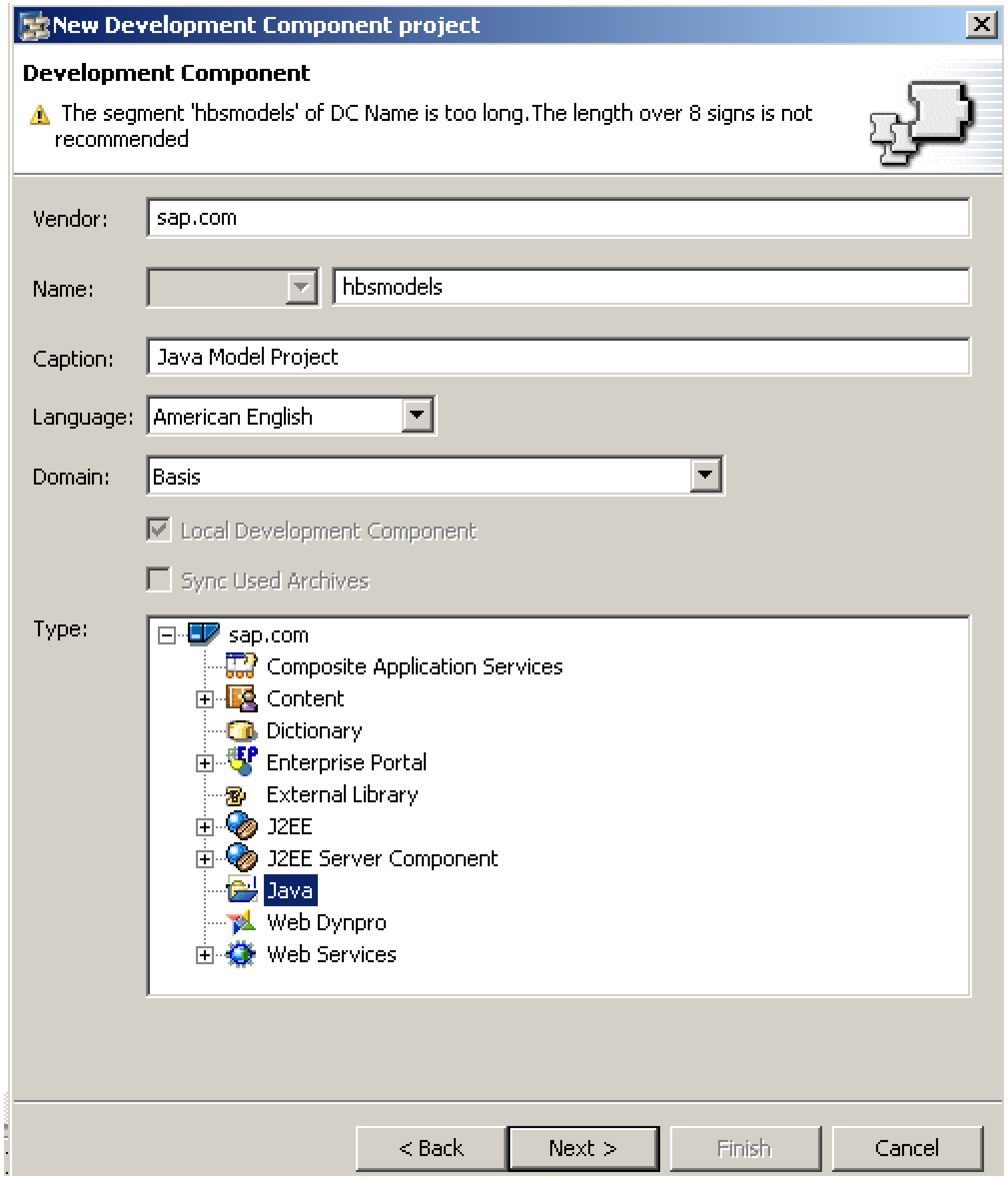
In this procedure you will create a separate Java project that serves as a container for the Data Access Command Beans and will be imported in the Web Dynpro model. If you already have a model created, this exercise can be skipped.

Procedure

Switch to the Java Perspective and choose File → New → Project to start the New Project wizard.




New Development Component project
Software Components
Assign the new Development Component to a Software Component
Local Development
MyComponents



New Development Component project

Development Component

⚠ The segment 'hbsmodels' of DC Name is too long.The length over 8 signs is not recommended

Vendor: sap.com

Name: hbsmodels

Caption: Java Model Project

Language: American English

Domain: Basis

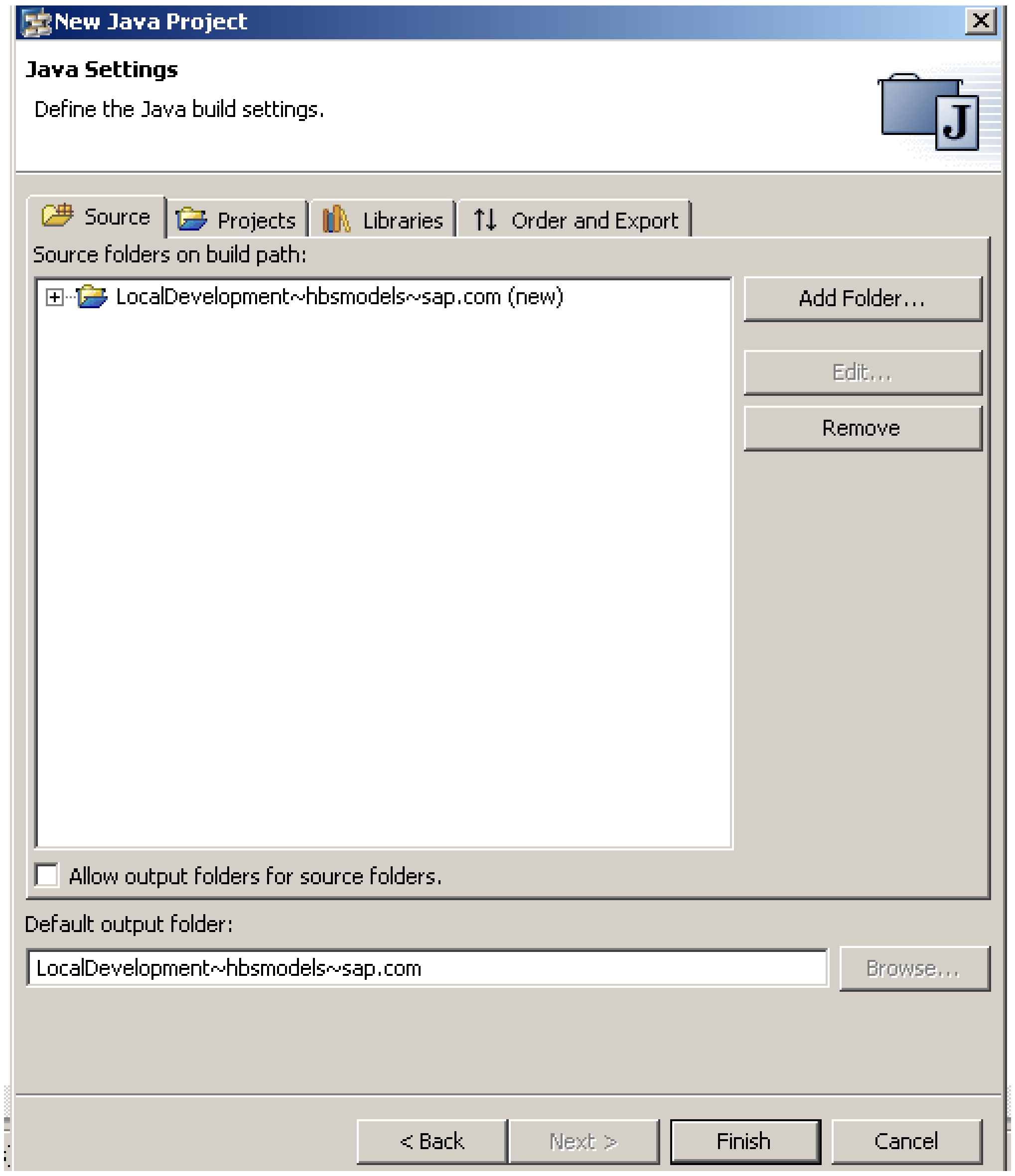
[✓] Local Development Component

[ ] Sync Used Archives

Type:

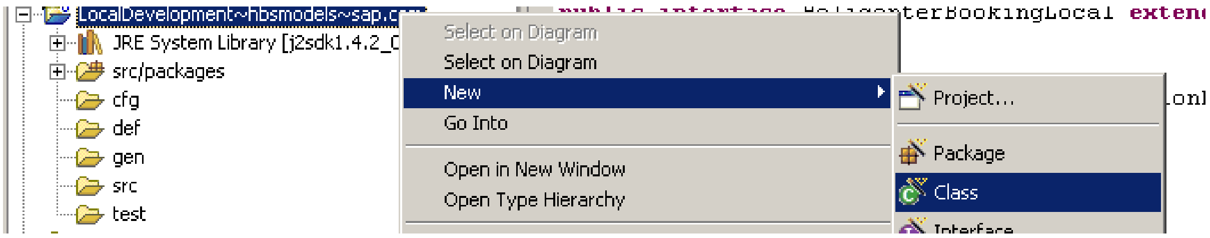
- sap.com
  - Composite Application Services
  - Content
  - Dictionary
  - Enterprise Portal
  - External Library
  - J2EE
  - J2EE Server Component
  - Java
  - Web Dynpro
  - Web Services

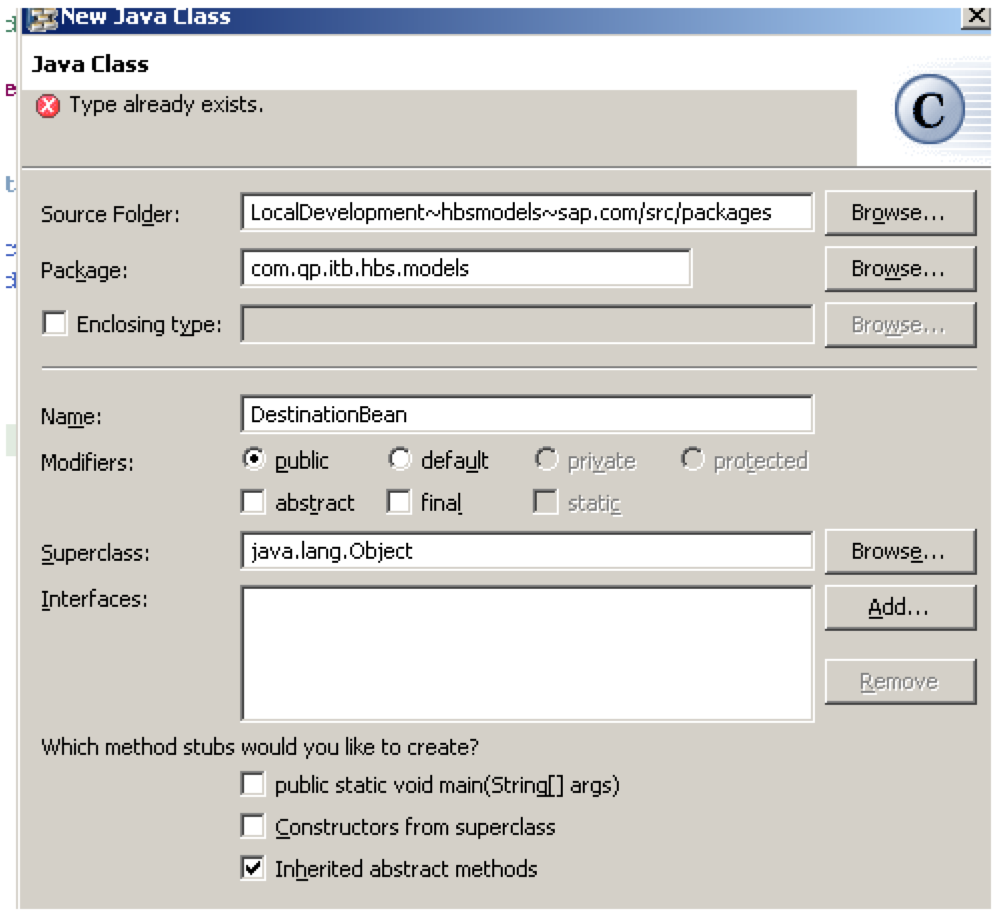
Click on the Next Button.


