Using a Custom EJB Function as a Mapping Function in SAP NetWeaver Business Process Management

Applies to:

SAP enhancement package 1 for SAP NetWeaver CE 7.1

For more information, visit the Business Process Modeling homepage.

Summary

This document describes how to create an external function, expose it as EJB and use it as a mapping function in SAP NetWeaver Business Process Management. Some example functions are also provided.

Company: SAP AG

Created on: 15 December 2008

Table of Contents

How to create an external function (user defined) and expose it as EJB and use it as a BPM mapping function 3
- Create a DC for EJB 3
- Create a DC for EAR 3
- Create Dependencies for the New DCs 3
- Create the EJB 4
- Implement the Mapping Function in the EJB 4
- Build and Deploy 4
- Use the Newly Implemented Mapping Function in the Business Process 4
- Example Concat Function 5
- Example Filter Function 15
- Example Complex Types 17

Copyright 19

How to create an external function (user defined) and expose it as EJB and use it as a BPM mapping function

This document describes how to extend standard BPM modeling and mapping functionalities with custom Java code. This way you will be able to introduce new functionality which is not provided in the standard BPM modeling environment.

In order to do that, you need three development components (DC):

- an EJB DC

- an EAR DC

- the existing process composer DC

Create a DC for EJB

In the SAP NetWeaver Developer Studio, open the *Development Infrastructure* perspective and select a software component (SC). Within this SC, create a new DC with type *J2EE -> EJB Module*. Keep the default settings.

Create a DC for EAR

In the Developer Studio, open, the *Development Infrastructure* perspective and select the same SC as for the EJB DC. Within this SC create a new DC with type *J2EE -> Enterprise Application*. Keep the default settings.

Create Dependencies for the New DCs

Open the *Development Component* perspective and open the *Component Properties* view for

- **The new EAR DC:** Open the *Dependencies* tab, select *Add* and add
  - the created EJB DC (Select only the *Build Time* reference to *ejbjar*)
  - the DC *tc/bpem/mapping/facade* in the SC BPEM-FACADE:
    - *tc/bpem/mapping/façade: Deploy Time*
    - *tc/bpem/mapping/façade: Runtime*
    - *tc/bpem/mapping/façade api: Build Time*
  - the DC *tc/je/sdo21/api* in the SC ENGFACADE:
    - *tc/je/sdo21/api: Deploy Time*
    - *tc/je/sdo21/api: Runtime*
    - *tc/je/sdo21/api api: Build Time*

- **The new EJB DC:** Open the *Dependencies* tab, select *Add* and add
  - the DC *tc/bpem/mapping/facade* in the SC BPEM-FACADE:
    - *tc/bpem/mapping/façade: Deploy Time*
    - *tc/bpem/mapping/façade: Runtime*
    - *tc/bpem/mapping/façade api: Build Time*
  - the DC *tc/je/sdo21/api* in the SC ENGFACADE:
    - *tc/je/sdo21/api: Deploy Time*
    - *tc/je/sdo21/api: Runtime*
    - *tc/je/sdo21/api api: Build Time*

Create the EJB

Open the *JavaEE* perspective and open the created EJB project. Open the context menu of *ejbModule* and choose *NEW -> EJB Session Bean 3.0*. Create a stateless container managed session bean and include a local business interface.

Implement the Mapping Function in the EJB

Let the local interface of your EJB implement the following two interfaces:

```
com.sap.glx.mapping.execution.api.function.Function
com.sap.glx.mapping.execution.api.invoker.SdoInvoker
```

Now you can implement the needed mapping function method `invokeSdo` in the bean implementation. The data is passed as service data object (SDO) into this method and the result is also returned in this format.

To access the individual data elements within the SDO the lookup name has to be calculated by the provided helper functions:

```
SdoRenamingHelper.renameXsdElementToSdoProperty
SdoRenamingHelper.renameXsdAttributeToSdoProperty
```

Build and Deploy

Open the *Development Infrastructure* perspective, open the context menu for the created EJB DC and choose *Build*. Do the same for the created EAR DC and deploy the EAR to the SAP AS Java afterwards.

Use the Newly Implemented Mapping Function in the Business Process

Open your business process in the *Process Composer* perspective and open the context menu on *Process Modeling -> Rules and Functions*. Create a new *EJB Function* and choose the parameter names according to your implemented EJB function. After creating the function, you have to enter the JNDI lookup name for the EJB.

To do that, open the NetWeaver Administrator in a web browser and navigate to *Problem Management -> Java -> JNDI Browser* (or use the quick link http://<host>:<port>/nwa/jndi). Find the local business interface of your EJB. You can do this by searching for the *Context Name* (the DC name is part of the context name) or by searching for *Object Name* the name of the local business interface.

Select the local business interface of the EJB and copy the content of the field *Object Name* to the *JNDI Name* field of the EJB function you created in the Developer Studio.

