Using EJB Functionality in Web Dynpro Java with the Web Service Model

Applies to:

Web Dynpro for Java applications for SAP NetWeaver CE 7.11

Summary

This tutorial shows you how to store and read data via an abstract database layer managed by the Java Persistence API using the Web Dynpro framework. You will be introduced into developing different components such as Dictionary DC, Enterprise Java Bean 3.0 Module, JPA Entity, Web services and finally Web Dynpro Java as UI layer.

Prerequisites

- Java JDK 1.5.0_13 is installed on your client and is identical to that of the Application Server
- Java Environment Variables are set properly
- Access to an SAP NetWeaver Application Server for Java CE 7.11
- Installed SAP NetWeaver Developer Studio SAP NetWeaver CE 7.11
- Basic knowledge of Web Dynpro for Java

Details

Level of complexity: Beginner

Time required for completion: 60 min.

Author: Martin Clabunde

Company: SAP AG

Created on: 30th September, 2008

Author Bio

Martin Clabunde is studying computer science at the University of Applied Science Worms and works as a working student in Product Management for SAP NetWeaver User Interaction - Web Dynpro for Java.

Table of Contents

Overview

The objective of this tutorial is to give you a short introduction on how to develop your own Web Service in the Java Enterprise Edition 5 (Java EE) world using the Java Persistence API (JPA) and the Web Dynpro for Java framework on the presentation layer side. The scenario encapsulates the business logic as Web Service using the *Simple Object Access Protocol* (SOAP) to communicate with the clients over the net. The Web Service will neither being published to any repository nor references a Service Group since Logical Destinations became deprecated with Enhancement Package 1 for SAP NetWeaver 7.1 Java.

This tutorial focuses on a basic concept needed in a simple *local scenario* only.

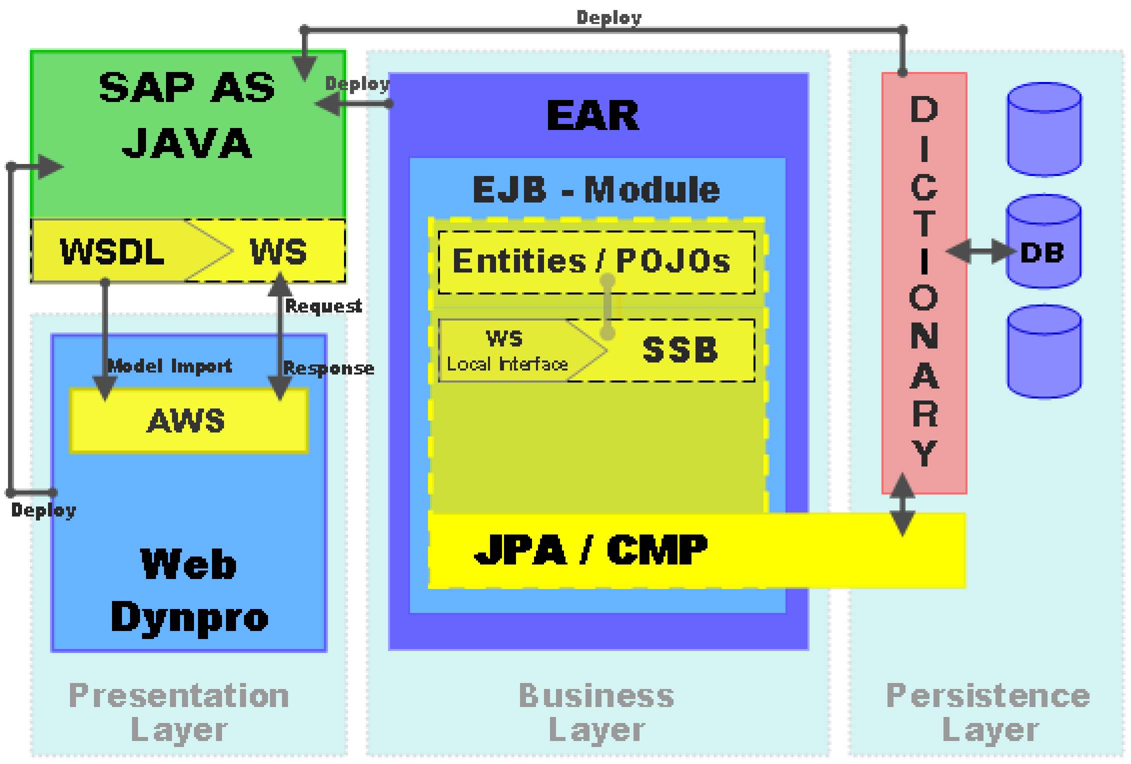
The Web Dynpro application which you are going to create lets you submit a simple form with two input fields. The submitted data then will be stored in the underlying default database and will be displayed in a Web Dynpro Table UI element using the Web Service methods implemented.

The illustration on the next side outlines a *simplified* overview on how the used components interact among each other. This illustration is not intended to cover the real world (environment and wiring).

For more detailed information please refer to the *Related Content* area at the end of this document. This section provides some hyperlinks to further reading in the SAP Developer Network (SDN) and the Developer's Guide in the SAP Library.

Note: This tutorial is only a simple *getting started* tutorial and is **not** intended for productive use.

Scenario



Starting with the Enterprise Java Bean 3.0 (EJB) Module Development Component (DC), the Java Persistence API (JPA) facet must be enabled first. The JPA then helps you managing the entities very comfortable. Entities are just Plain Old Java Objects (POJOs), i.e. Java classes. These entities are managed by the JPA within the EJB container. That scenario is called Container Managed Persistence (CMP).

Entities are controlled by the Entity Manager provided by the JPA framework. The instance of the Entity Manager is injected to a field of a Stateless Session Bean (SSB), where the business logic is implemented, and must be associated with a persistence context. The set of entities that can be managed by the given Entity Manager instance is defined by a persistence unit.

A client accesses the SSB thru its Service Endpoint Interface (SEI), which is the SSB's local interface. The transactional behavior can be configured slightly in a CMP scenario using annotation.

The EJB Module must reside in an Enterprise Application Archive (EAR) which can be deployed to the SAP Application Server (SAP AS) Java. Modules are not deployable standalone. Once the EAR is deployed the Web Service Description Language (WSDL) file is generated out of the annotations found in the entities, the SSB and its local interface. A Web Service is made of annotations only.

The Dictionary acts as an abstraction layer and is a generic metadata representation of a real data source. It provides special functionalities and is accessible thru a connection profile specified. Once it is deployed to the Application Server the defined table schema is applied to the real data source and can be accessed. The '*missing*' link between the underlying data source and the persistence context is described by a logical name, the data-source-alias which maps to a real data source name.

Once both DCs, the EAR and the Dictionary, are deployed to the SAP AS and the WSDL file is accessible via URL, the Web Dynpro DC will be created. The Adaptive Web Service (AWS) Model is then built up from the given WSDL file and the form can be arranged.

Glossary

SAP AS	= SAP Application Server
DB	= Database
EAR	= Enterprise Application Archive
EJB	= Enterprise Java Bean
SSB	= Stateless Session Bean
POJO	= Plain Old Java Object = Java Class
API	= Application Programming Interface
WS	= Web Service
AWS	= Adaptive Web Service Model
WSDL	= Web Service Description Language

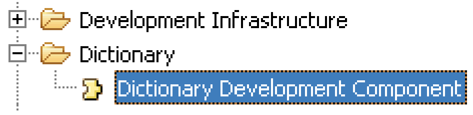
Create the Dictionary Development Component

The Dictionary Development Component (DC) acts as a platform-independent database abstraction layer that describes metadata for database objects and services. It also maps and manages your table definitions automatically to the underlying database once it is deployed.

It supports semantic verifications of data types and vendor-specific native SQL queries if needed. Naturally, the dictionary uses SQL/JDBC or Open SQL/SQLJ, which is limited to a subset of SQL statements to become more independent.

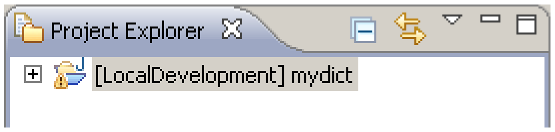
If you are running your application on the system data source using the Java Persistence Query Language (JPQL), as you are going to do in this tutorial, you must consider some restrictions on available statements.

1. To open the *Java EE* perspective, navigate **Window → Open Perspective → Other…→ Java EE**
2. To create a new *Dictionary DC*, choose
   File → New → Project… → Dictionary → Dictionary Development Component


Development Infrastructure
Dictionary
Dictionary Development Component

3. Press **Next >** open the **'Local Development'** branch and select the **MyComponents** Software Component (SC) and press **Next >**.
4. Name your new DC **mydict,** press **Next >** and confirm with **Finish**. Stay in the **Java EE Perspective**.

You should see your Dictionary DC **mydict** in the *Project Explorer* now.


Project Explorer
[LocalDevelopment] mydict

Create the database Connection Profile

To be able to establish a connection to your dictionary you have to create a *Connection Profile* of a specific type. In this tutorial you connect to the *Java Dictionary,* which is a metadata representation of your underlying database only.

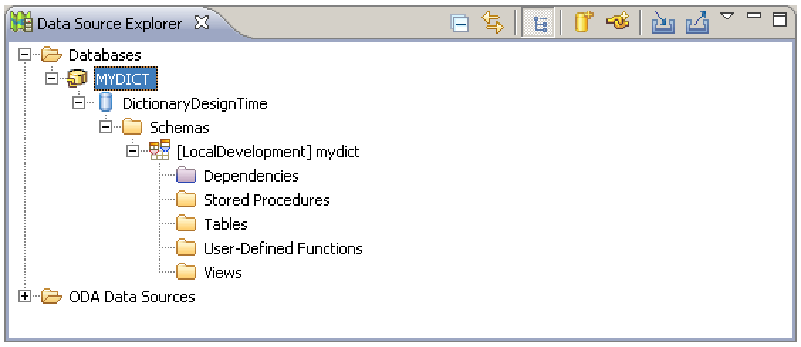
1. To open the *Data Source Explorer* View, select
   Window → Show View → Other…→ Connectivity → Data Source Explorer
2. To create a *new Connection Profile* right-click the **Databases** folder and choose **New…** from the context menu.
3. Select **Java Dictionary** profile type in the dialog window and press **Next >**.
4. Name the Connection Profile **MYDICT,** select **Auto-connect at startup** and press **Next >**.

Note: The Connection Profile name must be in **CAPITALS**

5. Browse for your dictionary DC **mydict** if it hasn't been chosen automatically yet and confirm with **Finish**.

The new Connection Profile **MYDICT** will be shown directly under the *Databases* branch.