New Java Project
Java Settings
Define the Java build settings.
Source
Projects
Libraries
Order and Export
Source folders on build path:
LocalDevelopment~hbsmodels~sap.com (new)
Add Folder...
Edit...
Remove
Allow output folders for source folders.
Default output folder:
LocalDevelopment~hbsmodels~sap.com
Browse...
< Back
Next >
Finish
Cancel

Click on Finish button.

Switch to the Package explorer view. Create a new java class.




New Java Class

Java Class

Type already exists.

Source Folder: LocalDevelopment~hbsmodels~sap.com/src/packages

Package: com.qp.itb.hbs.models

[ ] Enclosing type:

Name: DestinationBean

Modifiers: [✓ public] default private protected

[ ] abstract [ ] final [ ] static

Superclass: java.lang.Object

Interfaces:

Which method stubs would you like to create?

[ ] public static void main(String[] args)

[ ] Constructors from superclass

[✓] Inherited abstract methods

Implementing the Command Bean

The JavaBean class DestinationBean, which serves as a Command Bean (or Model), should be able to store all the data that is needed to display about the destination (including possible error messages). DestinationBean will be initialized from within the Web Dynpro and passes the destination code and destination name data to the session bean (HelicopterBookingBean) of the EJB Project.

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

Add the following attribute to the java class.

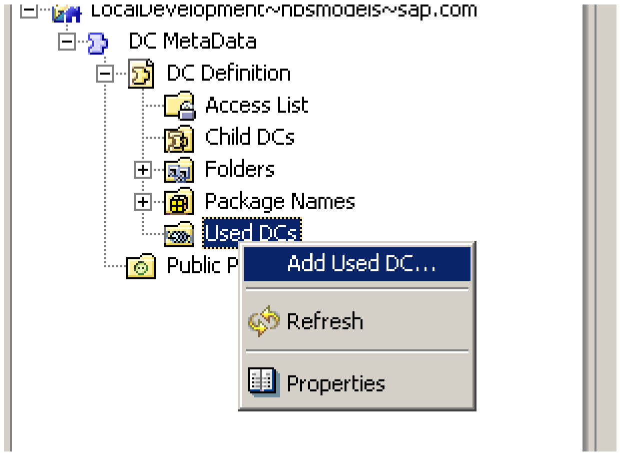
```
public class DestinationBean {
    
    private String destinationCode;
    private String destinationName;
    private String message;

}
```

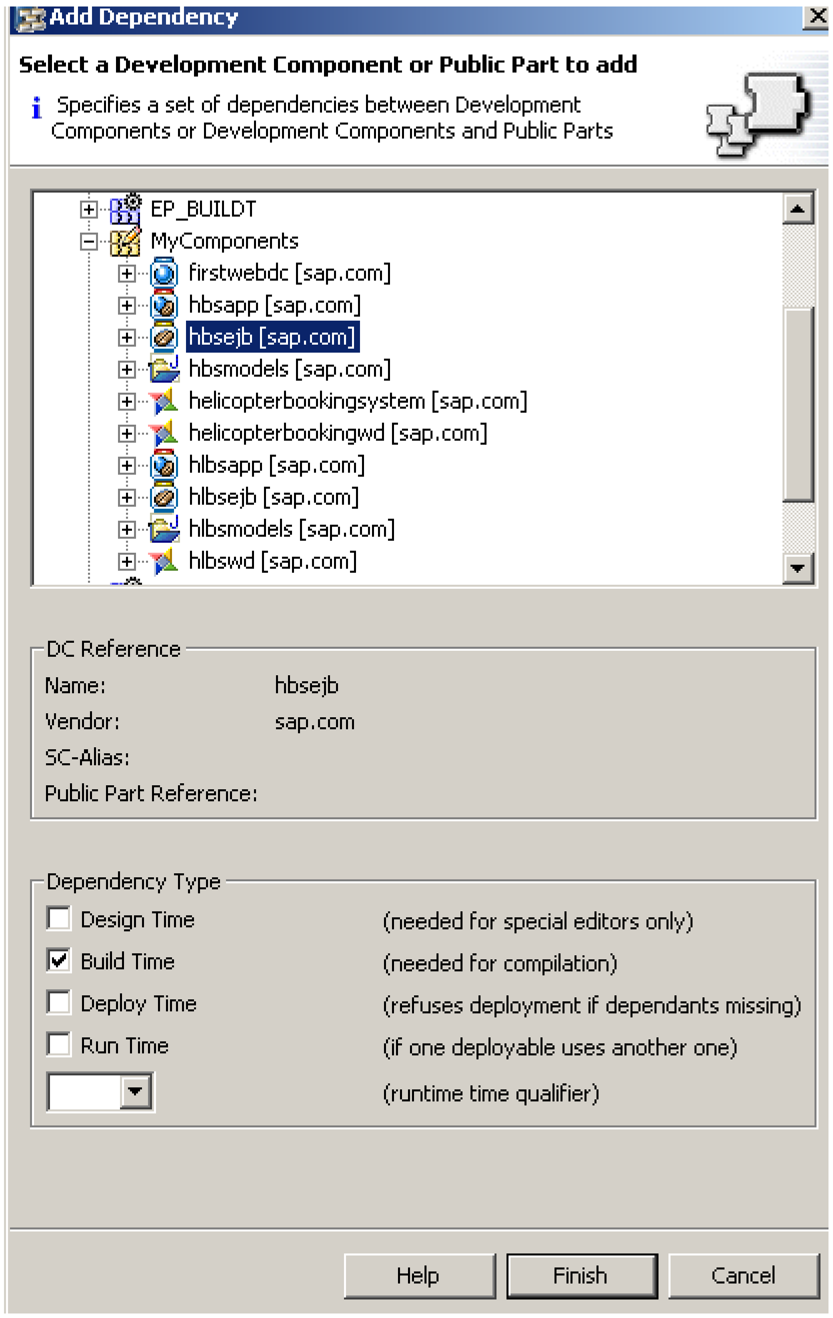
Since we want to call the session bean from within the command bean, we now need to declare variables for the local and the local home interface (HelicopterBookingLocal and HelicopterBookingLocalHome) of the session bean.

Insert the following lines of code:

```
public class DestinationBean {

    private String destinationCode;
    private String destinationName;
    private String message;

    HelicopterBookingLocal local = null;
    HelicopterBookingLocalHome home = null;


}
```

Now you noticed, there are compilation errors in the code. You need to add the EJB Project in the used DC in this project.




Add Dependency
Select a Development Component or Public Part to add
Specifies a set of dependencies between Development Components or Development Components and Public Parts
EP_BUILDT
MyComponents
firstwebdc [sap.com]
hbsapp [sap.com]
hbsejb [sap.com]
hbsmodels [sap.com]
helicopterbookingsystem [sap.com]
helicopterbookingwd [sap.com]
hlbsapp [sap.com]
hlbsejb [sap.com]
hlbsmodels [sap.com]
hlbswd [sap.com]
Help
Finish
Cancel

DC Reference

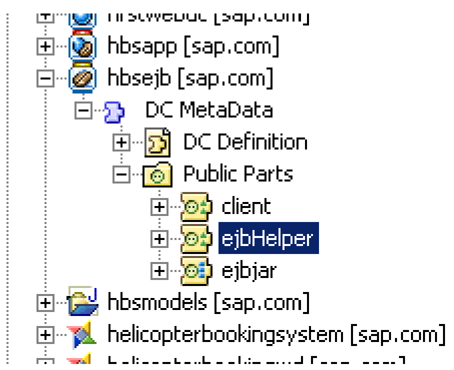
Name:	hbsejb
Vendor:	sap.com
SC-Alias:	
Public Part Reference:	

Dependency Type

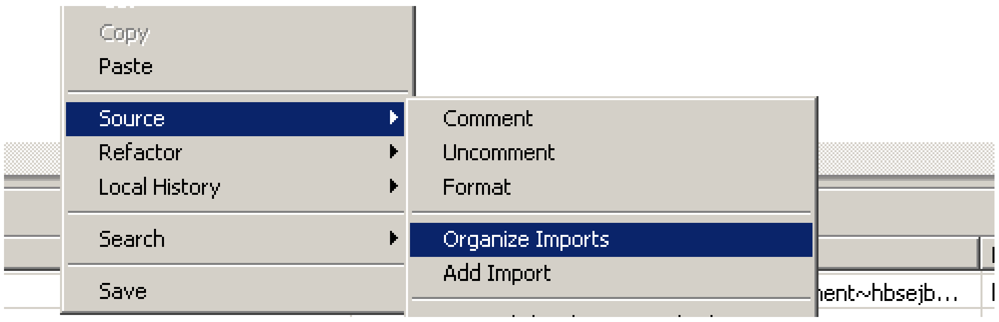
☐ Design Time	(needed for special editors only)
☑ Build Time	(needed for compilation)
☐ Deploy Time	(refuses deployment if dependants missing)
☐ Run Time	(if one deployable uses another one)
	(runtime time qualifier)

Click on Finish button.

Add Used DC – ejbHelper

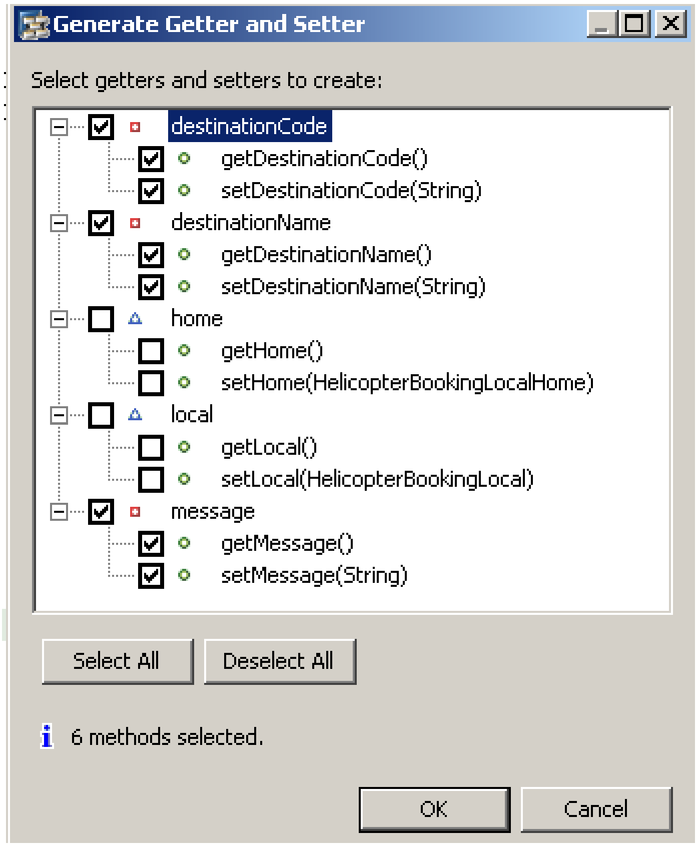


Now right click on the code and say organize imports.



Now the errors are gone.

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 destinationCode, destinationName and message.