Alternatively, type the JNDI lookup name directly in the *JNDI Name* field.The JNDI lookup name is constructed in the following way:

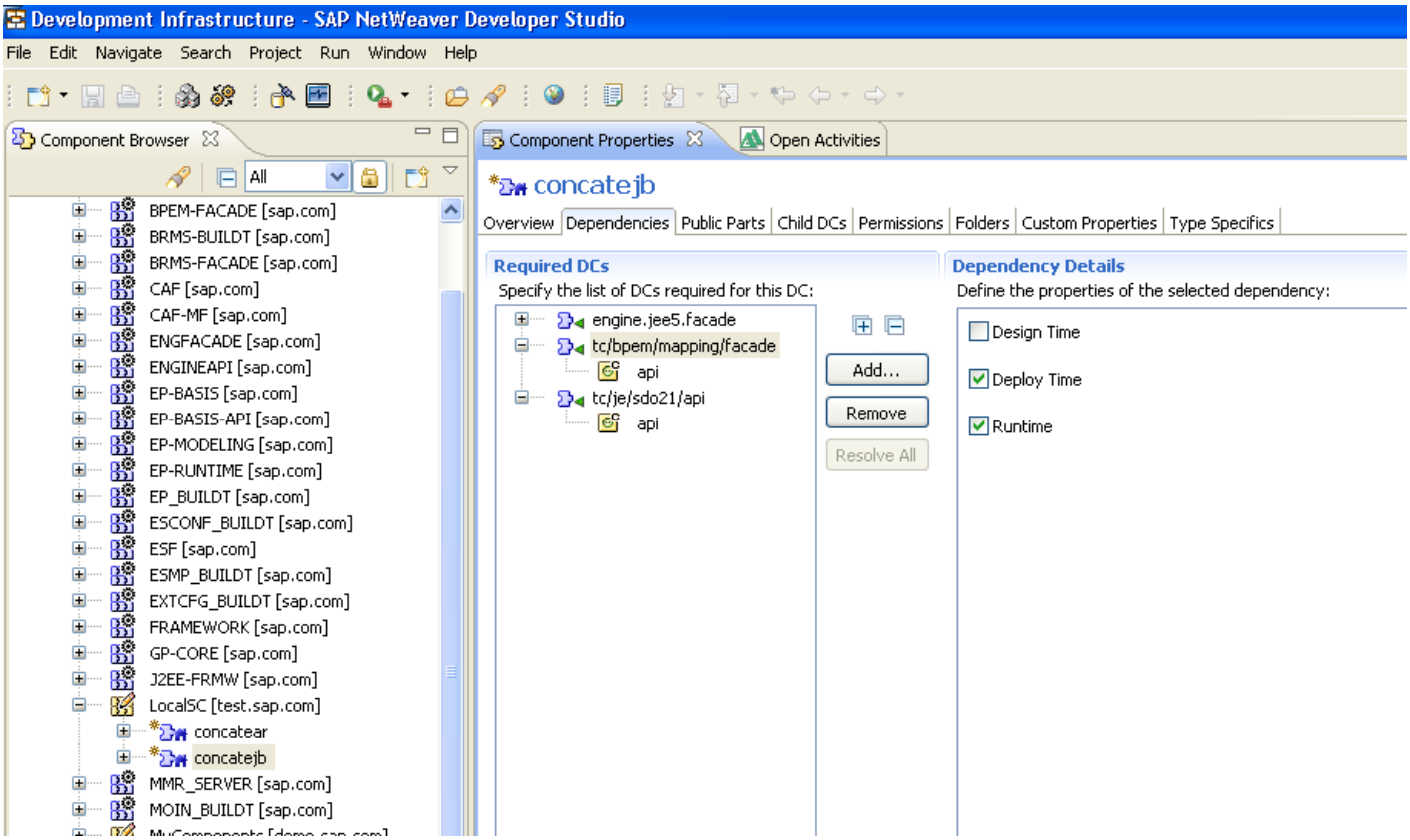
<SC-Vendor>***/***<EAR DC-Name>***/LOCAL/***<Simple EJB Class Name>***/***<Full Qualified Local Interface Name>