The **Data Source Explorer** view provides information and functions about the connection profiles, their Schemas and creating new ones as well as support for the Data Definition Language (DDL) .



The Enterprise Java Bean 3.0 Module

The *Enterprise Java Bean 3.0 (EJB3.0) Module* is an essential part of the Java EE 5 development and the central point of your business logic. It is embedded in an EJB container that provides the runtime environment for Java Beans within the application server. An EJB Module cannot be deployed standalone; instead, you put it into a Java Enterprise Application Archive (EAR). This EAR then will be deployed to the engine.

The EJB container manages different kinds of Java Beans, such as message-driven Beans, entity beans, and session beans. It also has transactional support for persisting data. You can distinguish between Bean Managed Persistence (BMP) and Container Managed Persistence (CMP). IWith BMP, the developer must handle the transactional behavior manually within bean code. CMP is managed and optimized by the container.

Since EJB 2.1 was replaced by EJB 3.0 , things are much better. There is neither a need for home interfaces nor for the design patterns Data Access Object (DAO) and Data Transfer Object (DTO). These two patterns become obsolete with the EJB 3.0 standard.

Entity beans are treated like simple Java classes, also known as *Plain Old Java Objects* (POJOs). Additionally, entity instances become detachable from the EJB- Container. And last but not least, annotations were introduced. Annotations are prefixed with an @ character describing code-embedded metadata.

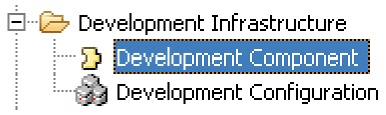
(Note: There are many more differences worth mentioning regarding EJB 3.0 – such as inheritance, polymorphism and the entire (very interesting) pattern-driven development. But this is not the intension of this tutorial. Please refer to the Related Content area for further reading).

Now, let's focus on the tutorial again and see how easy it is to get your data persisted using the Java Persistence API (JPA). The JPA framework is being used in the Java EE 5 EJB 3.0 environment for controlling persistence, transactional behavior, and Object Relational Mapping (ORM).

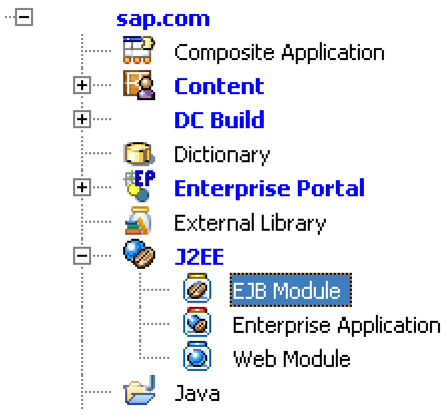
Create the EJB Module DC

1. In the **Java EE** perspective create a new Java EE **EJB Module** DC.

   File → New → Project… → Development Infrastructure → Development Component → Next >.

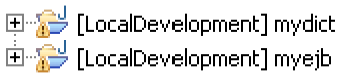


2. Select the **EJB Module** within the **J2EE** branch and press **Next >**.



3. Select the **MyComponents** DC and press **Next >**.
4. Name the DC **myejb,** choose **5.0** as *Java EE version* in the next window and confirm with **Finish**.

You should be able to see both DCs, **mydict** and **myejb,** in the *Project Explorer* view.



Enable the Java Persistence API Facet

Facets define characteristics and requirements for Java EE projects (apart from Java projects) and are used as part of the runtime configuration

Enabling the Java Persistence API (JPA) within your EJB Container gives you the option to let your entity beans be controlled by the *Entity Manager*. An instance of the Entity Manager will be directly *injected*[1] into a private field of your session bean at runtime and can then be used to manage your entities. The Entity Manager provides functions for creating, finding, removing, executing and fetching your named or dynamic queries (prepared statements). The transactional behavior can be controlled via annotation attributes or manually (Bean Managed Persistence).

[1] This technique is described by the design pattern *Dependency Injection*.

Enable the JPA Facet in the *Modify Faceted Project* dialog

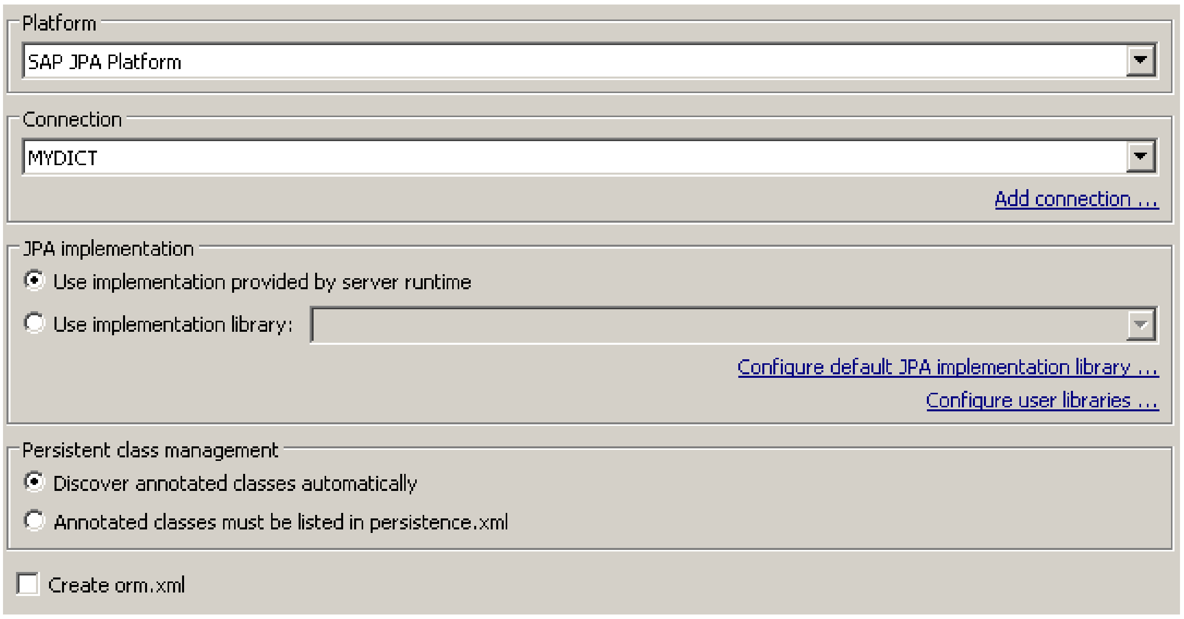
1. Select **myejb** DC, right-click it and select **Properties** from the context menu or press **Alt + Enter**.
2. Select **Project Facets** in the tree structure on the left hand side.
3. Press the **Modify Project…** button on the lower right hand side.
4. Select from the Project Facets list **Java Persistence** and press **Next >** to configure the JPA settings.

Configure the JPA Facet

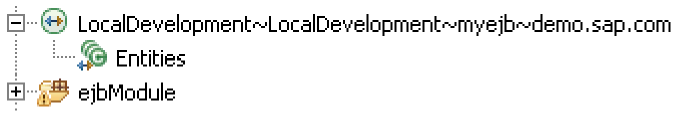
1. In the *JPA Facet* dialog choose **SAP JPA Platform** from the *Platform* dropdown list box. This will let you make use of the automatic creation of database tables from your defined entities enabled by the Data Definition Language (DDL) (also called forward mapping).
   There exist **naming conventions** for your table names using forward mapping.

   Table names:

   - must not exceed 18 characters
   - must be in capital letters
   - must have an prefix like TMP_
   - Column names must be in capitals.
2. As *Connection* choose your connection profile **MYDICT** if it is not suggested yet in the dropdown list.
3. If available, unselect the *Create orm.xml* option.
4. Leave the other default options unchanged, confirm with **Finish** and leave the *Properties* dialog with **OK**. The JPA Facet is then enabled and configured.


Platform: SAP JPA Platform
Connection: MYDICT
Add connection …
JPA implementation
[✓ Use implementation provided by server runtime]
Use implementation library:
Configure default JPA implementation library …
Configure user libraries …
Persistent class management
[✓ Discover annotated classes automatically]
Annotated classes must be listed in persistence.xml
☐ Create orm.xml

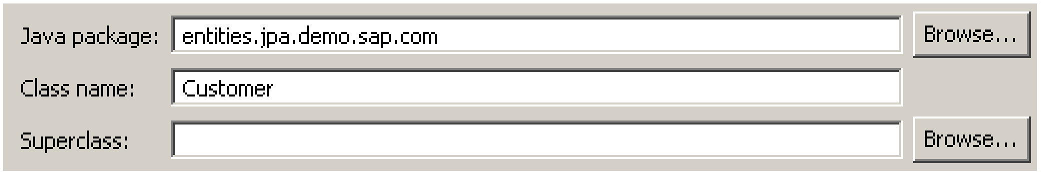
Browse your EJB Module DC structure. It should have an additional branch where all your entities then will be listed. Later on you will change this name of the node



Provide a JPA Entity (detailed)

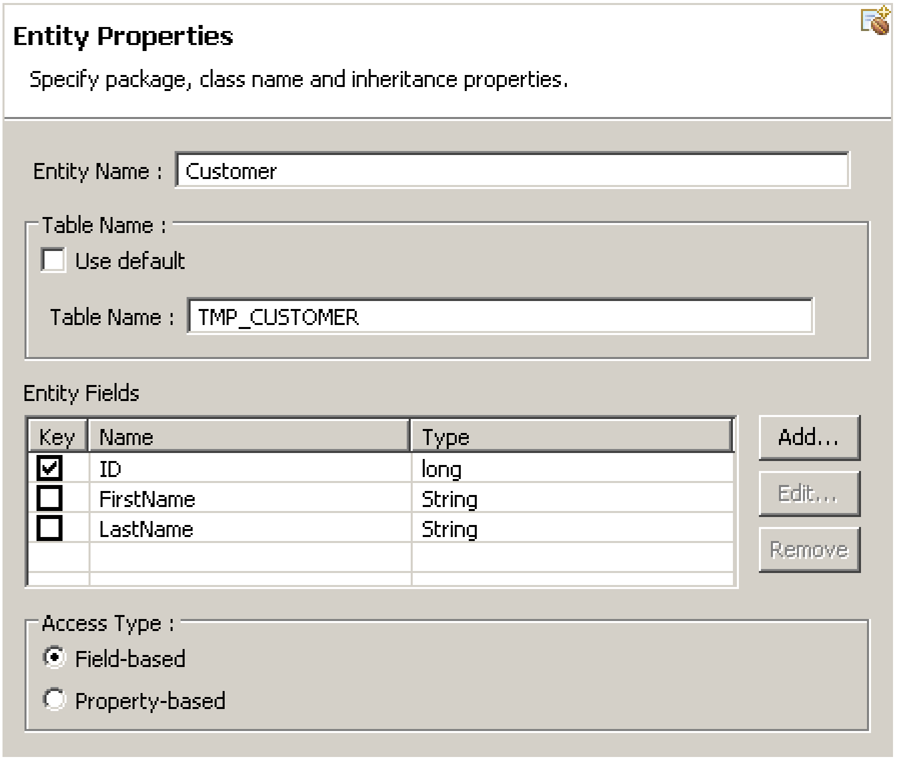
A JPA entity is simply an `@Entity` annotated POJO that implements the serializable interface so that it becomes detachable from the EJB Container.