Generate Getter and Setter

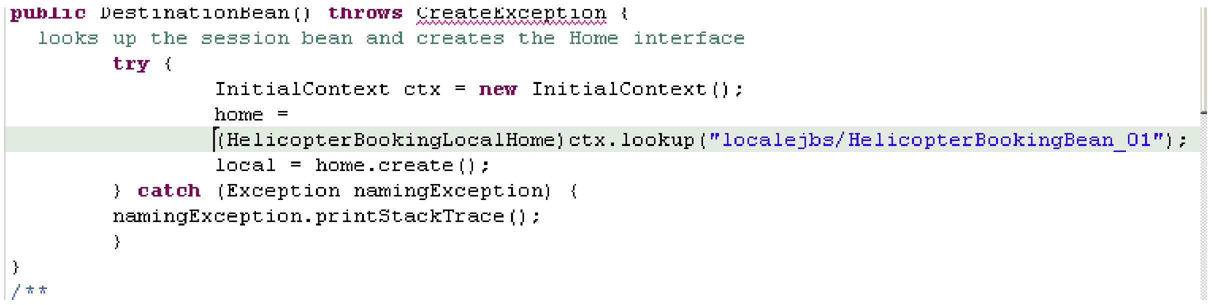
Select getters and setters to create:

- [✓] destinationCode
  - [✓] getDestinationCode()
  - [✓] setDestinationCode(String)
- [✓] destinationName
  - [✓] getDestinationName()
  - [✓] setDestinationName(String)
- [ ] home
  - [ ] getHome()
  - [ ] setHome(HelicopterBookingLocalHome)
- [ ] local
  - [ ] getLocal()
  - [ ] setLocal(HelicopterBookingLocal)
- [✓] message
  - [✓] getMessage()
  - [✓] setMessage(String)

6 methods selected.

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.

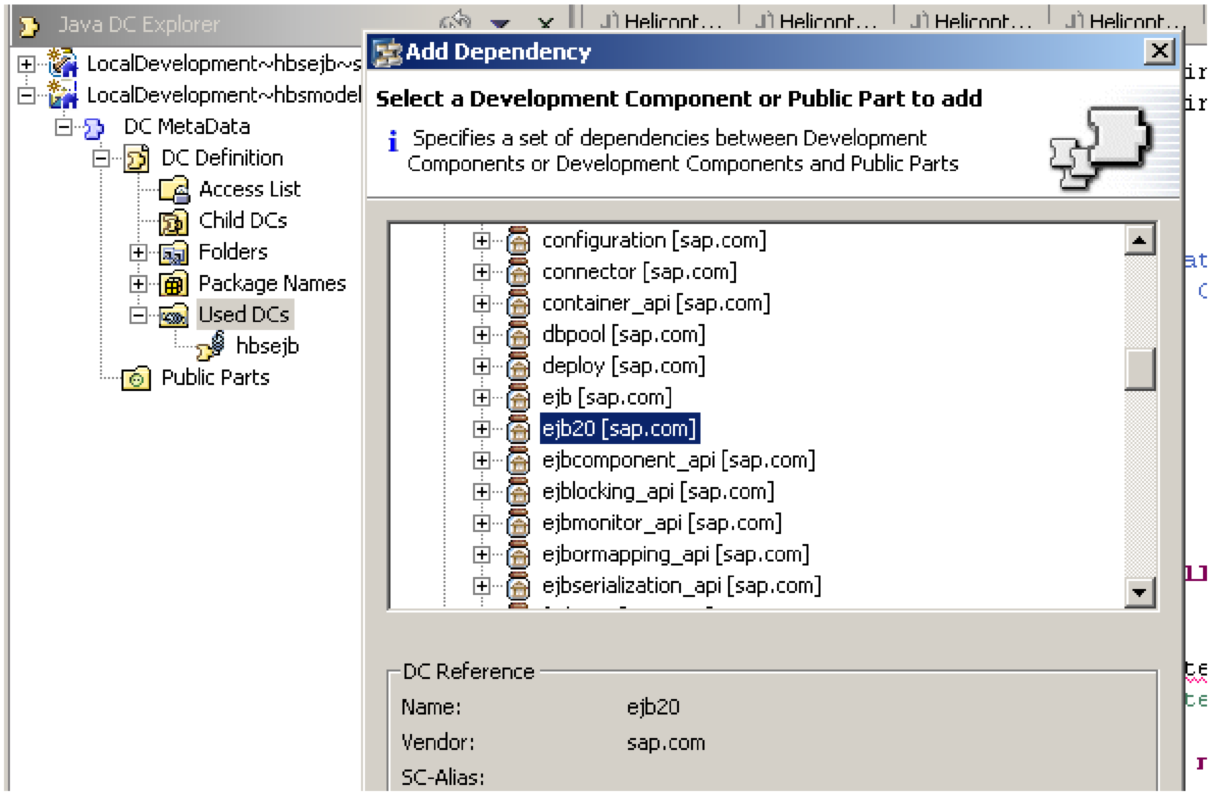
Add the following lines of code:



```
public DestinationBean() throws CreateException {
  looks up the session bean and creates the Home interface
        try {
                InitialContext ctx = new InitialContext();
                home =
                (HelicopterBookingLocalHome)ctx.lookup("localejbs/HelicopterBookingBean_01");
                local = home.create();
        } catch (Exception namingException) {
        namingException.printStackTrace();
        }
}
/**
```

Do not worry if errors occur.

Add ejb20 to the used DC.



Java DC Explorer
- LocalDevelopment~hbsejb~s
- LocalDevelopment~hbsmodel
  - DC MetaData
    - DC Definition
      - Access List
      - Child DCs
      - Folders
      - Package Names
      - Used DCs
        - hbsejb
    - Public Parts

Add Dependency

Select a Development Component or Public Part to add

Specifies a set of dependencies between Development Components or Development Components and Public Parts

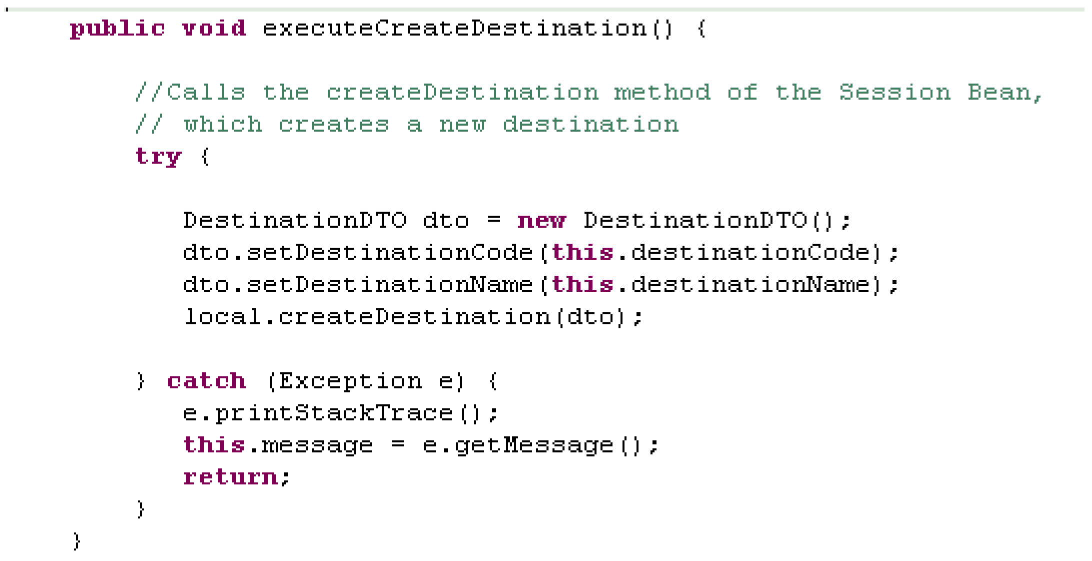
- configuration [sap.com]
- connector [sap.com]
- container_api [sap.com]
- dbpool [sap.com]
- deploy [sap.com]
- ejb [sap.com]
- ejb20 [sap.com]
- ejbcomponent_api [sap.com]
- ejblocking_api [sap.com]
- ejbmonitor_api [sap.com]
- ejbormapping_api [sap.com]
- ejbserialization_api [sap.com]

DC Reference

Name:	ejb20
Vendor:	sap.com
SC-Alias:	

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

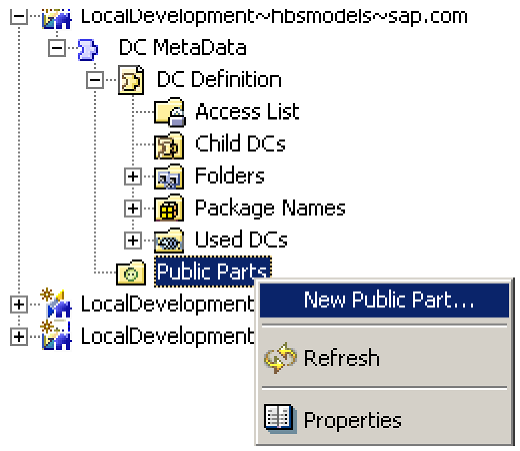
The next step is to create a method called executeCreateDestination, which calls the business method (createDestination()) of the session bean. Insert the following lines of code:

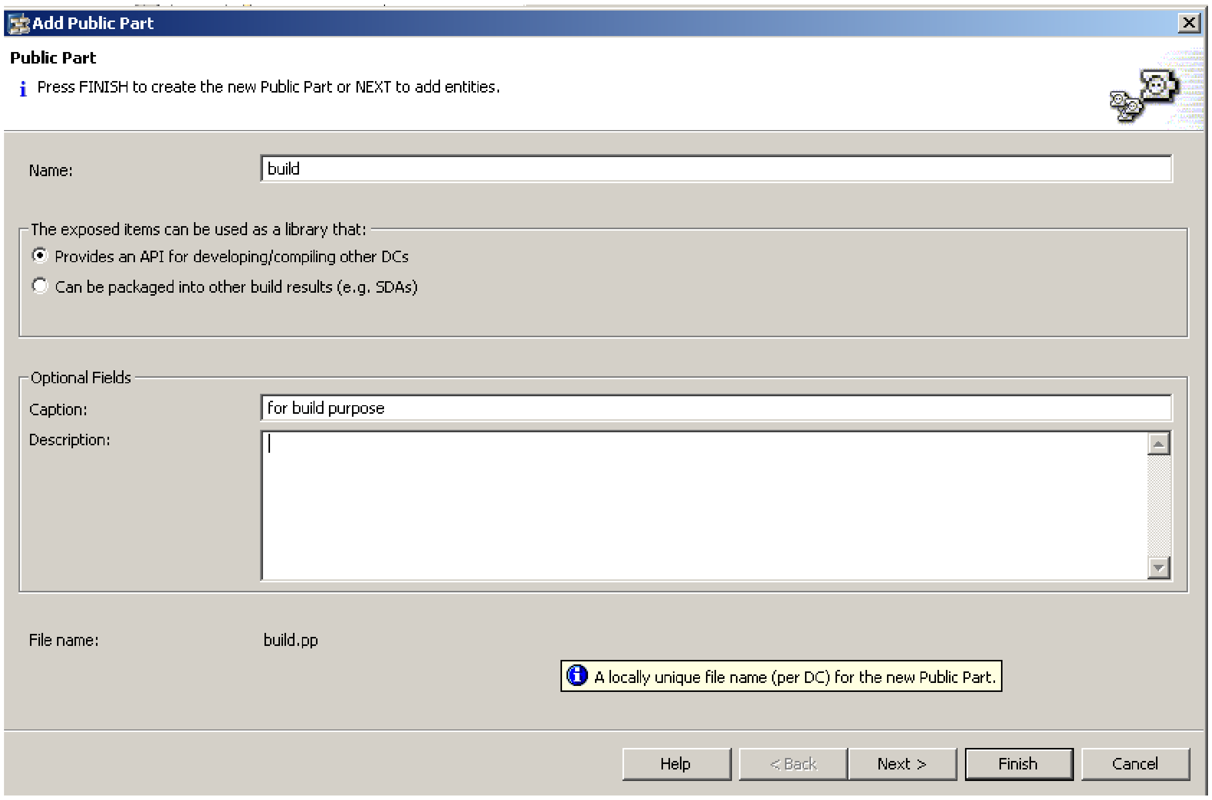


```
    public void executeCreateDestination() {

        //Calls the createDestination method of the Session Bean,
        // which creates a new destination
        try {

           DestinationDTO dto = new DestinationDTO();
           dto.setDestinationCode(this.destinationCode);
           dto.setDestinationName(this.destinationName);
           local.createDestination(dto);

        } catch (Exception e) {
           e.printStackTrace();
           this.message = e.getMessage();
           return;
        }
    }
```

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

Expose two public parts: assembly with all classes and build with DTOs and bean classes





Public Part

Press FINISH to create the new Public Part or NEXT to add entities.