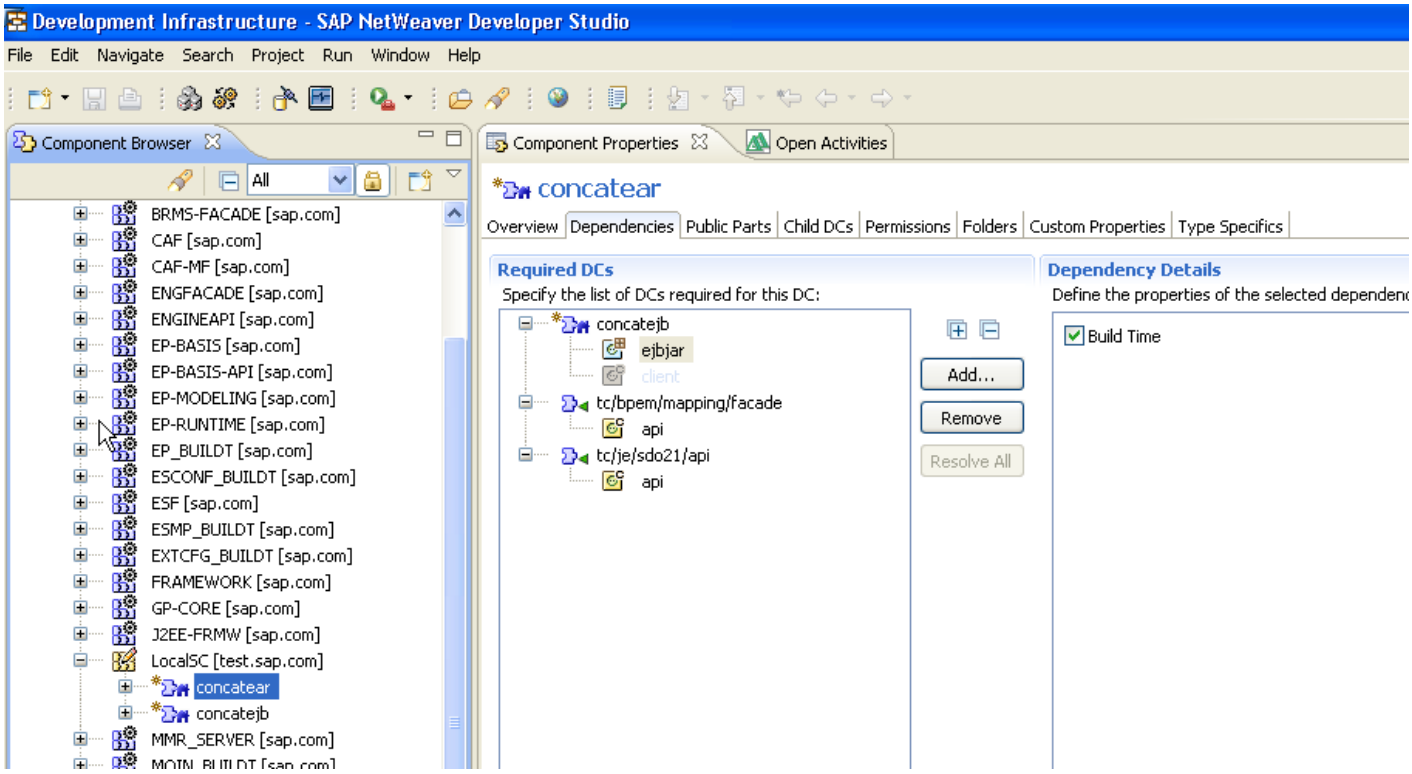
Hint: a / is substituted by a ~ in the <EAR DC-Name>

Example Concat Function

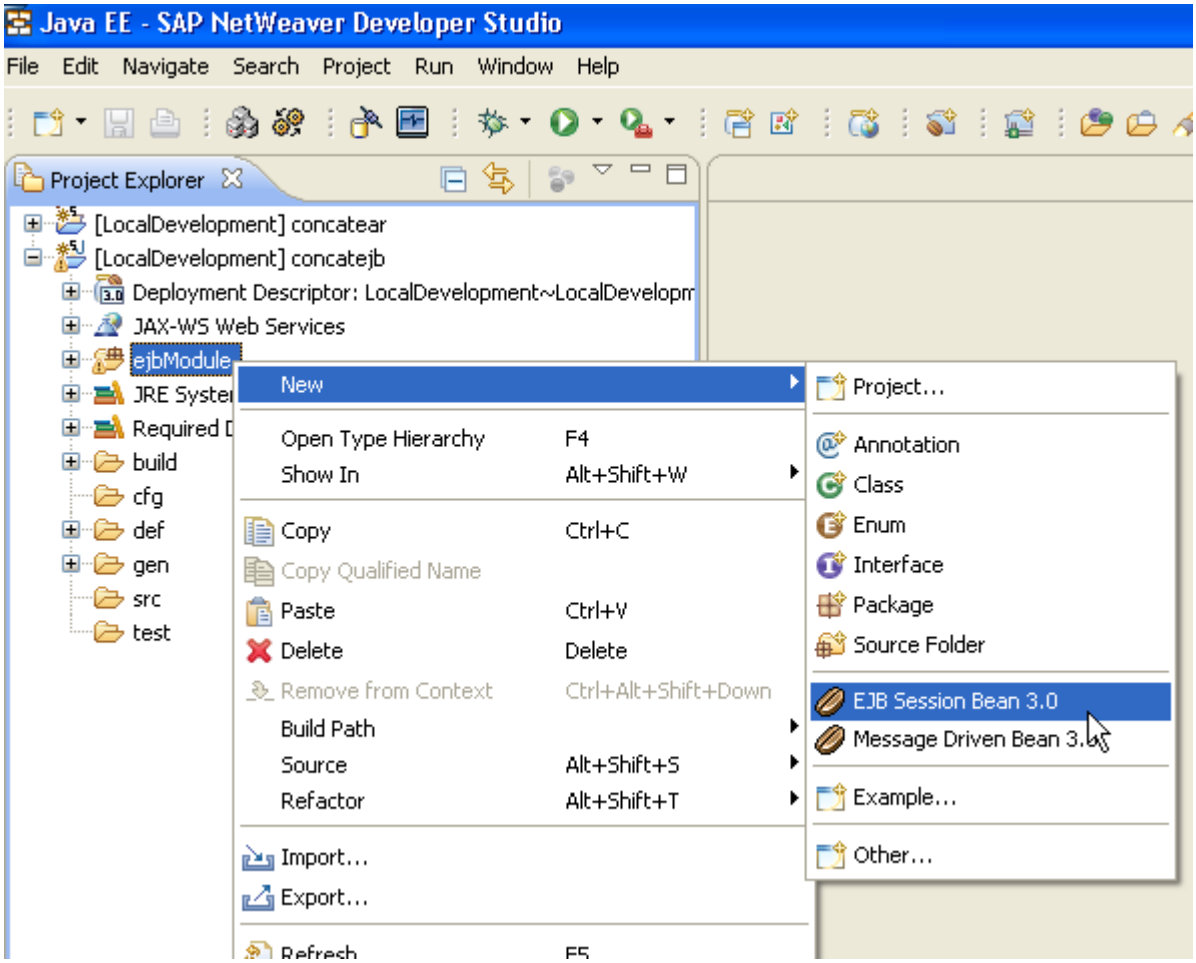
This example describes how to create an EJB function that will concatenate two strings.