1. Make sure that you are in the **Java EE** perspective.
2. Right-click the **myejb** DC and choose **New > JPA Entity** from the context menu. The *Entity Class* dialog opens.
3. Name the *Package* **entities.jpa.demo.sap.com** and your class **Customer** and press **Next >**.



Java package:	entities.jpa.demo.sap.com	Browse...
Class name:	Customer	
Superclass:		Browse...

4. In the *Entity Properties* dialog **deselect** the **Table name: Use default** option and provide a table name that f*ulfills the* naming convention, such as **TMP_CUSTOMER**.
5. Use the **Add…** *Button to provid*e some fields as shown in the next screenshot and select the **ID** field as *Primary* Key and confirm with Finish.



Entity Properties

Specify package, class name and inheritance properties.

Entity Name : Customer

Table Name :
[ ] Use default
Table Name : TMP_CUSTOMER

Entity Fields

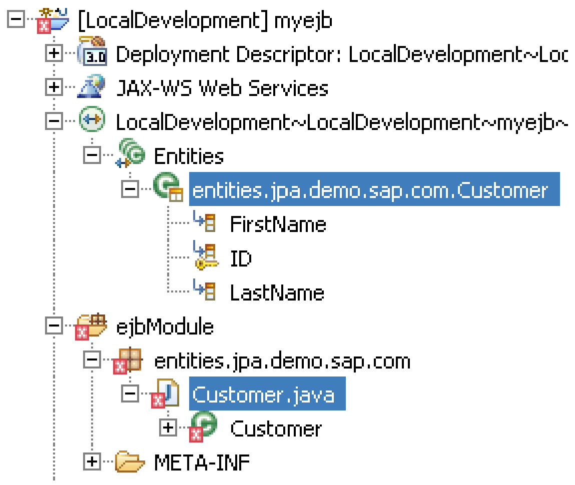
Key	Name	Type
✓	ID	long
☐	FirstName	String
☐	LastName	String

Access Type :
[✓ Field-based]
Property-based

The **Entity Properties** dialog helps you define your basic entity properties.

Your entity should be created now. Don't worry about the error that appears in the JPA Validator. This is due to the missing customer table definition in your dictionary. After applying some additional configuration steps to the entity, you are going to fix that using JPA-Tool, which generates and executes the appropriate DDL script.

Browse the **myejb** DC. It should looks like the following screenshot:

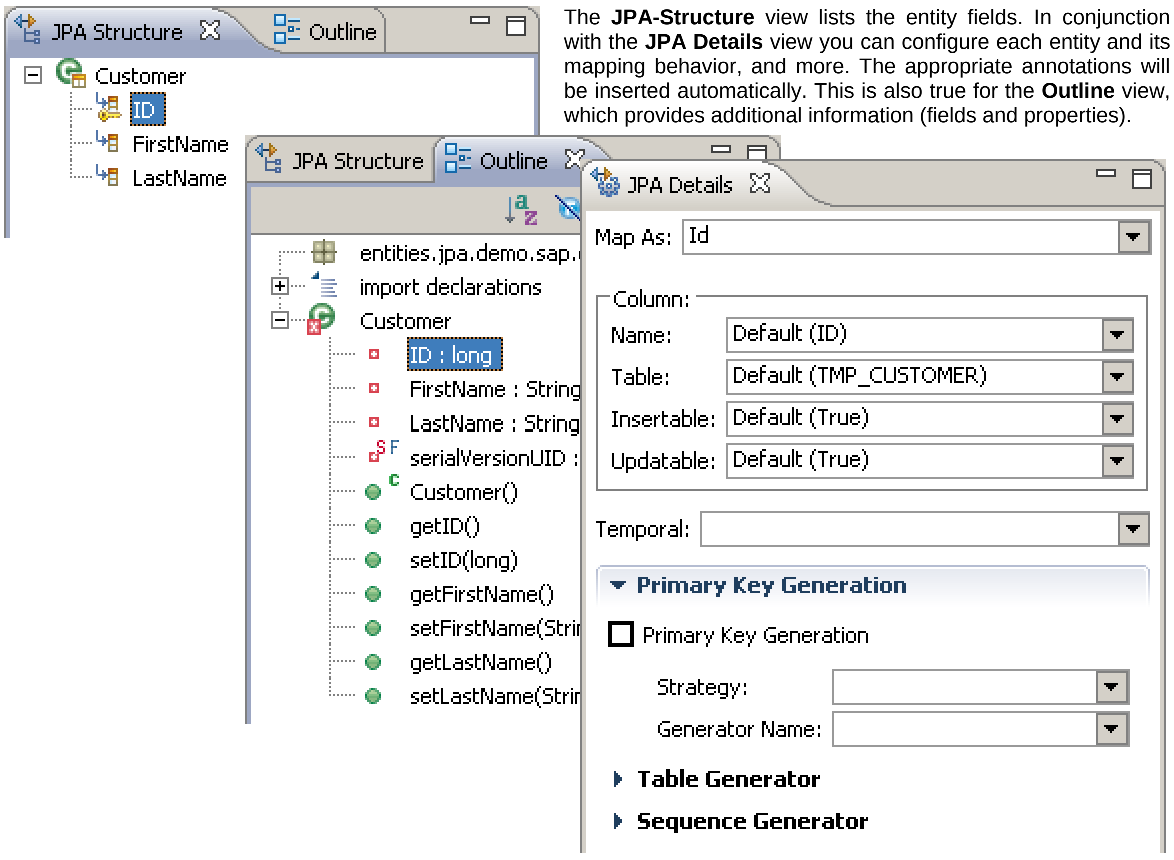

[LocalDevelopment] myejb
Deployment Descriptor: LocalDevelopment~Loc
JAX-WS Web Services
LocalDevelopment~LocalDevelopment~myejb~
Entities
entities.jpa.demo.sap.com.Customer
FirstName
ID
LastName
ejbModule
entities.jpa.demo.sap.com
Customer.java
Customer
META-INF

Configure the Entity in the JPA Perspective

The JPA perspective provides you with at least three important views so you can keep an eye on your entities and their relations: First, the **JPA Structure** view, second the **JPA Details** providing additional properties, and third the **Data Source Explorer**. To keep it simple, we will focus on one single entity with no relations to other entities.

1. To open the JPA-Perspective, choose *Window → Open Perspective → Other… → JPA Development.*
2. In the JPA Structure view select the **ID**.

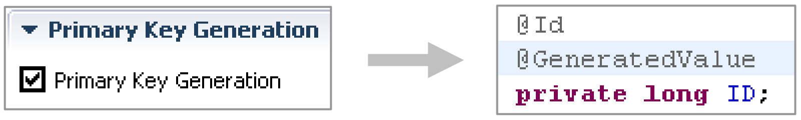
The **JPA-Structure** view lists the entity fields. In conjunction with the **JPA Details** view you can configure each entity and its mapping behavior, and more. The appropriate annotations will be inserted automatically. This is also true for the **Outline** view, which provides additional information (fields and properties).



Map As:	Id
Column:	
Name:	Default (ID)
Table:	Default (TMP_CUSTOMER)
Insertable:	Default (True)
Updatable:	Default (True)
Temporal:	
Primary Key Generation	☐
Strategy:	
Generator Name:	

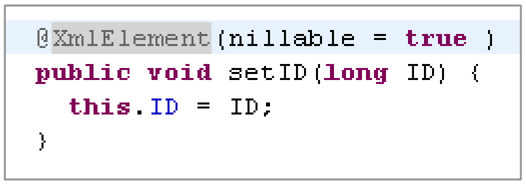
3. In the **JPA Details** view select **Primary Key Generation**.

   The corresponding annotation will be inserted in your entity code immediately.



```
@Id
@GeneratedValue
private long ID;
```

4. Switch to the **Outline** tab and select the **setID(long)** Method.
5. Insert the annotation **@XmlElement(nillable = true**) on top of the **setID()** method in the **Customer** entity code.



```
@XmlElement(nillable = true )
public void setID(long ID) {
  this.ID = ID;
}
```

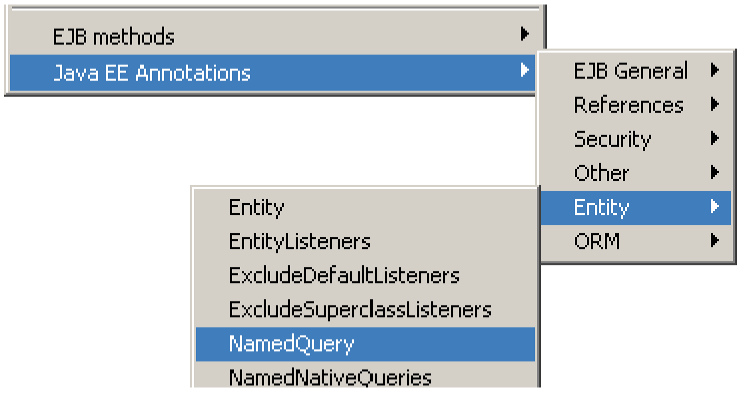
Since the *Primary Key Generation* is controlled by the JPA Framework, this annotation allows you to leave the ID field of your customer's object *null*. Hence, you don´t need to bind it to the context in Web Dynpro later on.

6. Copy and paste following code under the **@Table** annotation of the entity.

```
@XmlAccessorType(XmlAccessType.PROPERTY)
```

Note: This step is important because the *Java **A**rchitecture for **XML** **B**inding* (JAXB) layer expects XML annotations on public members by default. In the previous step you put an XML annotation in front of a *property*. At a later stage, when your Web Service is accessible via the URL, you can search within the generated WSDL file for the parameter **nillable="true"** if you want.