Name:	build

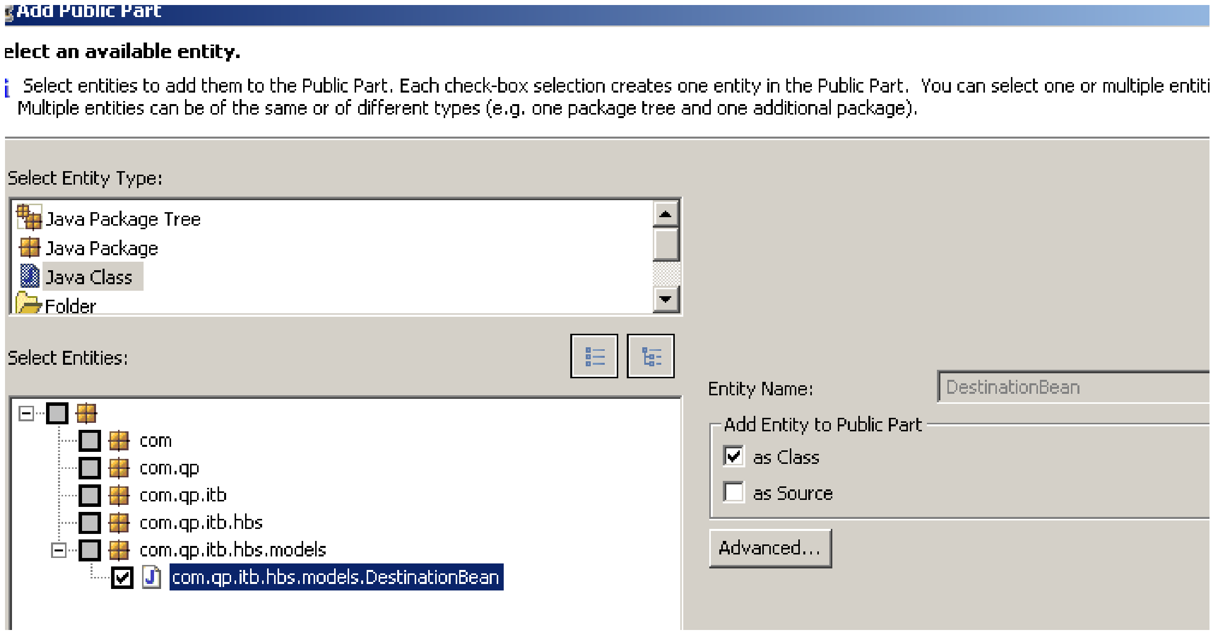
The exposed items can be used as a library that:

- [✓ Provides an API for developing/compiling other DCs]
- Can be packaged into other build results (e.g. SDAs)

Optional Fields

Caption:	for build purpose
Description:	
File name:	build.pp

A locally unique file name (per DC) for the new Public Part.



Add Public Part

elect an available entity.

Select entities to add them to the Public Part. Each check-box selection creates one entity in the Public Part. You can select one or multiple entiti
Multiple entities can be of the same or of different types (e.g. one package tree and one additional package).

Select Entity Type:

- Java Package Tree
- Java Package
- Java Class
- Folder

Select Entities:

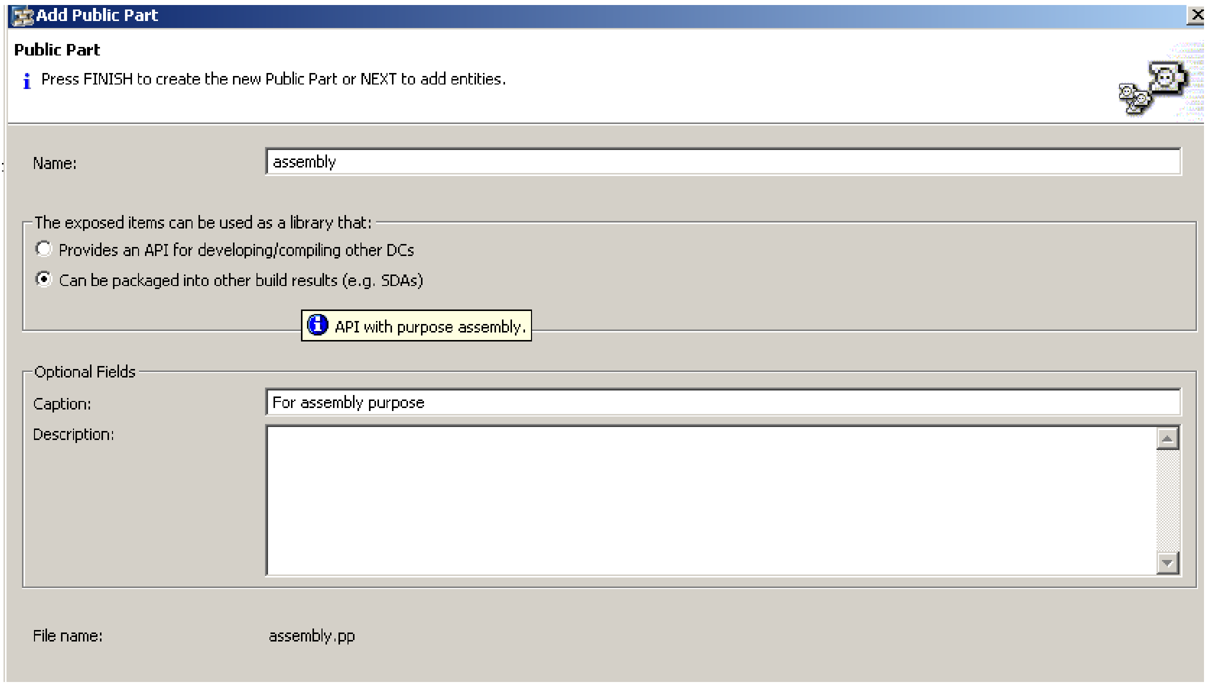
- [ ] com
- [ ] com.qp
- [ ] com.qp.itb
- [ ] com.qp.itb.hbs
- [ ] com.qp.itb.hbs.models
  - [x] com.qp.itb.hbs.models.DestinationBean

Entity Name: DestinationBean

Add Entity to Public Part
- [x] as Class
- [ ] as Source

Click on Finish button.

Then create a new public part called **assembly**.



Add Public Part

Public Part

Press FINISH to create the new Public Part or NEXT to add entities.

Name: assembly

The exposed items can be used as a library that:
- ( ) Provides an API for developing/compiling other DCs
- (•) Can be packaged into other build results (e.g. SDAs)

API with purpose assembly.

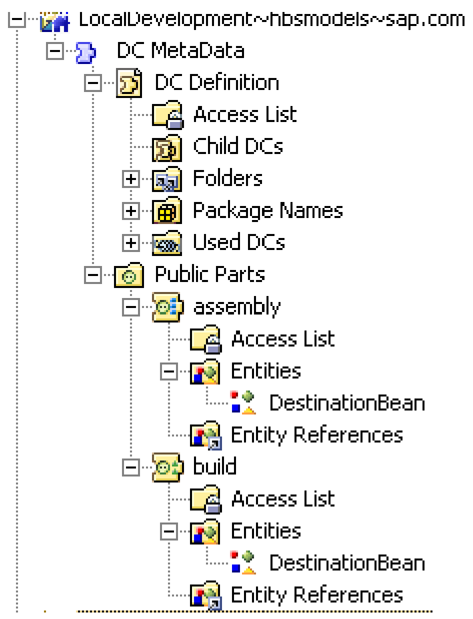
Optional Fields

Caption: For assembly purpose

Description:

File name: assembly.pp

Now your project structure will be looking as below:



Now let's move to the main part which is importing this model in an existing Web Dynpro application.

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 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).

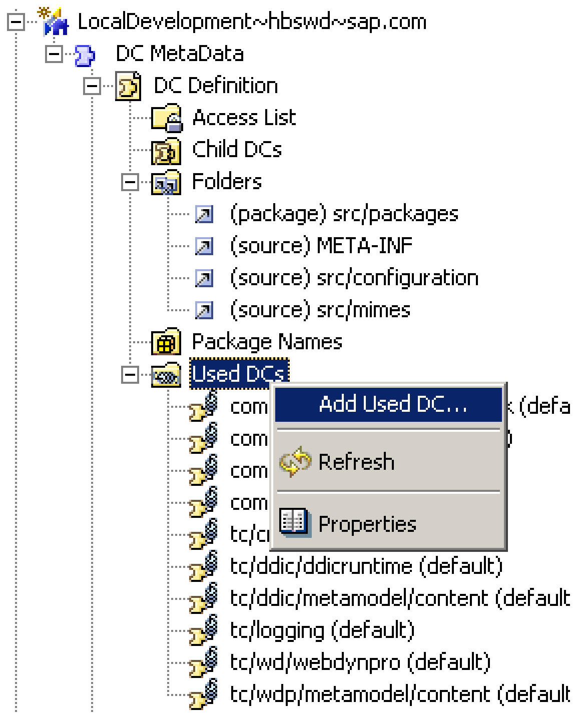
Importing the Java Bean Model

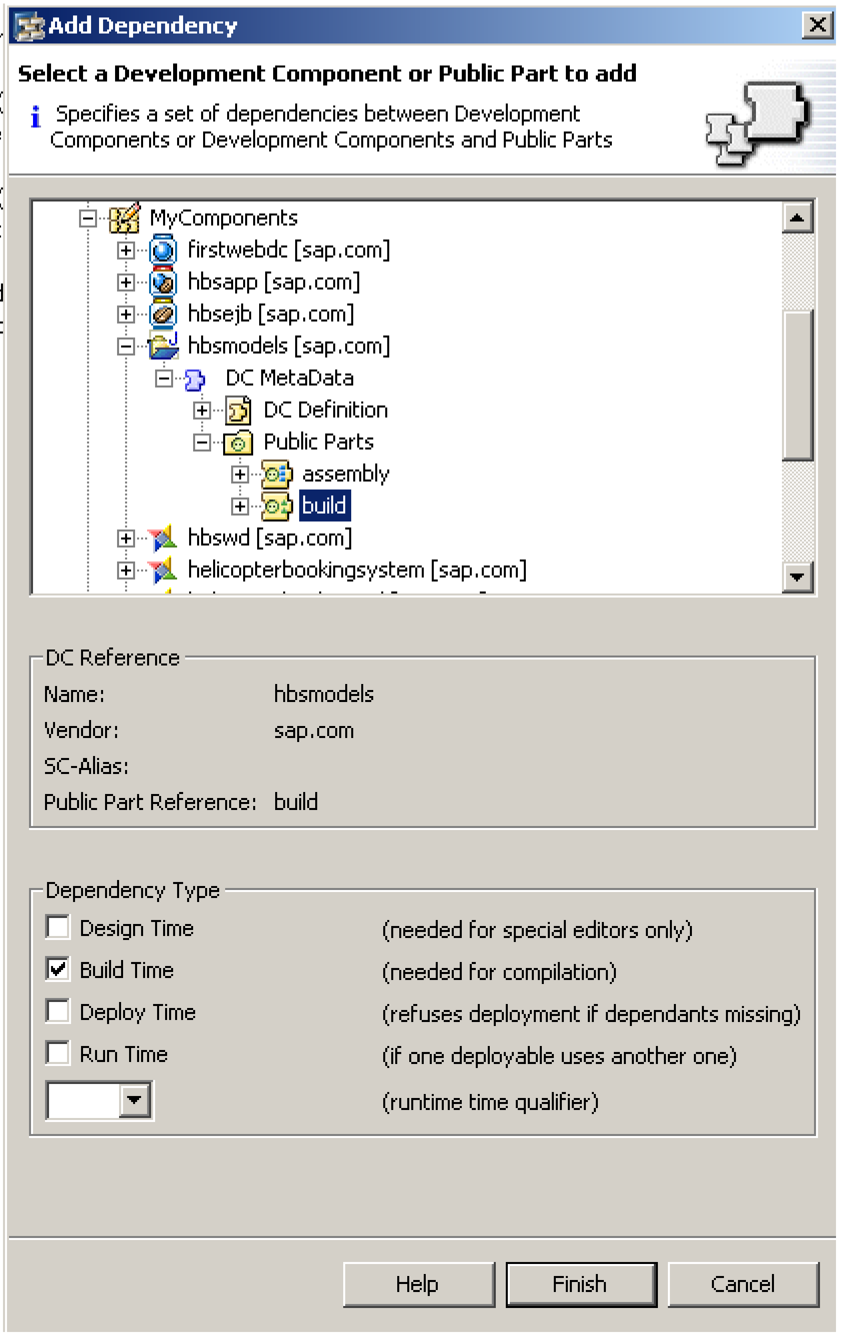
A model enables the user to access business data that is outside the Web Dynpro application from within the application. In this tutorial, the business data that belongs to the sample application is made accessible through a JavaBean import.