First, create the EJB DC,and name it **concatejb**. Then create the EAR DC, and name it **concatear**. After that, set the necessary dependencies as described above for both DCs respectively.

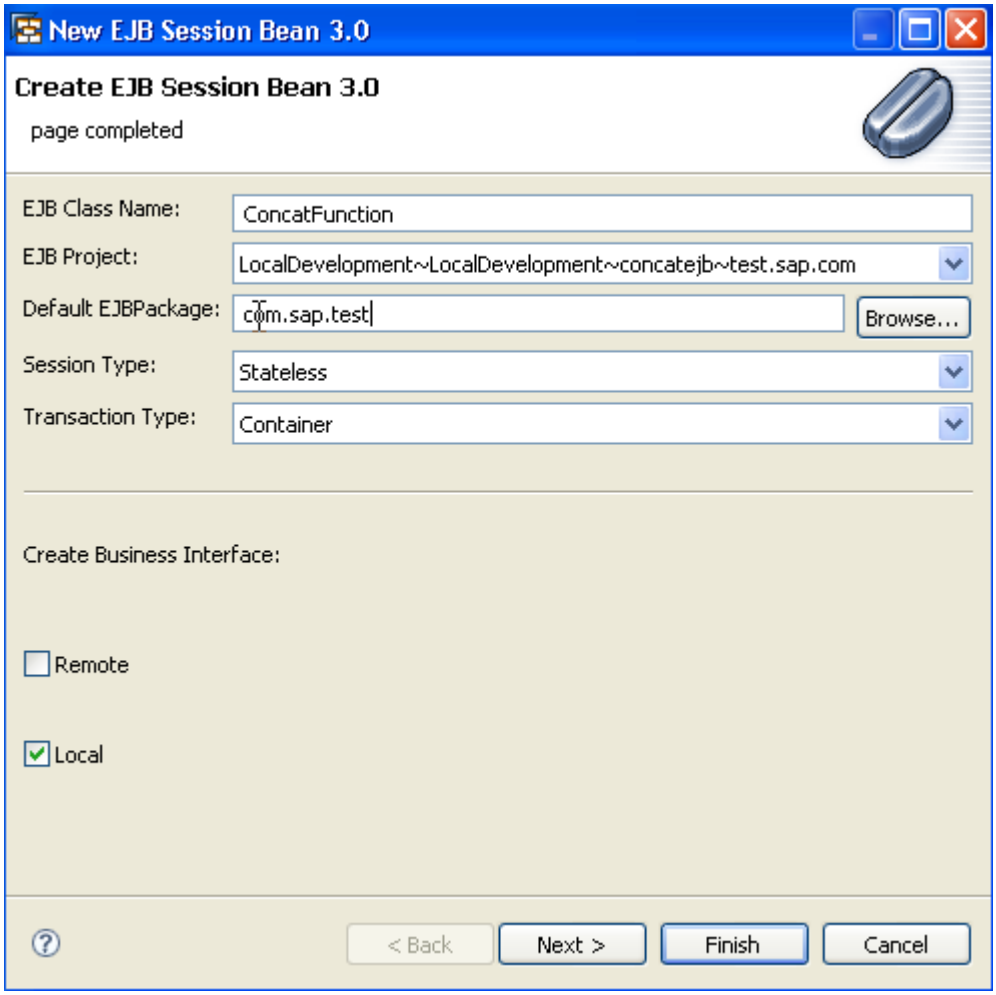



Development Infrastructure - SAP NetWeaver Developer Studio
File Edit Navigate Search Project Run Window Help
Component Browser
All
BRMS-FACADE [sap.com]
CAF [sap.com]
CAF-MF [sap.com]
ENGFACADE [sap.com]
ENGINEAPI [sap.com]
EP-BASIS [sap.com]
EP-BASIS-API [sap.com]
EP-MODELING [sap.com]
EP-RUNTIME [sap.com]
EP_BUILDT [sap.com]
ESCONF_BUILDT [sap.com]
ESF [sap.com]
ESMP_BUILDT [sap.com]
EXTCFG_BUILDT [sap.com]
FRAMEWORK [sap.com]
GP-CORE [sap.com]
J2EE-FRMW [sap.com]
LocalSC [test.sap.com]
concatear
concatejb