7. To insert a named query, select your **Customer** entity in the **Outline** view and choose **Java EE Annotations → Entity → NamedQuery**



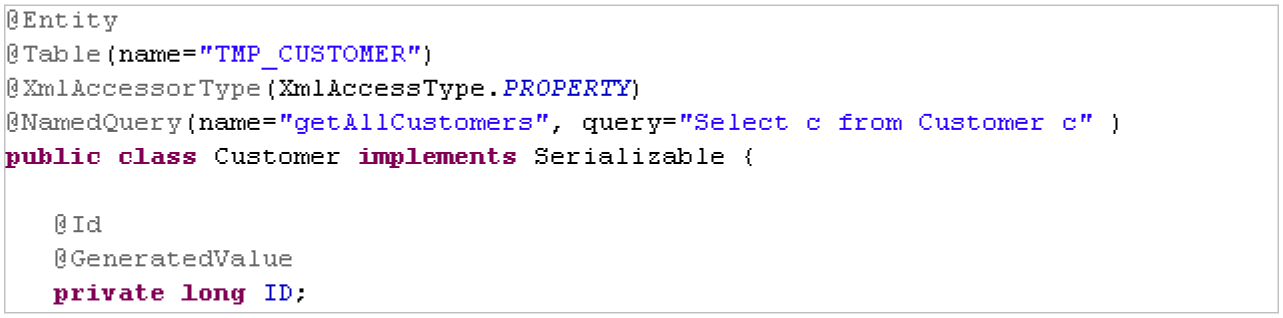
   This adds the **@NamedQuery** annotation with default placeholders right on top of your class definition.

8. Specify the **query** parameter to fetch all available customers in the **@NamedQuery(…)**.

   …query = "**Select c from Customer c**"…

9. Name your query "**getAllCustomers**".
10. Delete the **hints** parameter - you won't need it in this scenario.
11. **Organize Imports** by pressing **Ctrl+Shift+O**.
12. Safe your entity by choosing **Save All** from the *File* menu or by pressing **Ctrl+Shift+S**.

Your entity's definition should look like the screenshot section below, apart from the package and import statements and methods:



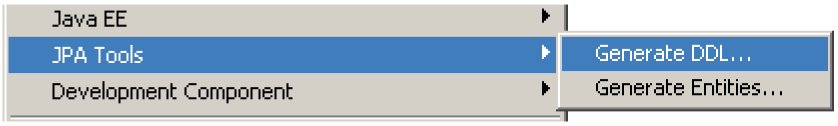
```
@Entity
@Table(name="TMP_CUSTOMER")
@XmlAccessorType(XmlAccessType.PROPERTY)
@NamedQuery(name="getAllCustomers", query="Select c from Customer c" )
public class Customer implements Serializable {

    @Id
    @GeneratedValue
    private long ID;
```

Apply Forward Mapping

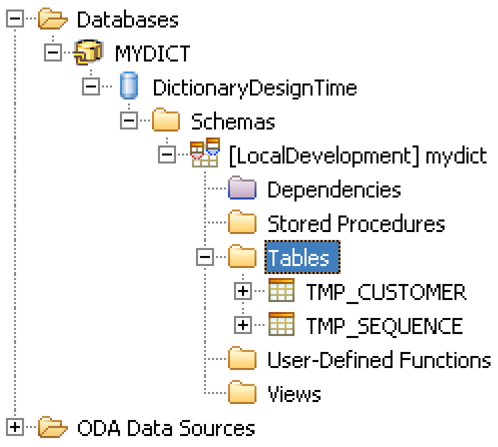
You can use the DDL to create your dictionary schema from your JPA entities and vice versa - that is, *Reverse Mapping*. In this tutorial we use *Forward Mapping* (Entities to Tables by DDL).

1. Right-click **myejb** DC and select **JPA Tools > Generate DDL…** from the context menu.



2. The *Summary* dialog should open. Confirm with **OK**.

In the **Data Source Explorer** browse your connection profile **MYDICT** for the generated table definitions.



The EJB Session Bean 3.0

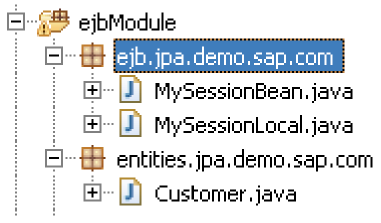
You will create a *stateless* session bean implementing two simple business methods to set a customer and retrieve a list of customers. These two methods would then be added to the bean's local interface.

- setCustomer(Customer c) : void
- getCustomers() : List<Customer>

Create the EJB Session Bean 3.0

1. Switch back to the jJava EE perspective, right-click **myejb** DC and select **New → EJB Session Bean 3.0** from the context menu.
2. Name the *EJB Class* **MySessionBean** and the *EJBPackage* **ejb.jpa.demo.sap.com**.
3. *Session* and *Transaction Type* leave as suggested: **Stateless** Bean, **Container** Managed.
4. As *Business Interface* leave as suggested *Local.*
5. Confirm with **Finish**.

The skeleton of your session bean and its local interface is generated within your *ejbModule*.



```
ejbModule
  ejb.jpa.demo.sap.com
    MySessionBean.java
    MySessionLocal.java
  entities.jpa.demo.sap.com
    Customer.java
```

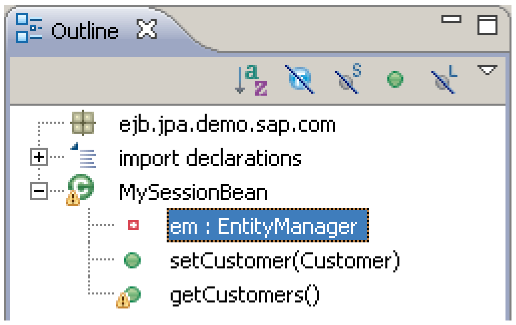
Implement the business methods (detailed)

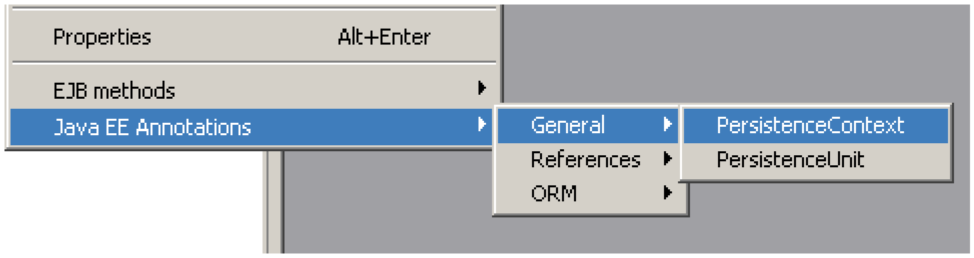
1. Open the **MySessionBean** in the Java Editor and copy and paste following code to its body:

```
private EntityManager em;

public void setCustomer( Customer c )
{
      em.persist( c );
}
public List<Customer> getCustomers()
{
      Query q = em.createNamedQuery("getAllCustomers");
      List<Customer> customers = q.getResultList();
      return ( customers  );
}
```

2. In the Outline view, right-click the **em : EntityManager** entry listed within your session bean and choose *Java EE Annotations → General → PersistenceContext* from the context menu.





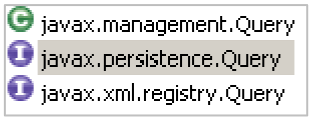
The annotation will be inserted in the line of code associated with the selection in the *Outline*.

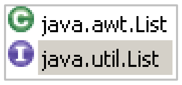
```
@PersistenceContext(unitName="myUnitName", type=PersistenceContextType.TRANSACTION)
private EntityManager em;
```

3. 3. Change the *unitName* parameter of the *@PersistenceContext* annotation from its default value to **myUnitName**.

Note: This name identifier is case sensitive and will be specified in the persistence.xml file later on.

4. Organize imports using **Ctrl+Shift+O** and select the appropriate imports as shown in the screenshots:

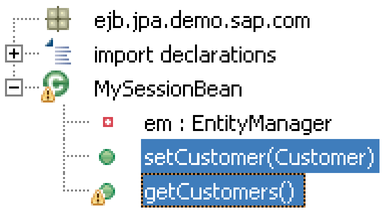
   a. 

   b. 

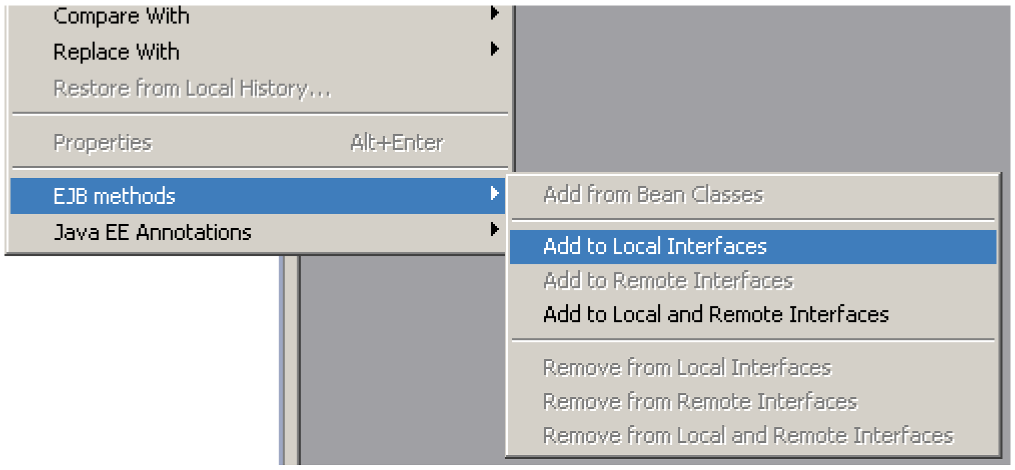
5. Save All: **Ctrl+Shift+S**

Add the business methods to the local interface

1. Select **both** methods in the **Outline** view by holding down the **Ctrl** key.



2. Right-click one of the selected methods without losing the selection and choose **EJB methods → Add to Local Interfaces** from the context menu.