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

Add Used DC





Select a Development Component or Public Part to add

Specifies a set of dependencies between Development Components or Development Components and Public Parts

```
MyComponents
  firstwebdc [sap.com]
  hbsapp [sap.com]
  hbsejb [sap.com]
  hbsmodels [sap.com]
    DC MetaData
      DC Definition
      Public Parts
        assembly
        build
  hbswd [sap.com]
  helicopterbookingsystem [sap.com]
```

DC Reference

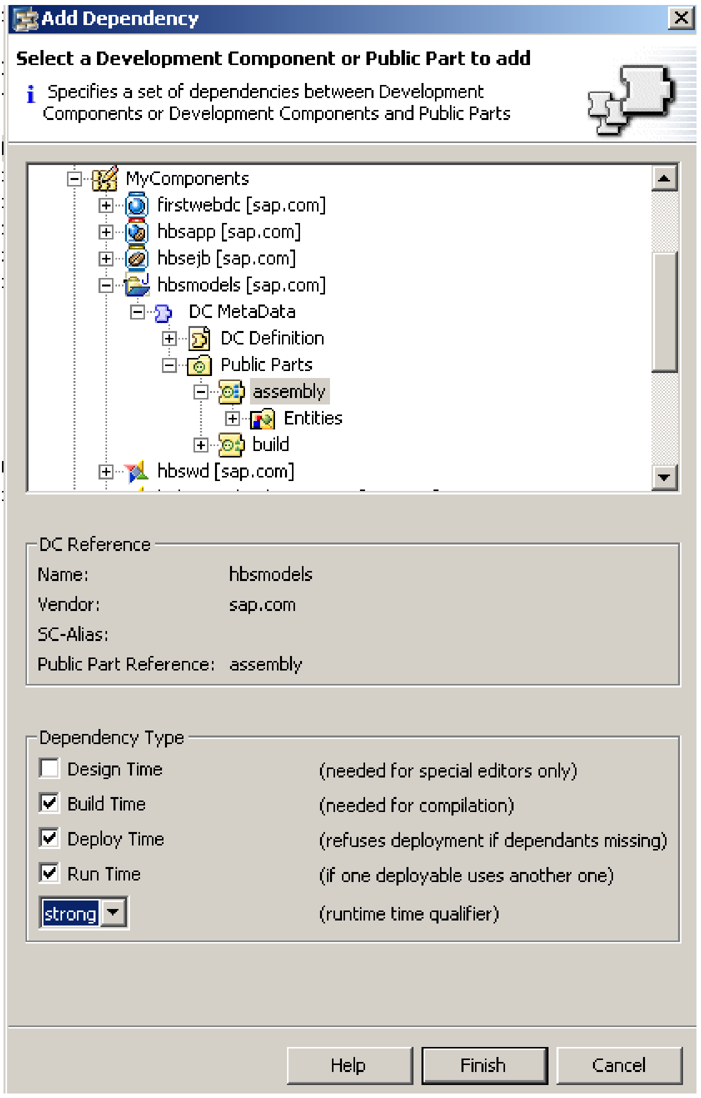
Name:	hbsmodels
Vendor:	sap.com
SC-Alias:	
Public Part Reference:	build

Dependency Type

☐ Design Time	(needed for special editors only)
☑ Build Time	(needed for compilation)
☐ Deploy Time	(refuses deployment if dependants missing)
☐ Run Time	(if one deployable uses another one)
	(runtime time qualifier)

Click on Finish button.

Select Public Part ‘assembly’ to add



Select a Development Component or Public Part to add

Specifies a set of dependencies between Development Components or Development Components and Public Parts

- MyComponents
  - firstwebdc [sap.com]
  - hbsapp [sap.com]
  - hbsejb [sap.com]
  - hbsmodels [sap.com]
    - DC MetaData
      - DC Definition
      - Public Parts
        - assembly
          - Entities
        - build
  - hbswd [sap.com]

DC Reference

Name:	hbsmodels
Vendor:	sap.com
SC-Alias:	
Public Part Reference:	assembly

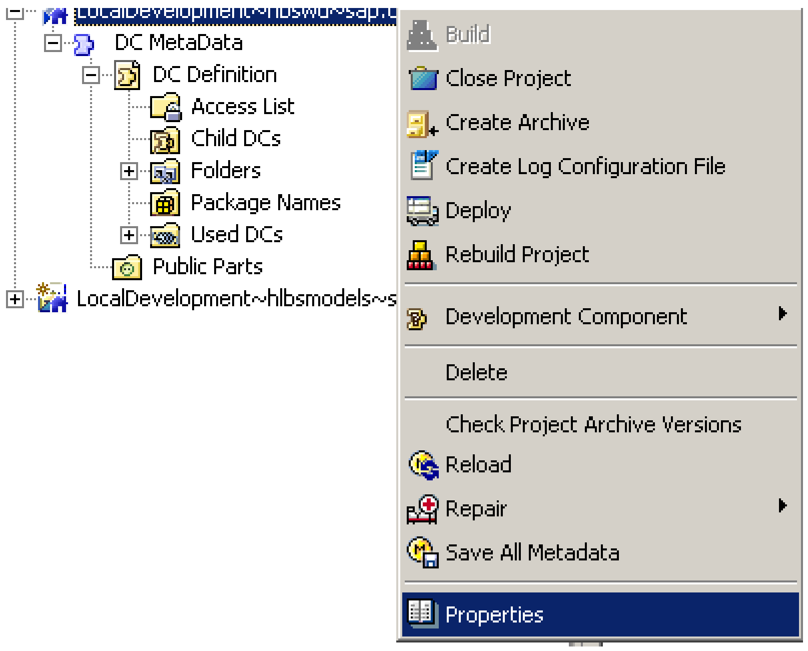
Dependency Type

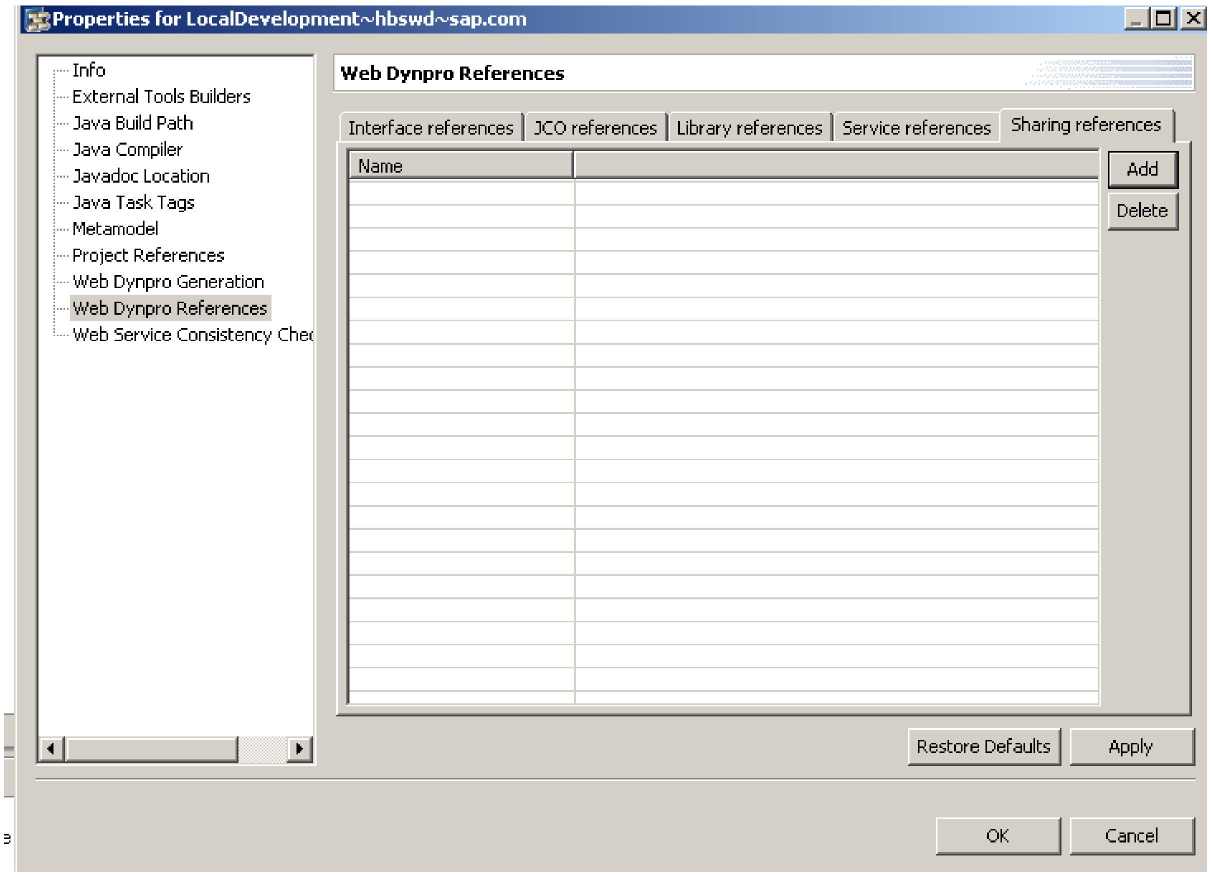
☐ Design Time	(needed for special editors only)
☑ Build Time	(needed for compilation)
☑ Deploy Time	(refuses deployment if dependants missing)
☑ Run Time	(if one deployable uses another one)
strong	(runtime time qualifier)

Click on Finish button.