MMR_SERVER [sap.com]
Component Properties
Open Activities
concatear
Overview Dependencies Public Parts Child DCs Permissions Folders Custom Properties Type Specifics
Required DCs
Specify the list of DCs required for this DC:
concatejb
ejbjar
client
tc/bpem/mapping/facade
api
tc/je/sdo21/api
api
Add...
Remove
Resolve All
Dependency Details
Define the properties of the selected dependenc
Build Time

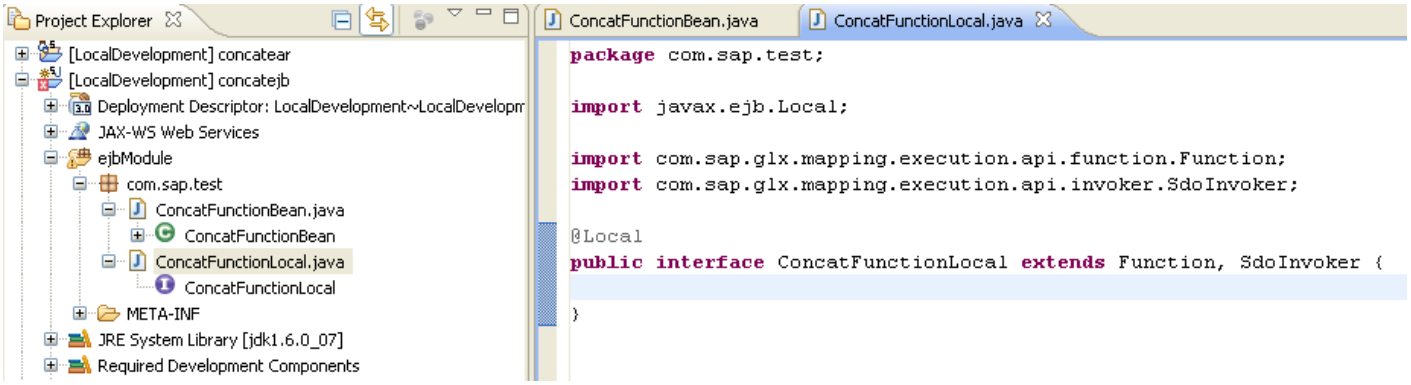
In the *JavaEE* perspective, create the new stateless session bean.



Name it **ConcatFunction** in the package *com.sap.test* and choose *Finish*.



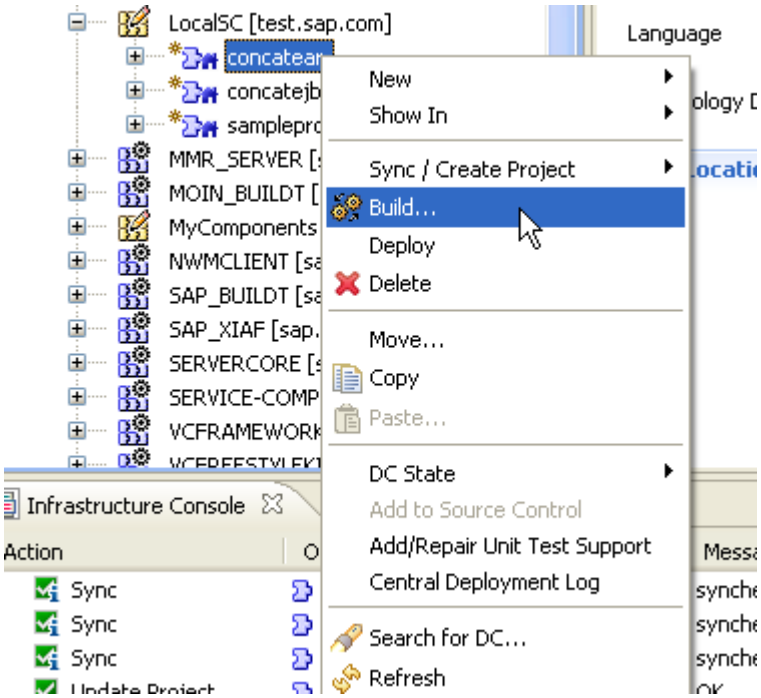
Open the local business interface (*ConcatFunctionLocal*) of your EJB and let it extend the interfaces *Function* and *SdoInvoker*