Compare With
Replace With
Restore from Local History...
Properties Alt+Enter
EJB methods
Java EE Annotations
Add from Bean Classes
Add to Local Interfaces
Add to Remote Interfaces
Add to Local and Remote Interfaces
Remove from Local Interfaces
Remove from Remote Interfaces
Remove from Local and Remote Interfaces

The selected methods will be added to the local interface of your session bean.
Open the the Java Editor and have a look at your session bean's local interface.

```
@Local
public interface MySessionLocal {

    public void setCustomer (Customer c);

    public List<Customer> getCustomers ();

}
```

3. Save All. (**Ctrl+Shift+S)**

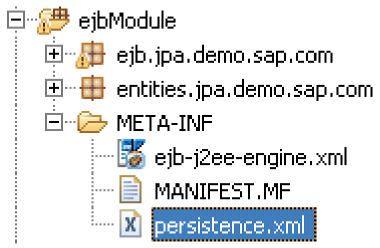
Configure the Persistence Unit Name and the *jta-data-source*

The *Persistence Unit* element is specified in the persistence.xml file found in the META-INF folder of the *ejbModule.* The *name* attribute specifies the unit's name and is referenced by the *@PersistenceContext* annotation in your session bean.

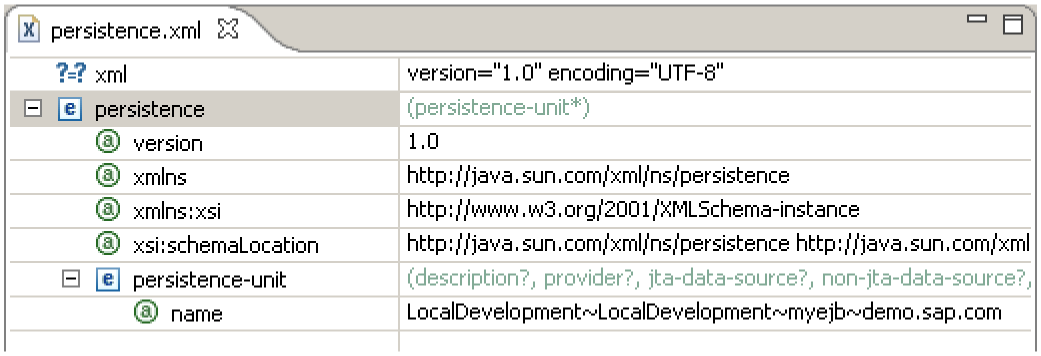
Additional elements and attributes can be added using the context menu. Hence you insert a *Java Transaction Data Source* (*jta-data-source*) child element to the persistence- unit element. This allows you to link to your data source through the corresponding data source alias, specified in the Application Enterprise Archive DC. This procedure is explained in the next chapter.

Both values are used to dissolve the relations between the persistence context and the data source and therefore they are case sensitive.

1. Open the **persistence.xml** file in the **META-INF** folder within your *ejbModule* in the *Xml-Editor* by choosing **Open** from the context menu or press the **F3** key.

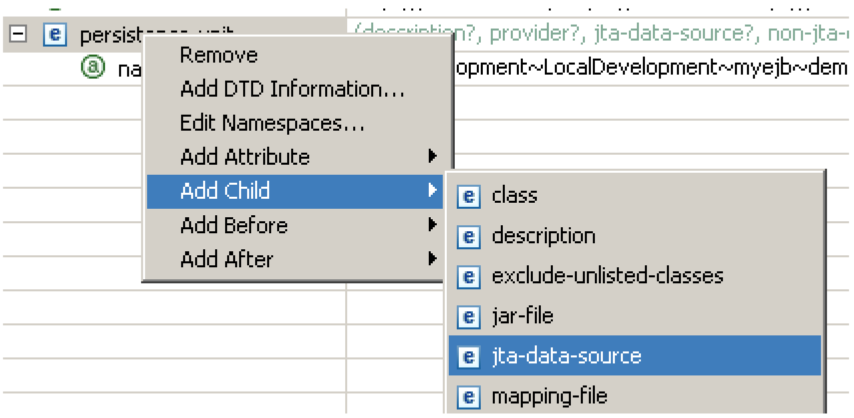


2. Expand the **persistence** and **persistence-unit** elements e in the Xml-Editor.

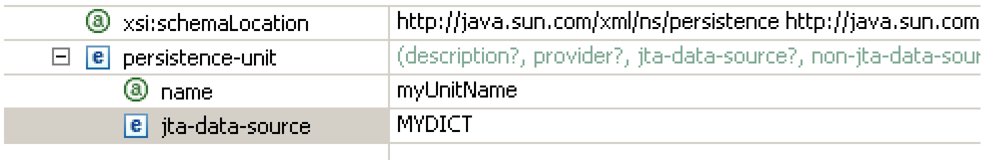


?-? xml	version="1.0" encoding="UTF-8"
persistence	(persistence-unit*)
version	1.0
xmlns	http://java.sun.com/xml/ns/persistence
xmlns:xsi	http://www.w3.org/2001/XMLSchema-instance
xsi:schemaLocation	http://java.sun.com/xml/ns/persistence http://java.sun.com/xml
persistence-unit	(description?, provider?, jta-data-source?, non-jta-data-source?,
name	LocalDevelopment~LocalDevelopment~myejb~demo.sap.com

3. Right-click the **persistence-unit** element and add the **jta-data-source** element by choosing **Add Child → jta-data-source** from the context menu.



4. Change the value of the persistence-unit element **name** attribute ⓐ to **myUnitName**.
5. Change the **jta-data-source** value to **MYDICT**.



xsi:schemaLocation	http://java.sun.com/xml/ns/persistence http://java.sun.com
persistence-unit	(description?, provider?, jta-data-source?, non-jta-data-sour
name	myUnitName
jta-data-source	MYDICT

The Java EE Enterprise Application Archive

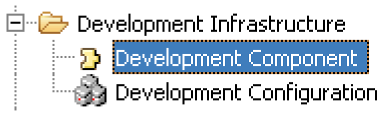
In an Enterprise Application Archive (EAR) you can configure dependencies and references to other Java EE DCs and modules. Contained modules are then deployed together with the EAR.

Now you are going to reference your EJB Module from the EAR DC and set up the *SAP Data Source Aliases Provider Module* in order to specify the *data-source-alias* that represents a logical name which is mapped to the name of a real data source.

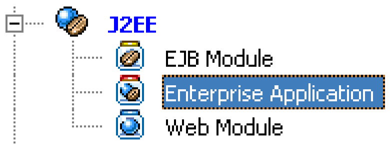
You also need to specify the *data-source-alias,* This represents a logical name which is mapped to the name of a real data source. You do this by set up the *SAP Data Source Aliases Provider Module.*

Create an Enterprise Application Archive

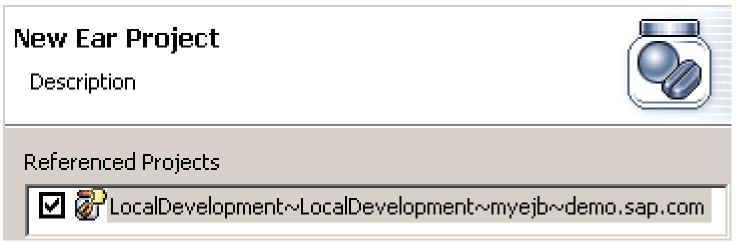
1. **File → New → Project… → Development Infrastructure → Development Component → Next >**.



2. Select the **Enterprise Application** in the **J2EE** branch and press **Next >**.



3. Select the **MyComponents** SC and press **Next. >.**
4. Name the new DC **myear** and press **Next >** twice.
5. Select **myejb** DC as *Referenced Project* and confirm with **Finish**.



Referenced Projects

☑ LocalDevelopment~LocalDevelopment~myejb~demo.sap.com

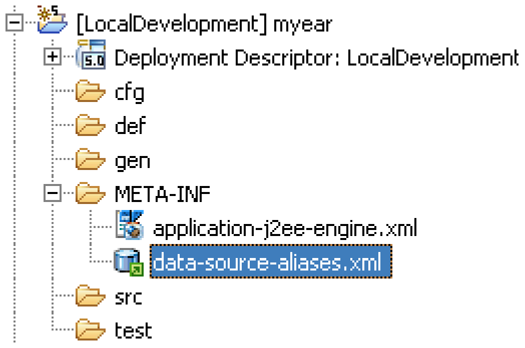
Enable the SAP Data Source Aliases Provider Module

1. Select **myear** DC, right-click it and select **Properties** from the context menu,or press **Alt+Enter**.
2. Select **Project Facets** in the tree structure on the left hand side.
3. Click the **Modify Project…** button on the lower right hand side.
4. Select from the *Project Facets* list **SAP Data Source Aliases Provider Module** and confirm with **Finish**.

The **data-source-aliases.xml** is added to the META-INF folder of the **myear** DC.

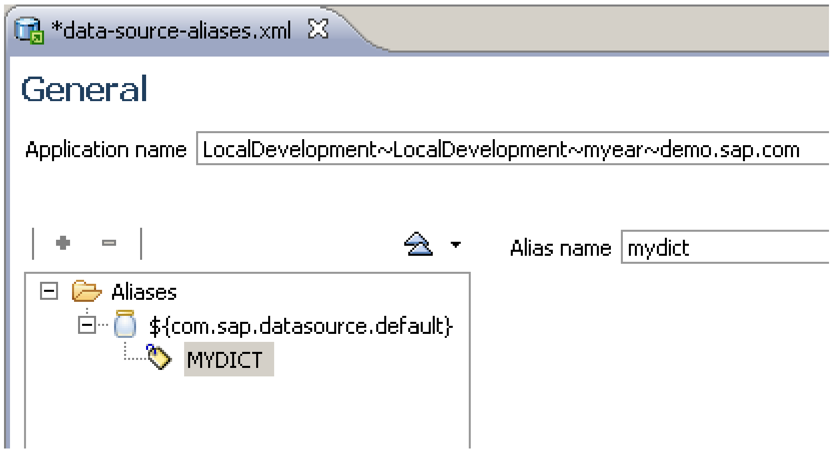
Create a Data Source Alias

1. Browse to the META-INF folder in the **myear** DC.



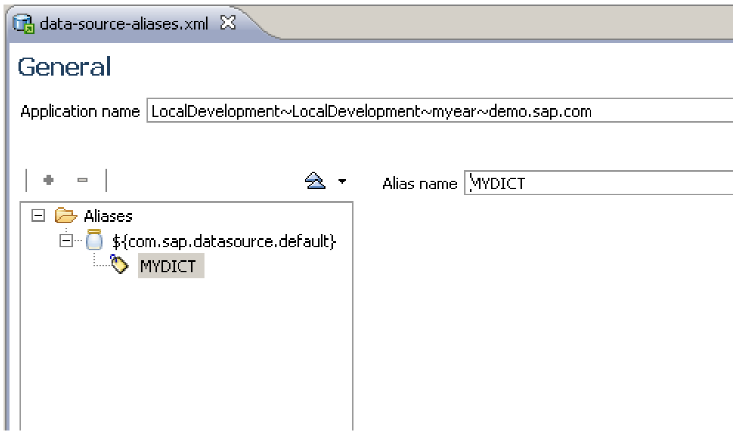
2. Open the **data-source-aliases.xml** file by choosing **Open** from the context menu or press **F3** key.