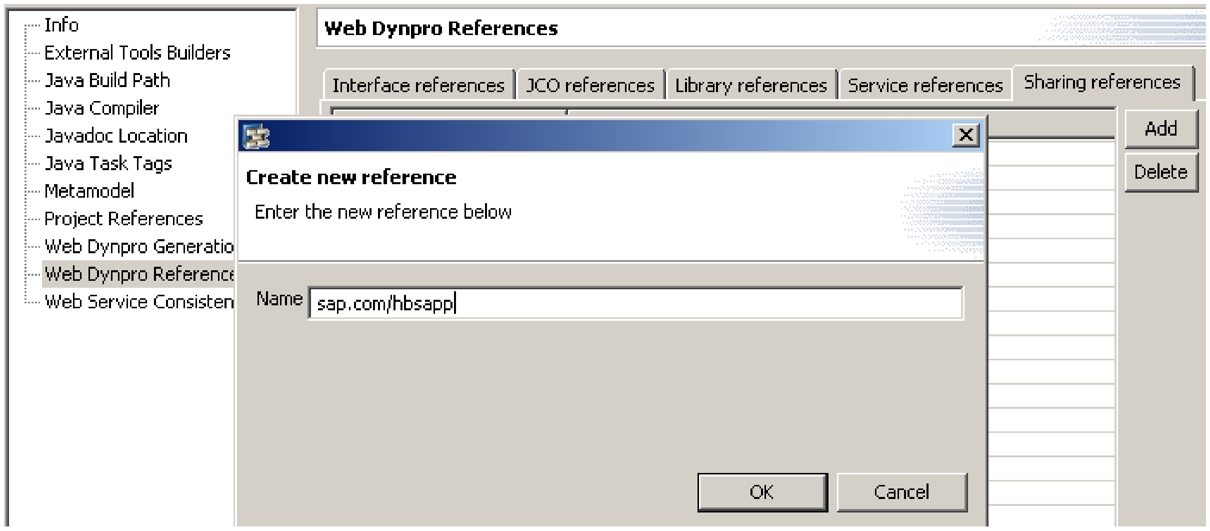
Add Web Dynpro Sharing References to EJB

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

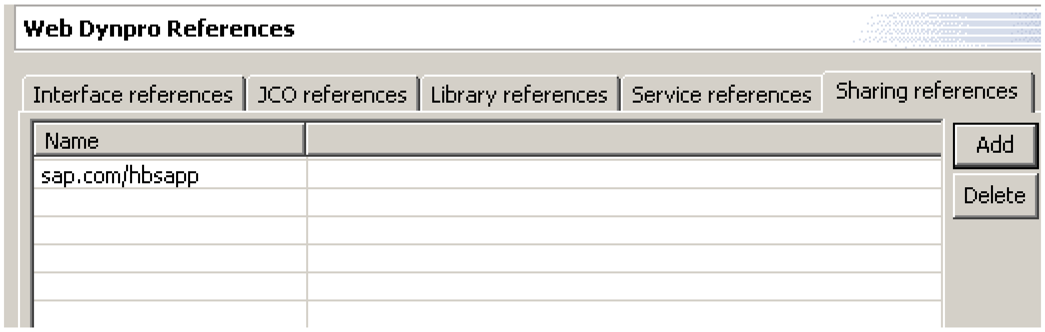




Choose the Add button and enter the fully qualified name (vendor name/Ear name) for the Ear required



Name: sap.com/hbsapp

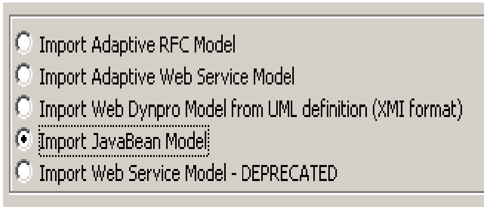


Name	
sap.com/hbsapp	

Now you have the sharing reference to enterprise application.

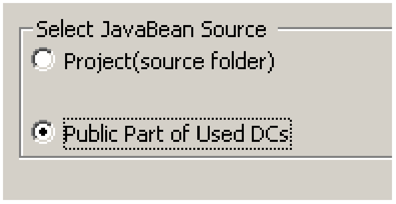
Importing the Java Bean Model Implementation

1. In the project structure, expand the node Web Dynpro→ Models. From the context menu, choose Models to start the appropriate wizard.
2. Choose the Import JavaBean Model option, followed by Next. Enter the name DestinationModel as the model name and com.qp.itb.hbs.comp.dest.models as the package name.



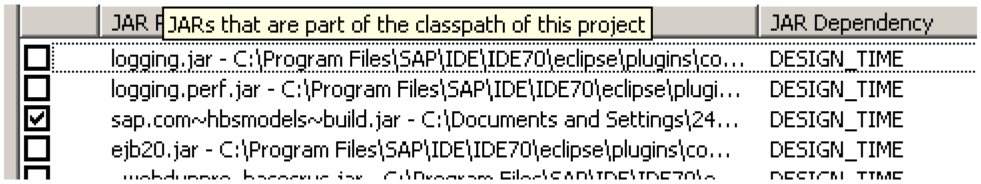
Import Adaptive RFC Model
Import Adaptive Web Service Model
Import Web Dynpro Model from UML definition (XMI format)
[✓ Import JavaBean Model]
Import Web Service Model - DEPRECATED

3. Select Public Part of Used DCs and click Next.



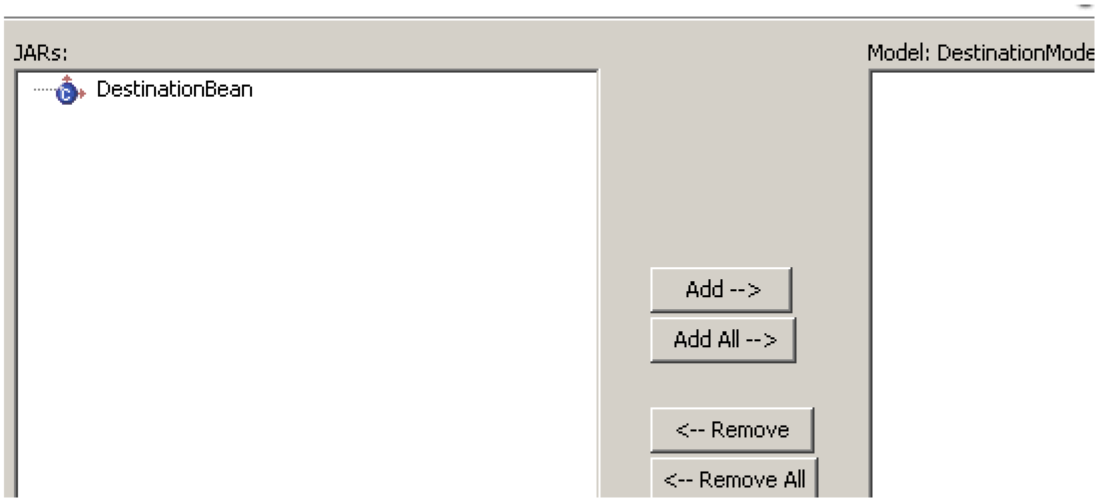
Select JavaBean Source
Project(source folder)
[✓ Public Part of Used DCs]

4. In the next screen, select Avaialble public part DCs

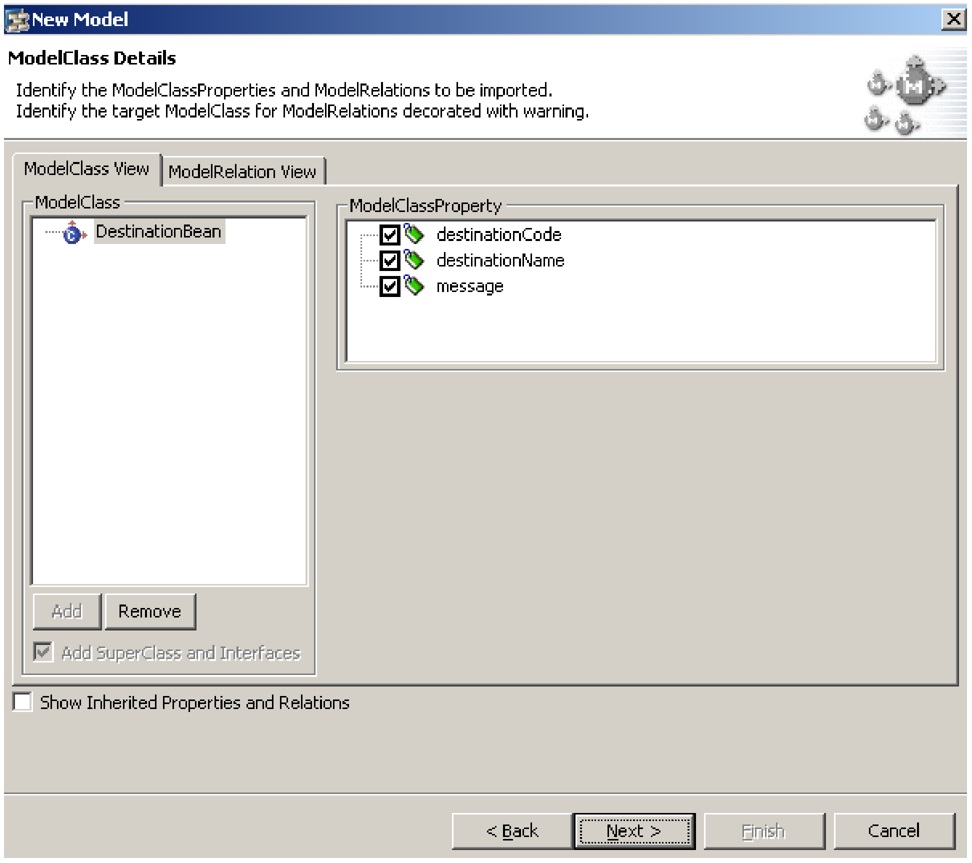


	JAR F	JAR Dependency
☐	logging.jar - C:\Program Files\SAP\IDE\IDE70\eclipse\plugins\co...	DESIGN_TIME
☐	logging.perf.jar - C:\Program Files\SAP\IDE\IDE70\eclipse\plugi...	DESIGN_TIME
☑	sap.com~hbsmodels~build.jar - C:\Documents and Settings\24...	DESIGN_TIME
☐	ejb20.jar - C:\Program Files\SAP\IDE\IDE70\eclipse\plugins\co...	DESIGN_TIME

5. In the next screen, select DestinationBean and chooseAdd →



6. In the Next Screen, Select the default.


New Model

ModelClass Details

Identify the ModelClassProperties and ModelRelations to be imported.
Identify the target ModelClass for ModelRelations decorated with warning.

- [✓] destinationCode
- [✓] destinationName
- [✓] message

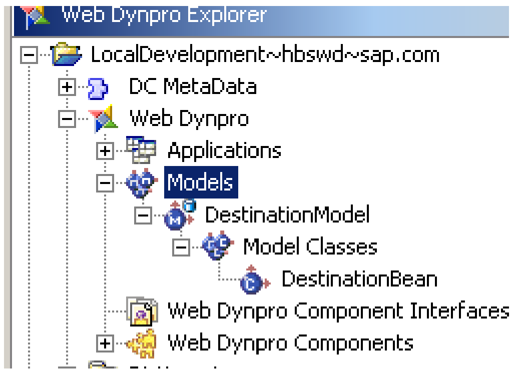
[✓] Add SuperClass and Interfaces

[ ] Show Inherited Properties and Relations

Click on Next and in the next screen, click on Finish button.