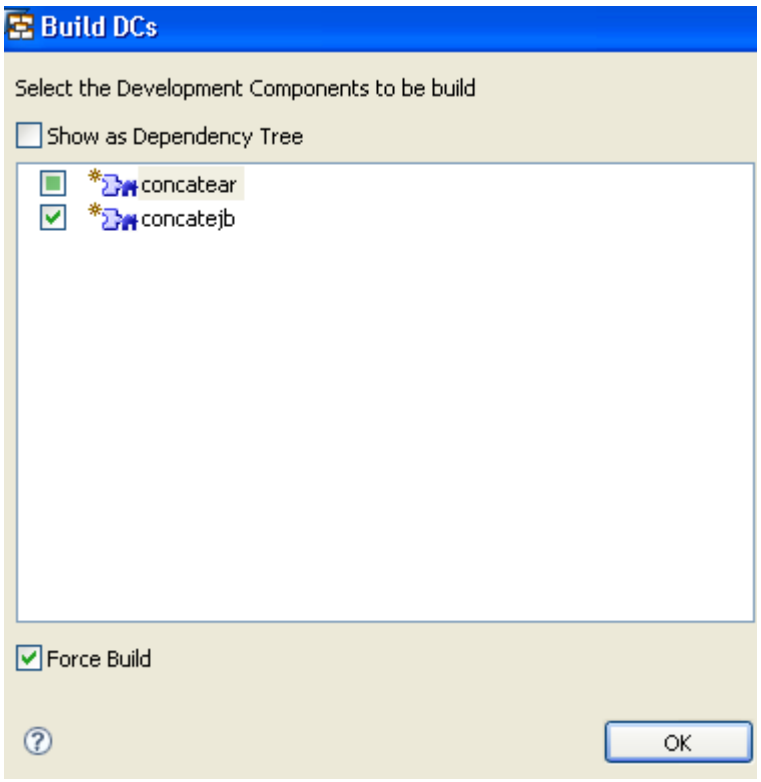


The implementation of the *ConcatFunctionBean* can look like this:

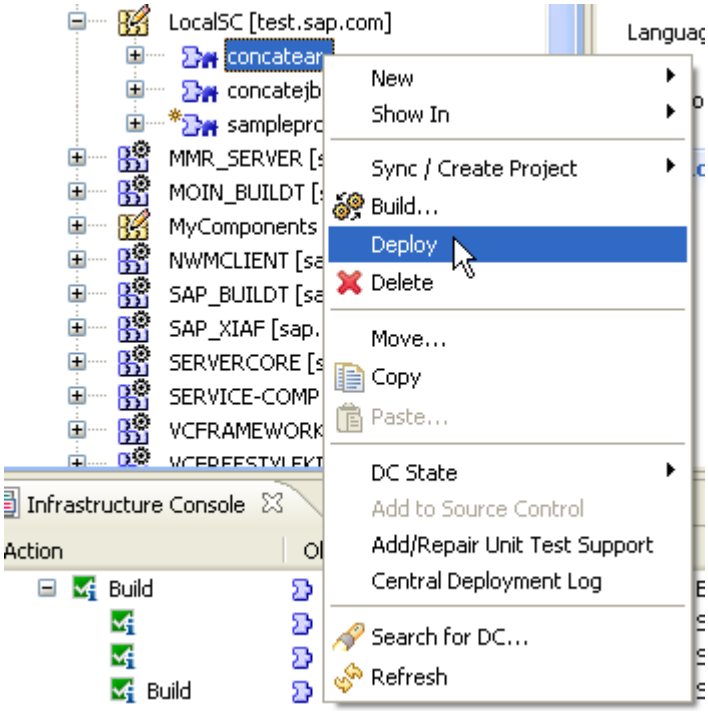
```
package com.sap.test;

import javax.ejb.Stateless;
import javax.xml.namespace.QName;

import com.sap.glx.sdo.api.SdoRenamingHelper;
import commonj.sdo.DataObject;
import commonj.sdo.Type;

@Stateless
public class ConcatFunctionBean implements ConcatFunctionLocal {

   private static final String NAMESPACE_FUNCTION_PARAMETER =
"test.sap.com/sampleprocess";
   private static final String PARAMETER1 = "parameter1";
   private static final String PARAMETER2 = "parameter2";
   private static final String RESULT = "result";

   private static final String NAME_PROPERTY_INPUT_PARAMETER1 = SdoRenamingHelper
               .renameXsdElementToSdoProperty(new QName(
                           NAMESPACE_FUNCTION_PARAMETER, PARAMETER1), false);

   private static final String NAME_PROPERTY_INPUT_PARAMETER2 = SdoRenamingHelper
               .renameXsdElementToSdoProperty(new QName(
                           NAMESPACE_FUNCTION_PARAMETER, PARAMETER2), false);

   private static final String NAME_PROPERTY_OUTPUT_RESULT = SdoRenamingHelper
               .renameXsdElementToSdoProperty(new QName(
                           NAMESPACE_FUNCTION_PARAMETER, RESULT), false);

   public DataObject invokeSdo(DataObject inputDO,
               InvocationContext invocationContext) {

         // input
         Type typeInput = inputDO.getType();
         String parameter1 = inputDO.getString(typeInput
                     .getProperty(NAME_PROPERTY_INPUT_PARAMETER1));
         String parameter2 = inputDO.getString(typeInput
                     .getProperty(NAME_PROPERTY_INPUT_PARAMETER2));

         // function code
         String result = parameter1.concat(parameter2);

         // output
         DataObject outputDO = invocationContext.createOutputDataObject();
         outputDO.setString(outputDO.getType().getProperty(
                     NAME_PROPERTY_OUTPUT_RESULT), result);

         return outputDO;
   }
}
```