   Expand the *Aliases* folder to the default **ALIAS** and change its **name** to **MYDICT**.



Application name LocalDevelopment~LocalDevelopment~myear~demo.sap.com

Alias name mydict



Application name LocalDevelopment~LocalDevelopment~myear~demo.sap.com

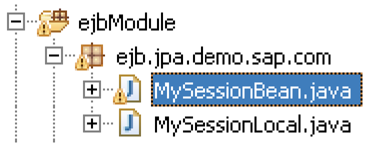
Alias name MYDICT

3. Save All. **Ctrl+Shift+S**

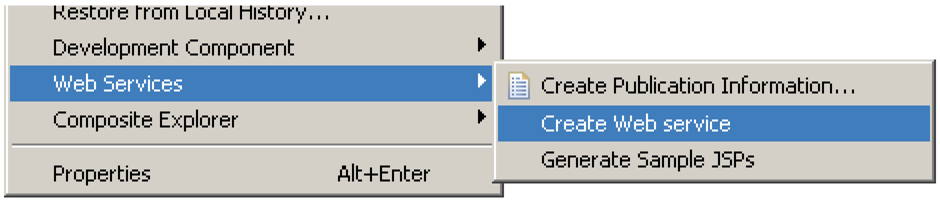
Expose the session bean's local interface as a Web Service

A Web Service client accesses the exposed business methods of a stateless session bean through its local interface - that is, the Service Endpoint Interface (SEI). The Developer Studio provides a wizard that guides you through the necessary configuration steps. The required annotations are added automatically to the code.

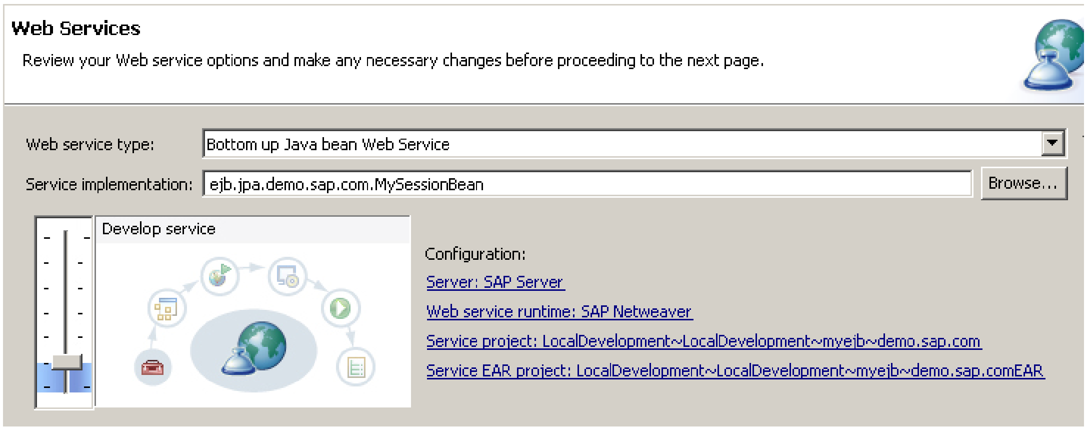
1. Open the **ejbModule** branch in the **myejb** DC.



2. Right-click your session bean **MySessionBean** and choose **Web Services → Create Web Service** from the context menu.



3. In the *Web Services* dialog set the Slider value to **Develop Service**.



Web Services

Review your Web service options and make any necessary changes before proceeding to the next page.

Web service type: Bottom up Java bean Web Service

Service implementation: ejb.jpa.demo.sap.com.MySessionBean

Develop service

Configuration:

Server: SAP Server

Web service runtime: SAP Netweaver

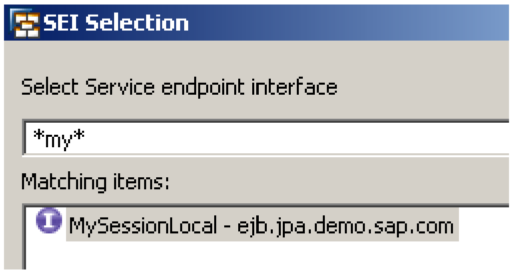
Service project: LocalDevelopment~LocalDevelopment~myejb~demo.sap.com

Service EAR project: LocalDevelopment~LocalDevelopment~myejb~demo.sap.comEAR

4. Leave the default settings unchanged and press **Next >**.

5. Now, expose its method signatures as the service endpoint , by selecting the **Specify existing interface** option and **Browse** for your bean's local interface.

   I. Filter for your bean's interface by typing ***my*** into the filter input field.



SEI Selection

Select Service endpoint interface

my

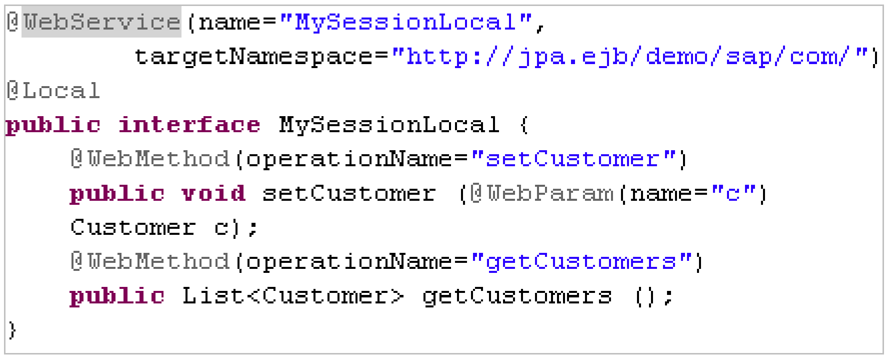
Matching items:

MySessionLocal - ejb.jpa.demo.sap.com

   II. Select the found **MySessionLocal** interface and press **OK** to close the dialog.

6. Back in the *Web Service* dialog go for *the Web Service customizations* dialog by pressing **Next >**.
7. Leave the default values unchanged and **note** your **WSDL URL preview** if you want.

   http://<host>:<port>/MySessionService/MySessionBean?wsdl
8. Confirm all dialogs with **Finish.**
9. Have a look at your bean and its local interface. The corresponding annotations should be added.



```
@WebService(name="MySessionLocal",
        targetNamespace="http://jpa.ejb/demo/sap/com/")
@Local
public interface MySessionLocal {
    @WebMethod(operationName="setCustomer")
    public void setCustomer (@WebParam(name="c")
    Customer c);
    @WebMethod(operationName="getCustomers")
    public List<Customer> getCustomers ();
}
```

10. Save All. **Ctrl+Shift+S**

Note: At a later stage, when you develop your first larger scenario with more than a single entity and more complex relationships, you have to consider that there are some limitations using Web Services in conjunction with ORM i. e **Object/Relational Impedance**. All this means is that if you are going to try to build up an XML/SOAP file from particular entity relationships you will face an infinite xml file structure. The JAXB layer redeems that with an appropriate exception message. Typically this occurs on bidirectional relations between entities. You can avoid this by hiding the relevant fields and/or properties with the `@XmlTransient` annotation. However, this may not be sufficient. Please refer to further reading.

Deploy the Dictionary DC

As soon as you deploy the dictionary DC to the SAP Application Server, the table's schema of your connection profile is applied to the underlying database. In this tutorial the SAP default data source. The deployment must be successful.

1. To deploy your dictionary DC to the SAP Application Server: Select the **mydict** DC and choose **Development Component → Deploy…** from the context menu.

Deploy the Enterprise Application Archive DC

While your EAR is being deployed, the dependent modules are built and deployed as well. If a referenced EJB Module defines a Web Service, the WSDL file will be generated automatically. The deployment must be successful.

1. To deploy your Enterprise Application DC to the SAP Application Server select the **myear** DC and choose **Development Component → Deploy…** from the context menu.

Consuming your Web Service in Web Dynpro

In Web Dynpro you consume Web Services using the Adaptive Web Service Model (AWS). The model is built up from the given WSDL file from the URL or file system at design-time. In this tutorial you will access your Web Service directly, via the given WSDL URL from *Remote Location*.

Create the Web Dynpro DC

1. Switch to the **Web Dynpro** perspective.

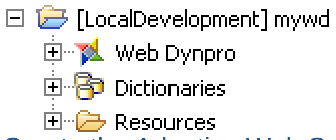
   Window → Open Perspective → Other…→ Web Dynpro

2. Create a new Web Dynpro DC.

   File → New → Web Dynpro Development Component

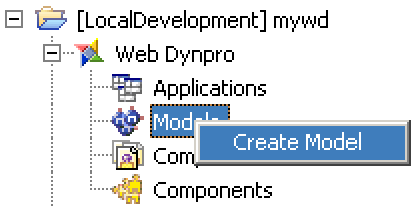
3. Open the **'Local Development'** branch**,** select the **MyComponents** SC and press **Next >**.
4. Name your new DC **mywd,** press **Next >** and confirm with **Finish**.

You should see your Web Dynpro DC in the *Web Dynpro Explorer* view now.

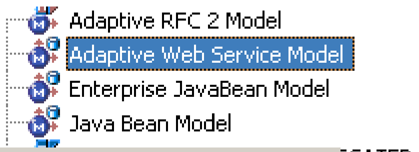


Create the Adaptive Web Service Model

1. Right-click the **Models** node within the Web Dynpro branch and select **Create Model** from the context menu.

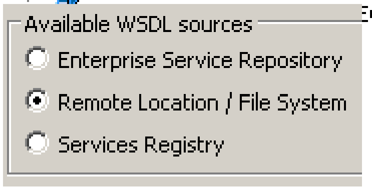


2. Select the **Adaptive Web Service Model** as model type and press **Next >**.



3. Select as *Available WSDL sources*

   Remote Location / File System and press **Next >**.


Available WSDL sources

( ) Enterprise Service Repository
(•) Remote Location / File System
( ) Services Registry

4. Provide the URL to your WSDL and press **Next >**.

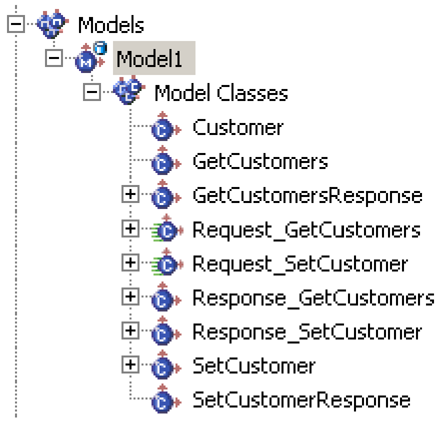
   http://<host>:<port>/MySessionService/MySessionBean?wsdl

Note: If you are accessing the internet via proxy, this must be configured accordingly under preferences.