Result

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



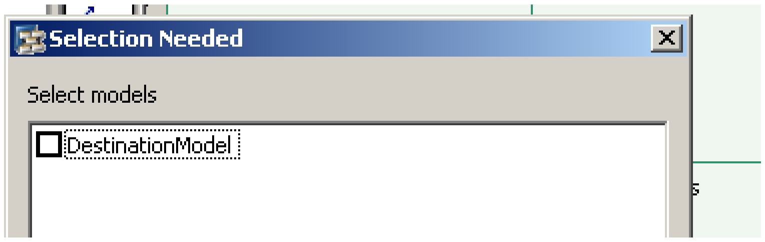
Creating the Context

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 destination details. 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

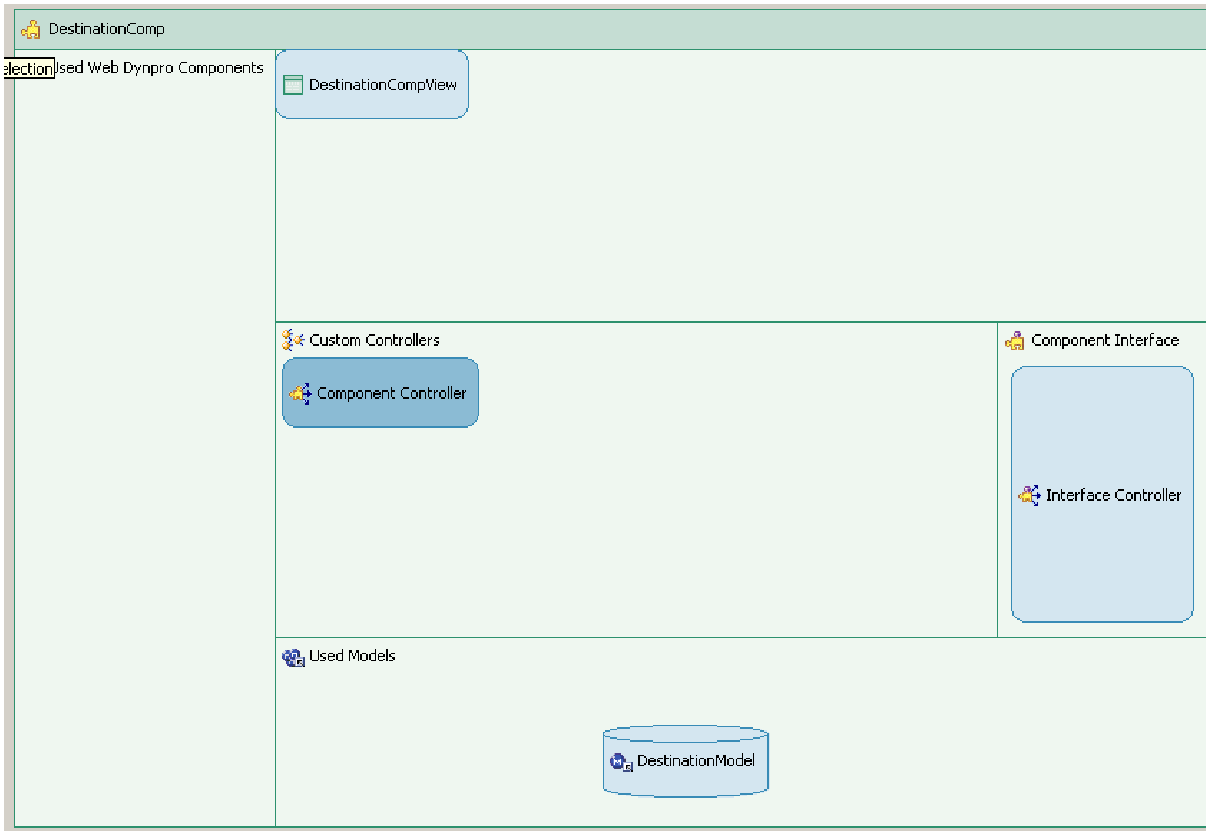
Procedure

Adding a Model to the Web Dynpro Component

- Open the Data Modeler through the context menu entry Open Data Modeler Open Data Modeler of the node DestinationComp. 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.



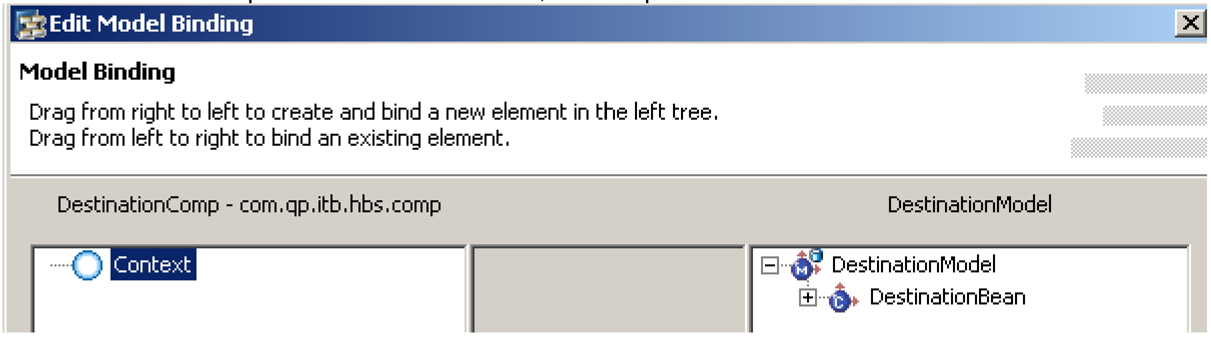
- Select the DestinationModel checkbox and confirm by pressing OK. The structure of the Web Dynpro component DestinationComp will look like this in the Data Modeler:



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.

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

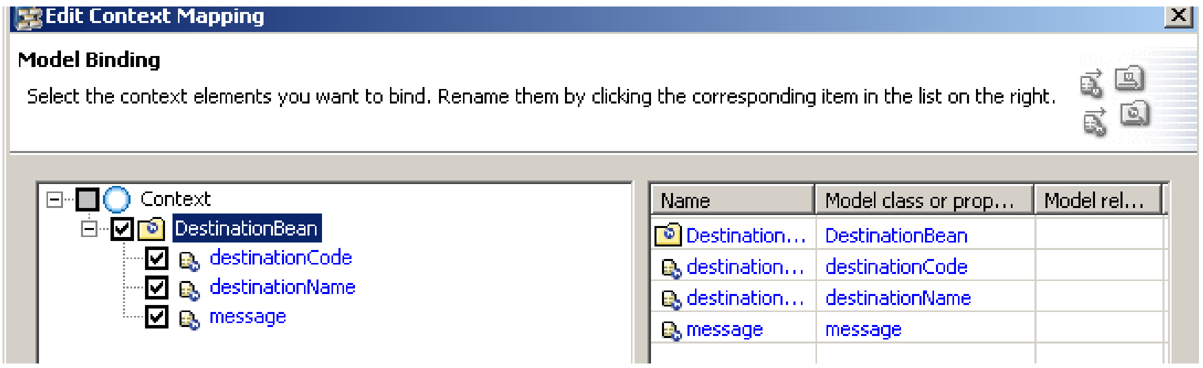


Edit Model Binding

Model Binding

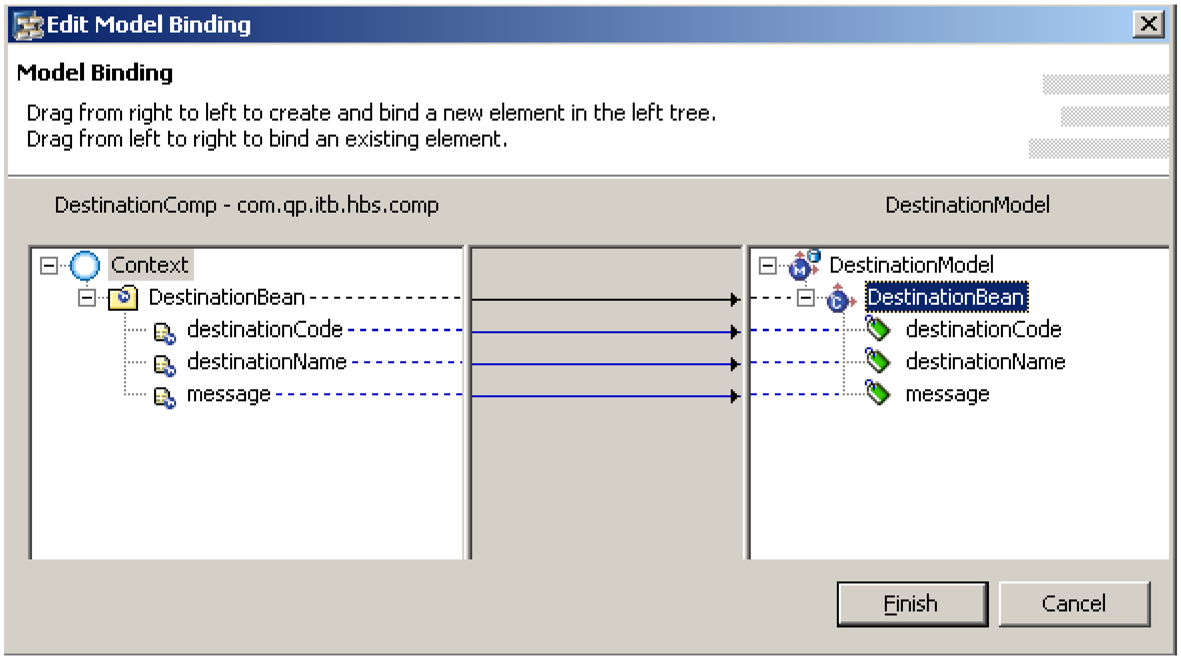
Drag from right to left to create and bind a new element in the left tree.
Drag from left to right to bind an existing element.

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


Edit Context Mapping
Model Binding
Select the context elements you want to bind. Rename them by clicking the corresponding item in the list on the right.
Context
DestinationBean
destinationCode
destinationName
message

Name	Model class or prop...	Model rel...
Destination...	DestinationBean	
destination...	destinationCode	
destination...	destinationName	
message	message	

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.



- 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 DestinationComp, 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.

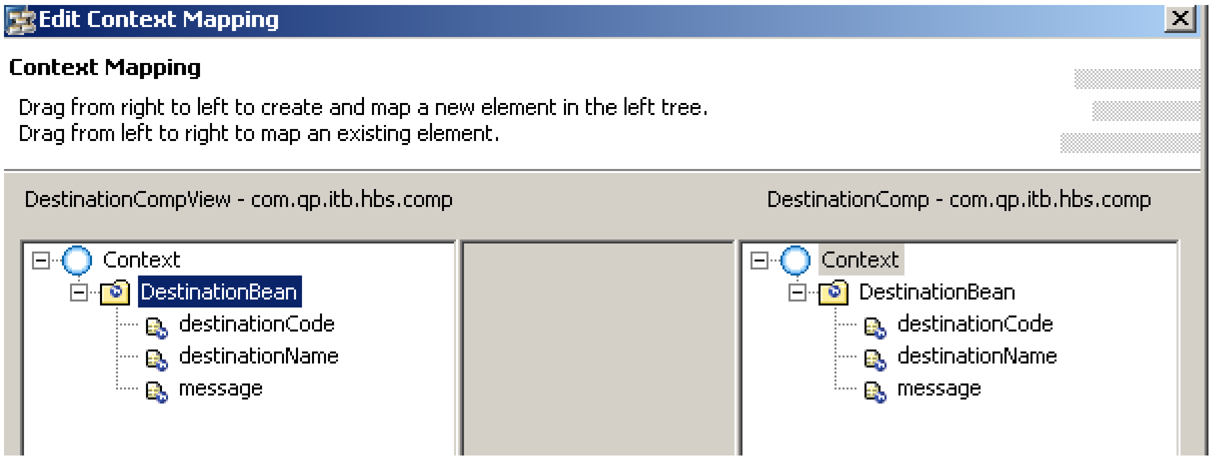
It would be technically possible for a view context to access the model, but this would not be good design style. To use the MVC paradigms in a consistent manner, you require the component controller as a binding link between the view and the model. In this step of the tutorial, you will map the context 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 DestinationComp.

Procedure

- In the toolbar, choose Create a data link. Draw a line, beginning with the DestinationCompView view and ending at the Component Controller. The Model Binding Wizard starts automatically.
- Using Drag & Drop, drag the node of the model class DestinationBean 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:



Edit Context Mapping

Context Mapping

Drag from right to left to create and map a new element in the left tree.
Drag from left to right to map an existing element.