5. In the *Create Service Group* dialog select **No service group configuration** and press **Next >**.
6. In the *Logical Destinations* dialog select **No logical destinations** and confirm with **Finish**.

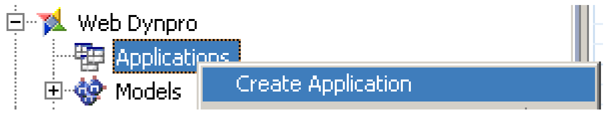
Note: In this tutorial you do not need to define either logical destinations or service groups. But bear in mind that since *Logical Destinations* are deprecated, it is recommended that you make use of *Service Groups* instead. Service Groups for *consumer applications* are maintained in the Application Communication module of the NetWeaver Administrator. A physical *Provider System* must be assigned to the relevant Service Group.

Your Adaptive Web Service Model has been imported now.

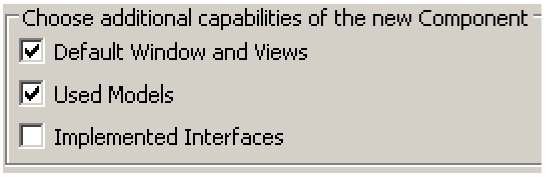


Create the Web Dynpro Application with default components

1. Right-click the **Applications** node and select **Create Application** from the context menu.



2. Name your application **MyApp** and press **Next >** twice.
3. In the *Specify Component Properties* dialog select the **Used Models** option and press **Next >** twice.



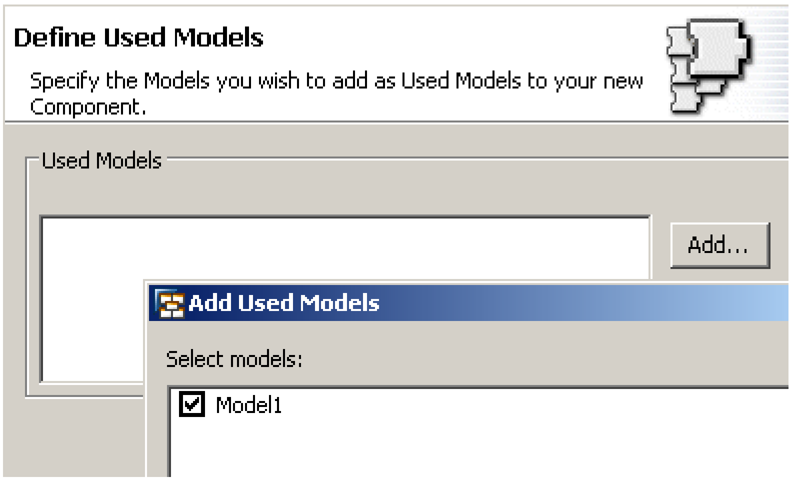
Choose additional capabilities of the new Component

[✓] Default Window and Views

[✓] Used Models

[ ] Implemented Interfaces

4. In the *Define Used Models* dialog press the **Add…** button and select your **Model1**.



Define Used Models

Specify the Models you wish to add as Used Models to your new Component.

Used Models

Add…

Add Used Models

Select models:

[✓] Model1

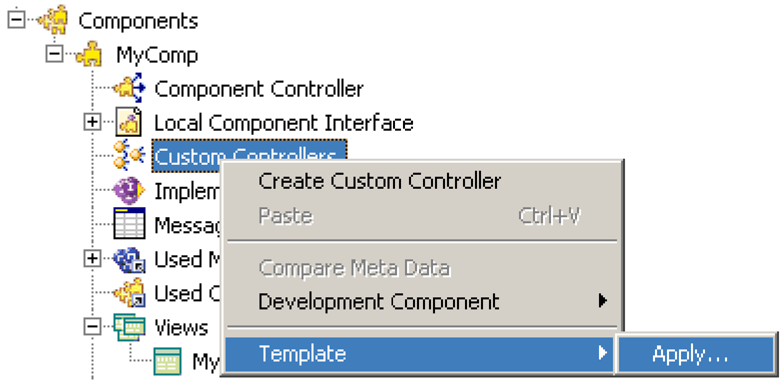
5. Press **Ok** and confirm with **Finish** to close the wizard.

Apply the Service Controller Code Template

The *Service Controller Code Template* can be used to bind your model nodes to the controller's context automatically. You select the relevant *Request model class* and the code will be created in the *wdDoInit()* hook method. You must repeat these steps for each request model class you would use. In this tutorial you use two *Request model classes*: getCustomers(), setCustomer().

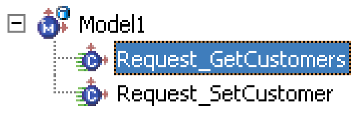
1. Right-Click the **Custom Controller** node in your MyComp component and choose

   Template → Apply… from the context menu.

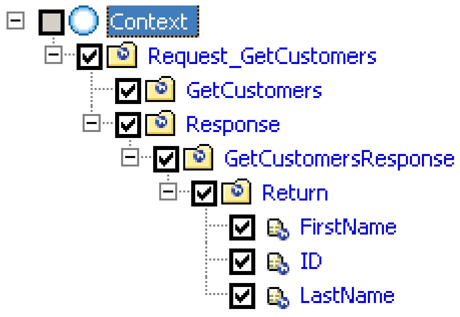


2. Select the **Service Controller** template from the dialog that opens and press **Next >**.

3. Select the **Request_GetCustomers** model class and press **Next >**.

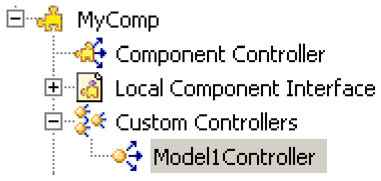


4. Select **all** available context elements and press **Next >**.



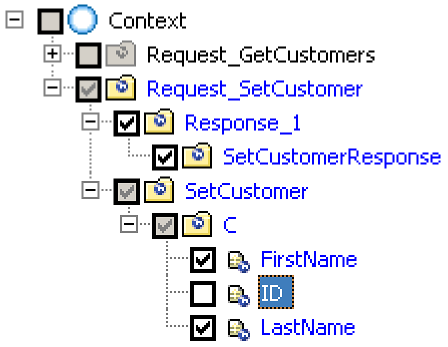
5. The template even provides an execution method for the request model class. Leave the proposal unchanged and confirm with **Finish**.

6. Right-click the automatically created **Model1Controller** and choose **Template → Apply…** from the context menu again.



7. Select the **Service Controller** icon and press **Next >**.

8. Select the **Request_SetCustomer** model class and press **Next >**.
9. Select all available context elements except for the customer object **ID** as shown in the screenshot below. The customer ID is managed by the JPA framework and will be set automatically.
10. Press **Next >**.



11. Confirm with **Finish**.

To fix the **exception** raised go to the *Java Editor* of **Model1Controller** and delete the redundant model instantiation.

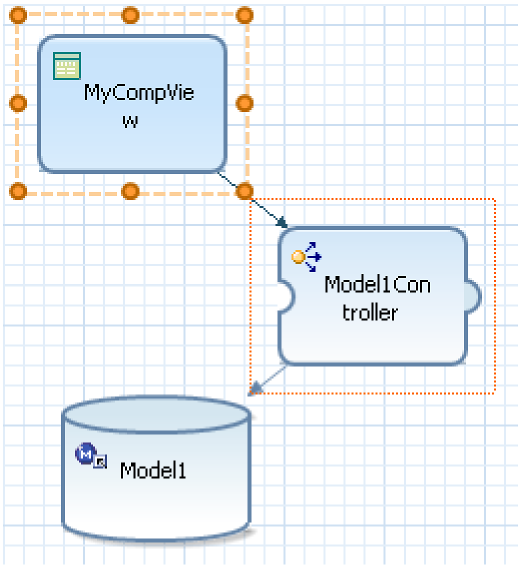
1. Right-click the **Model1Controller** and select **Open → Java Editor** from the context menu.
2. Go to the **wdDoInit()** method and **delete** the red marked line.

```
Model1 model1Model = new Model1();
```

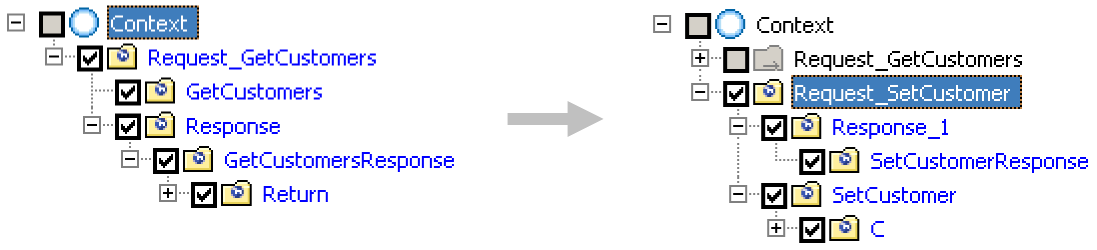
3. Save All. **Ctrl+Shift+S**

Map the View Controller Context to the Custom Controller Context

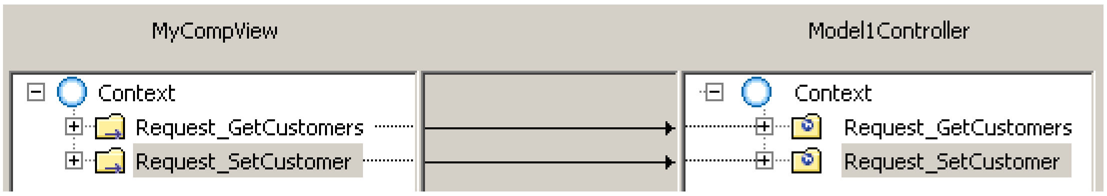
1. Open the *Data Modeler* of your component by choosing **MyComp → Open Data Modeler** from the context menu.
2. Draw a *Data Link* from **MyCompView** to **Model1Controller**.



3. Drag **both** Request-Nodes, one after another, from the right to the left pane and drop them onto the **MyCompView** context-root node and select all available nodes as shown in the screenshots below.



Result: The context of your view controller context is mapped to the Model1Controller context.



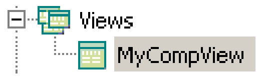
4. Confirm with **Finish**.

Layout the view using templates

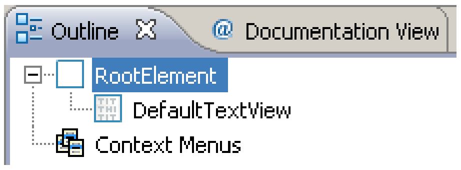
To be able to display a list of existing customers and to add new ones, you will provide a Web Dynpro table UI element and a simple form with an action button. The table will be supplied by the response result when the service's getCustomers() method is executed. The button action executes the setCustomer() method. Using the *Table* and *Form* templates your basic layout is arranged very quickly.

Display available customer records in a Table UI element

1. Open the *View Editor* of your view **MyCompView** by choosing **Open → View Editor** from the context menu.

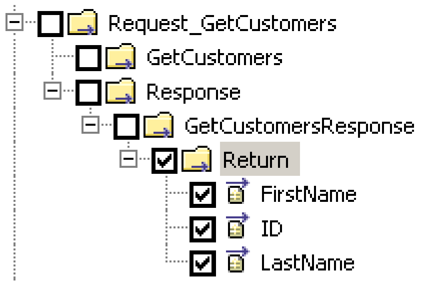


2. In the *Outline* view select the **RootElement** and choose **Apply Template** from the context menu.

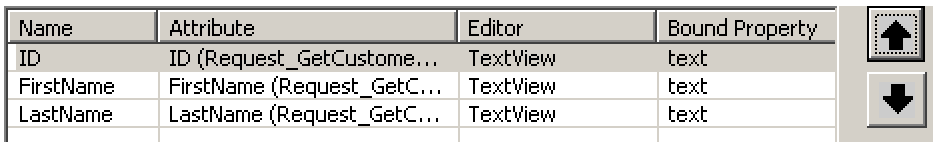


3. Select the **Table** template from the *Select Template* dialog that opens and press **Next >**.

4. Select the **Request_GetCustomers → Response → GetCustomersResponse → Return** node and press **Next >**.

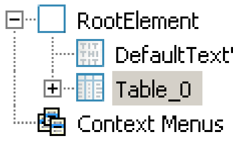


5. **Sort** the table columns by your needs using the **up/down** buttons and confirm with **Finish**.



Name	Attribute	Editor	Bound Property
ID	ID (Request_GetCustome...	TextView	text
FirstName	FirstName (Request_GetC...	TextView	text
LastName	LastName (Request_GetC...	TextView	text

The Table UI element should be shown in *Outline* view now.

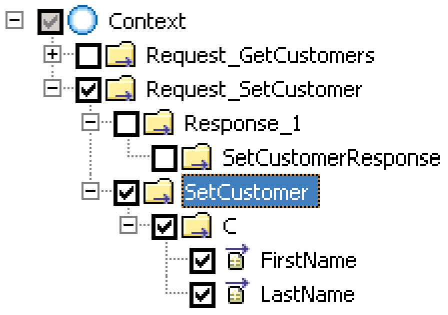


6. Open **MyCompView** in the *Java Editor*. Choose **Open → Java Editor** from the context menu and copy and paste following code between the @begin and @end tags within the **wdDoModifyView()**-method.

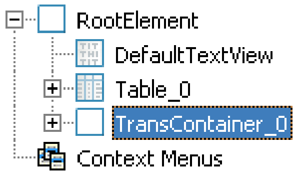
```
if( firstTime ){
      wdThis.wdGetModel1ControllerController().executeGetCustomers();
}
```

Provide an input form to add a new customer

1. Open the *View Editor* of **MyCompView** again.
2. In the *Outline* view select the **RootElement** and choose **Apply Template** from the context menu.
3. Select the **Form** template and press **Next >**.
4. Select the **Request_SetCustomer → SetCustomer** node and confirm with **Finish**.

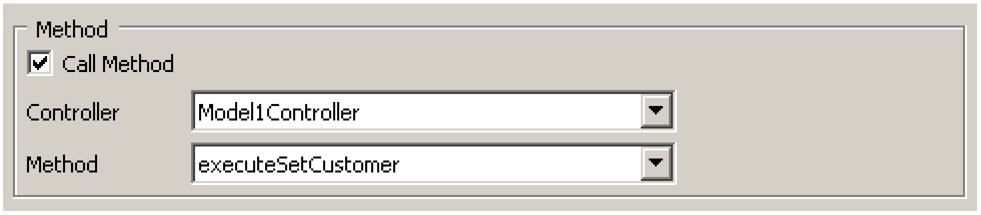


The Form will be created in a Transparent Container UI element and added to the *Outline* view.



Add an Action Button to the input form to submit the data

1. In the *Outline* view select the **TransContainer_0** UI element and choose **Apply Template** from the context menu.
2. Select the **Action Button** template and press **Next >**.
3. Change the Button Label to **Save Customer** and press **Next >**.
4. Select the **Call Method** option to associate the button action with an existing function.
5. Choose from the two dropdown list boxes: **Model1Controller** as *Controller* and **executeSetCustomer** as *Method* and confirm with **Finish**.



Method
[✓] Call Method

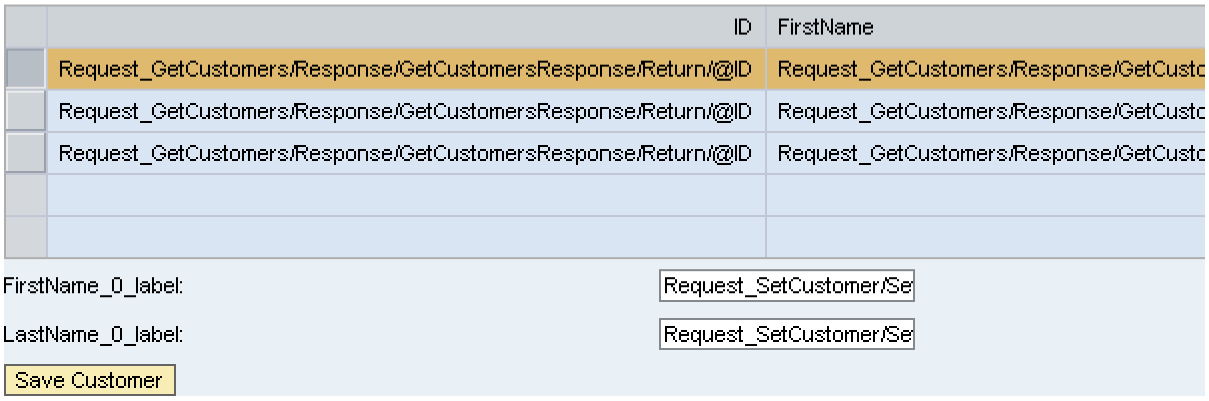
Controller	Model1Controller
Method	executeSetCustomer

5. Navigate to the button action implementation **onActionSaveCustomer(…)** and insert the following line of code right after the …**executeSetCustomer()** method call.

```
wdThis.wdGetModel1ControllerController().executeGetCustomers();
```

6. In the *Outline* view expand the **TransContainer_0** UI element and switch to the **Properties** tab in the lower screen area.
7. Change the *layout* property of *TransContainer_0* to **MatrixLayout**.
8. Set the each *layout* property of **FirstName_0_label**, **LastName_0_label** and the **Button** to **MatrixHeadData**. Multiple selections are also possible by holding the **Ctrl** key while selecting each UI element in the *Outline* view.
9. **Delete** the **DefaultTextView** element in the *Outline* view.
10. Save All. **Ctrl+Shift+S**

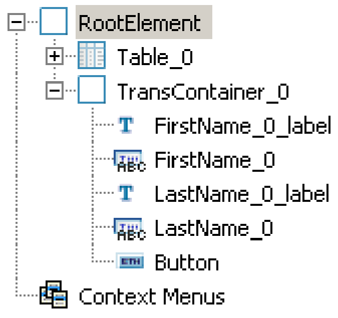
Your layout should look like the screenshot below:



ID	FirstName
Request_GetCustomers/Response/GetCustomersResponse/Return/@ID	Request_GetCustomers/Response/GetCusto
Request_GetCustomers/Response/GetCustomersResponse/Return/@ID	Request_GetCustomers/Response/GetCusto
Request_GetCustomers/Response/GetCustomersResponse/Return/@ID	Request_GetCustomers/Response/GetCusto

FirstName_0_label:	Request_SetCustomer/Se
LastName_0_label:	Request_SetCustomer/Se

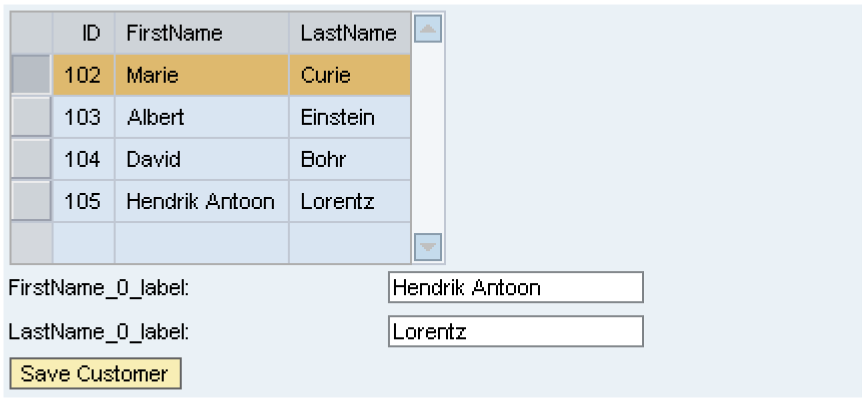
The relevant *Outline* view.



Deploy new Archive and Run

1. Right-click your application **MyApp** in the **mywd** DC and choose **Deploy new Archive and Run** from the context menu.
2. Have Fun!

Result



ID	FirstName	LastName
102	Marie	Curie
103	Albert	Einstein
104	David	Bohr
105	Hendrik Antoon	Lorentz

FirstName_0_label:	Hendrik Antoon
LastName_0_label:	Lorentz

Pressing the *Save Customer* Action Button of the form causes the framework to persist the given data. The table shows the new record instantly.

Related Content

Adobe PDF

Creating Your First Web Dynpro Application

Basics of JPA - Defining and Using Relationships

Basics of JPA - Understanding the Entity Manager

SAP Library

Component Model

Developing Java EE 5 Applications

Using JPA in the Persistence Layer

Developing User Interfaces with Web Dynpro for Java

Copyright