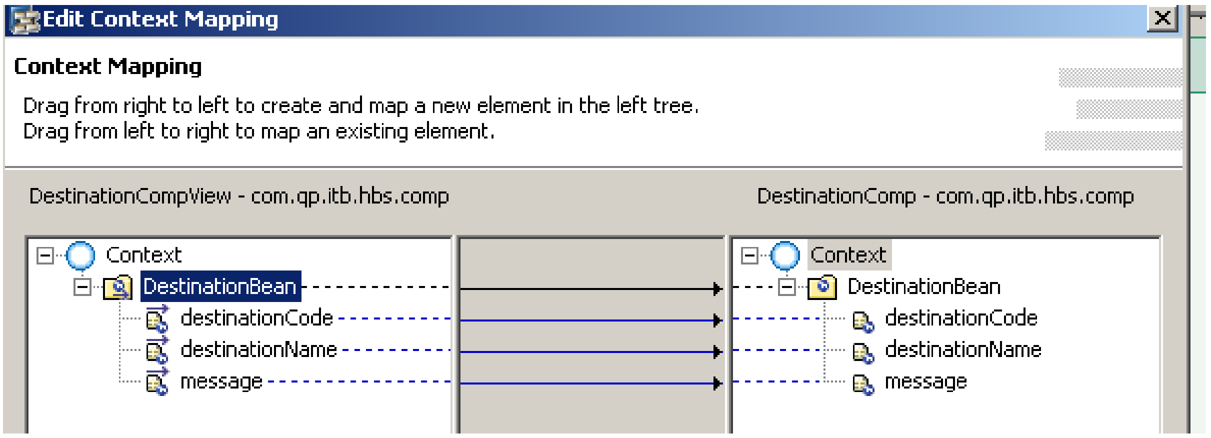
DestinationCompView - com.qp.itb.hbs.comp

DestinationComp - com.qp.itb.hbs.comp

```
Context
  DestinationBean
    destinationCode
    destinationName
    message
```

```
Context
  DestinationBean
    destinationCode
    destinationName
    message
```

-



Edit Context Mapping

Context Mapping

Drag from right to left to create and map a new element in the left tree.
Drag from left to right to map an existing element.

DestinationCompView - com.qp.itb.hbs.comp

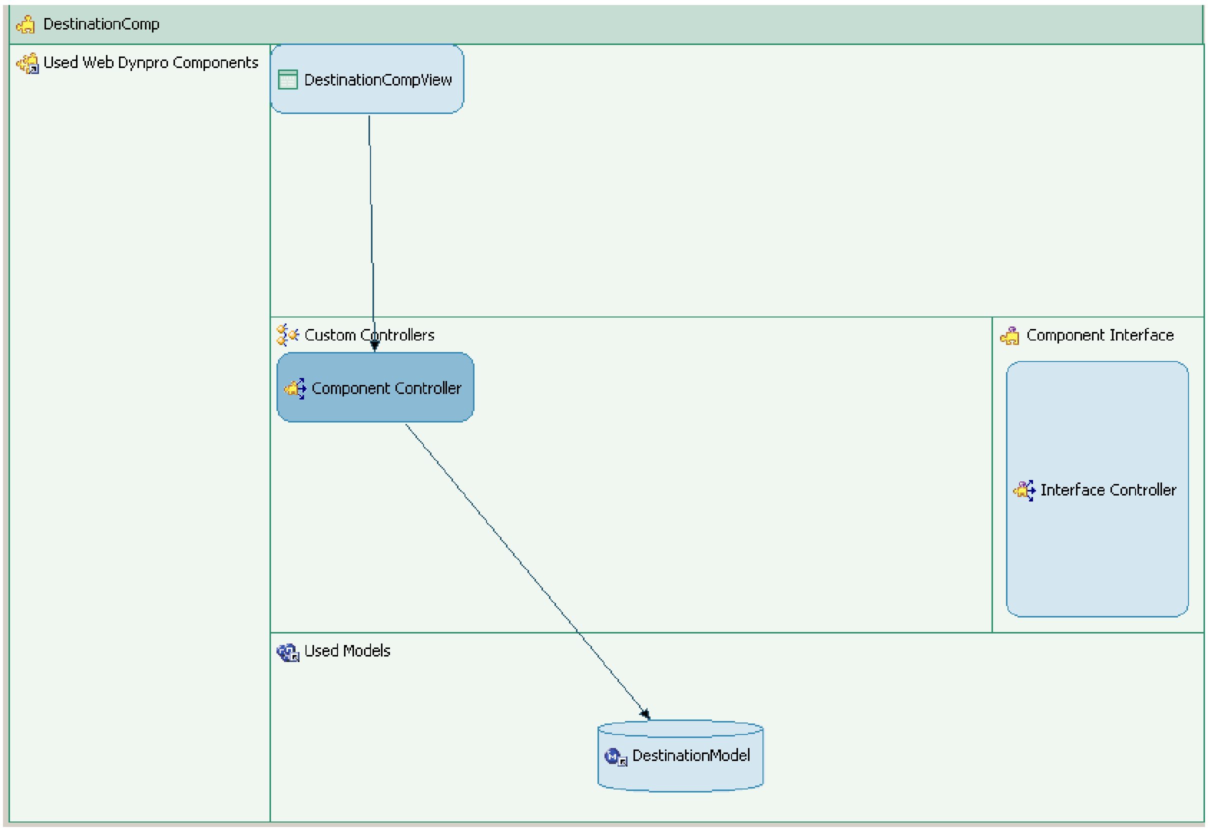
DestinationComp - com.qp.itb.hbs.comp

```
Context                                     Context
  DestinationBean   ---------->  DestinationBean
    destinationCode ---------->    destinationCode
    destinationName ---------->    destinationName
    message         ---------->    message
```

- 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.



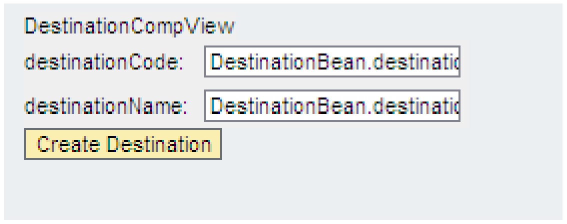
•

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 DestinationCompView, these context variables are added in the form using the Apply Template Therefore, you do not need to worry about the layout of this 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.
Set up the data bindings between the UI element properties and the respective context model attributes.


DestinationCompView

destinationCode:	DestinationBean.destinatic
destinationName:	DestinationBean.destinatic

Implementing 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

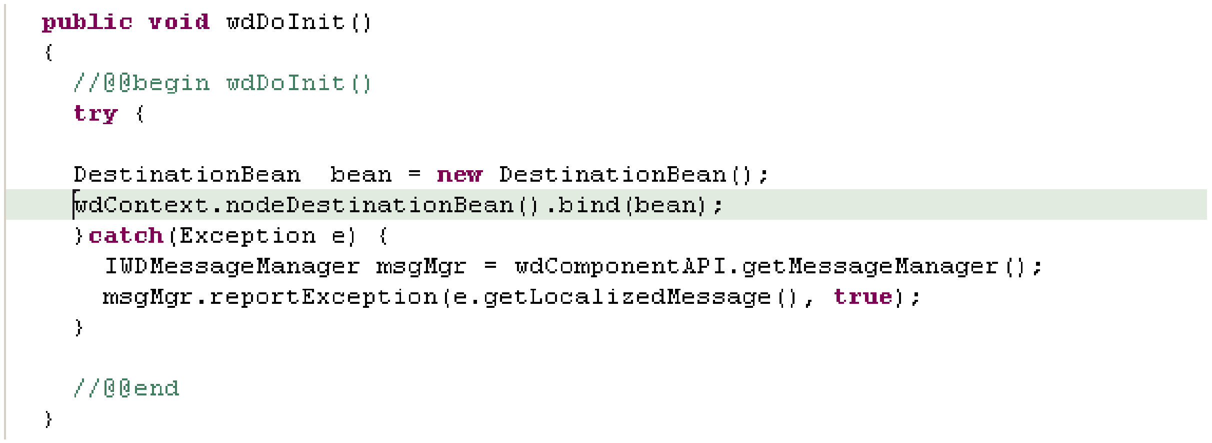
Implementing the Component Controller

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 DestinationBean, which was declared at design time.

- Choose the Implementation tab for the component controller DestinationComp.

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

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



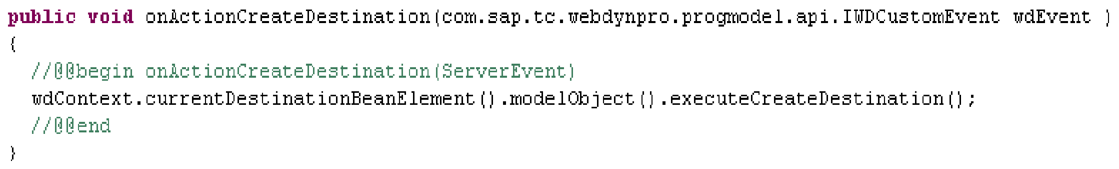
```
public void wdDoInit()
{
  //@@begin wdDoInit()
  try {

  DestinationBean  bean = new DestinationBean();
  wdContext.nodeDestinationBean().bind(bean);
  }catch(Exception e) {
    IWDMessageManager msgMgr = wdComponentAPI.getMessageManager();
    msgMgr.reportException(e.getLocalizedMessage(), true);
  }

  //@@end
}
```

- 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

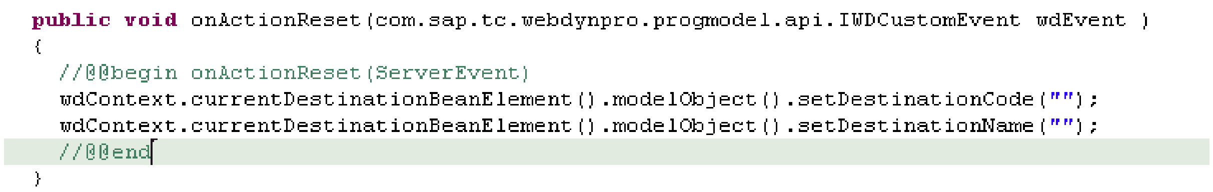
The actual command bean is now called through the executeCreateDestination() method call of the model object currently stored in the context model node. This already contains the destination code and destination name data entered by the user – through data binding and context mapping.

- ...
- Choose the Implementation tab for the view controller DestinationCompView.
- In the onActionCreateDestination () method, add the following source code:



```
public void onActionCreateDestination(com.sap.tc.webdynpro.progmodel.api.IWDCustomEvent wdEvent )
{
  //@@begin onActionCreateDestination(ServerEvent)
  wdContext.currentDestinationBeanElement().modelObject().executeCreateDestination();
  //@@end
}
```

- In the onActionReset() method, add the following source code:



```
public void onActionReset(com.sap.tc.webdynpro.progmodel.api.IWDCustomEvent wdEvent )
{
  //@@begin onActionReset(ServerEvent)
  wdContext.currentDestinationBeanElement().modelObject().setDestinationCode("");
  wdContext.currentDestinationBeanElement().modelObject().setDestinationName("");
  //@@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.

Acronyms Used

MVC – Model View Controller

SQL – Standard Query Language

RDBMS – Relational Database Management System

JDBC – Java Database Connectivity

UI – User Interface

DC – Development Component

API – Application Programmer's Interface

EJB – Enterprise Java Bean

DAO – Data Access Object

DTO – Data Transfer Object

Related Content

a) WebDynpro Tutorials:

   https://www.sdn.sap.com/irj/sdn/nw-wdjava?rid=/webcontent/uuid/00b64d9f-fea2-2910-c988-ee2544047f8c

b) Importing Complex JavaBean model into WebDynpro by creating relationships for the model classes: https://www.sdn.sap.com/irj/scn/weblogs?blog=/pub/wlg/3511

c) Oracle as Back end with Web Dynpro https://forums.sdn.sap.com/thread.jspa?threadID=53343

d) https://forums.sdn.sap.com/thread.jspa?threadID=98009

e) For more information, visit the User Interface Technology homepage.

Copyright