Build the created EJB and the EAR:



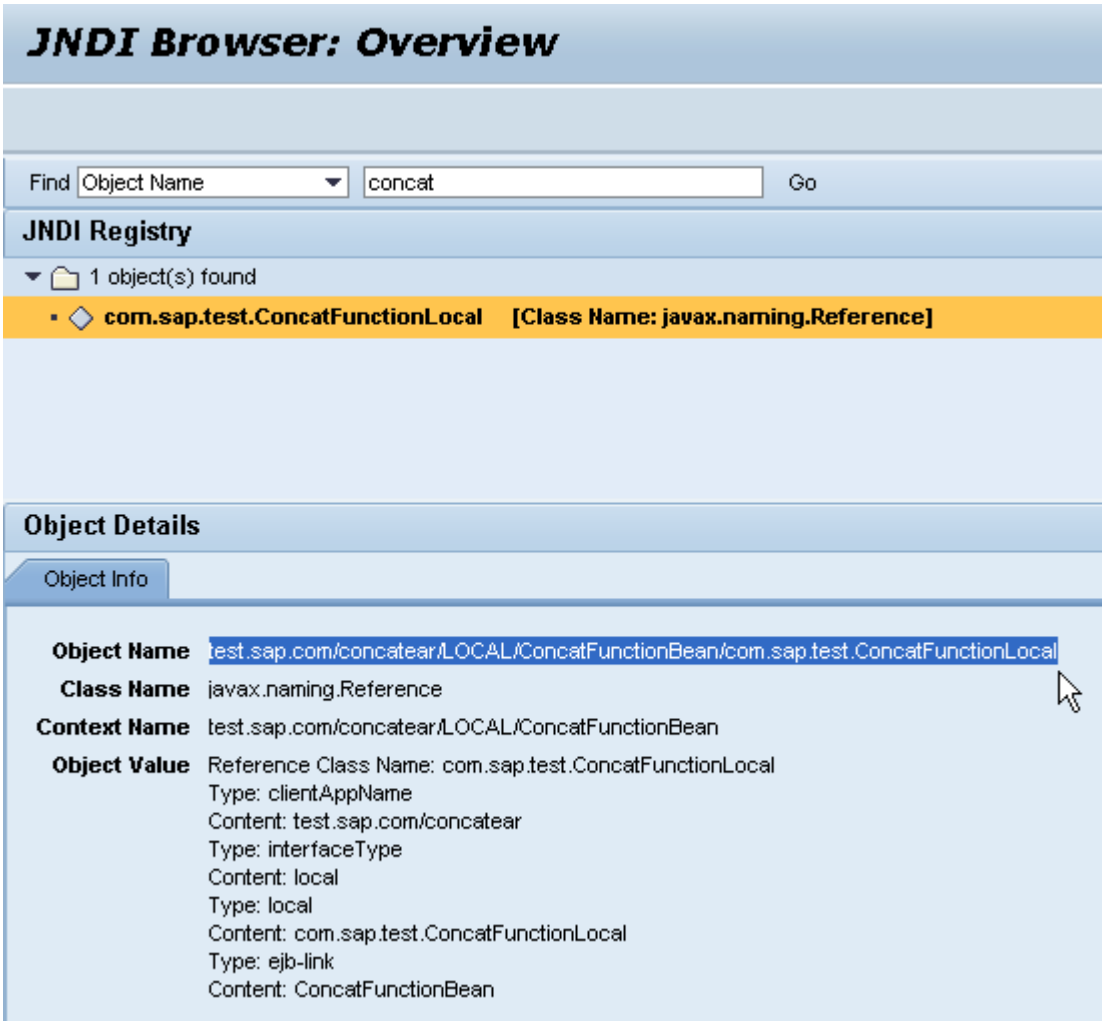


Deploy the EAR DC, which contains the EJB:



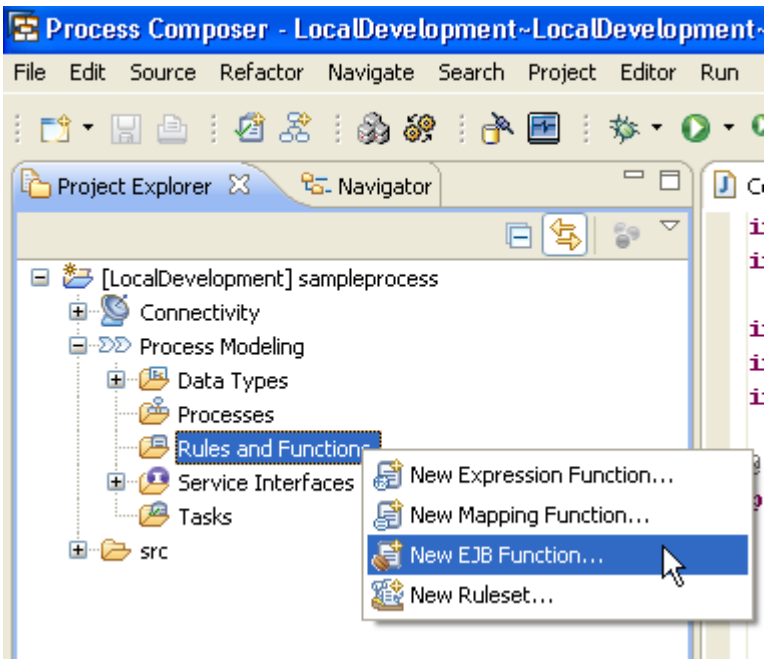
Before you are able to deploy your DC, SAP AS Java, which runs the BPM Process Server, must be configured in the Developer Studio. To configure the SAP AS Java, choose *Window -> Preferences -> SAP AS Java*.

Find out the JNDI lookup name for your EJB via opening the NetWeaver Administrator in a web browser pointing to http://<host>:<port>/nwa/jndi

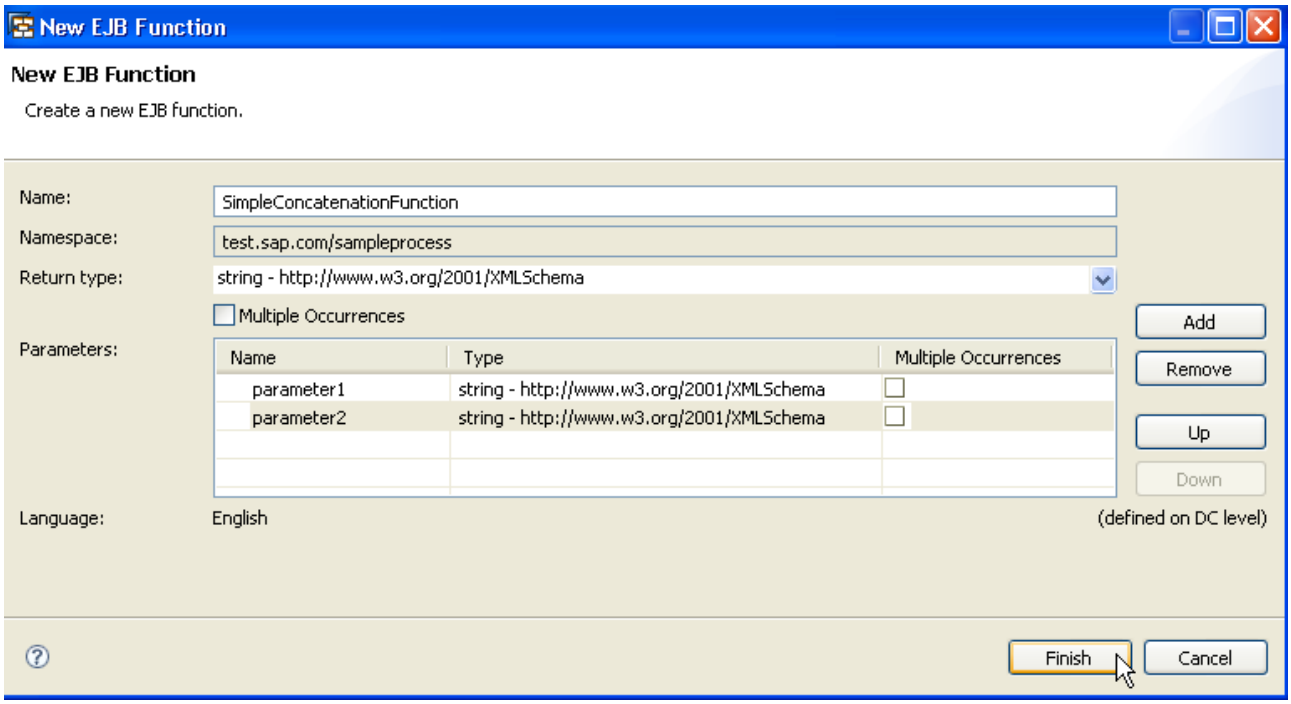


Alternatively, you can construct the JNDI lookup name by the formula described above in this document.

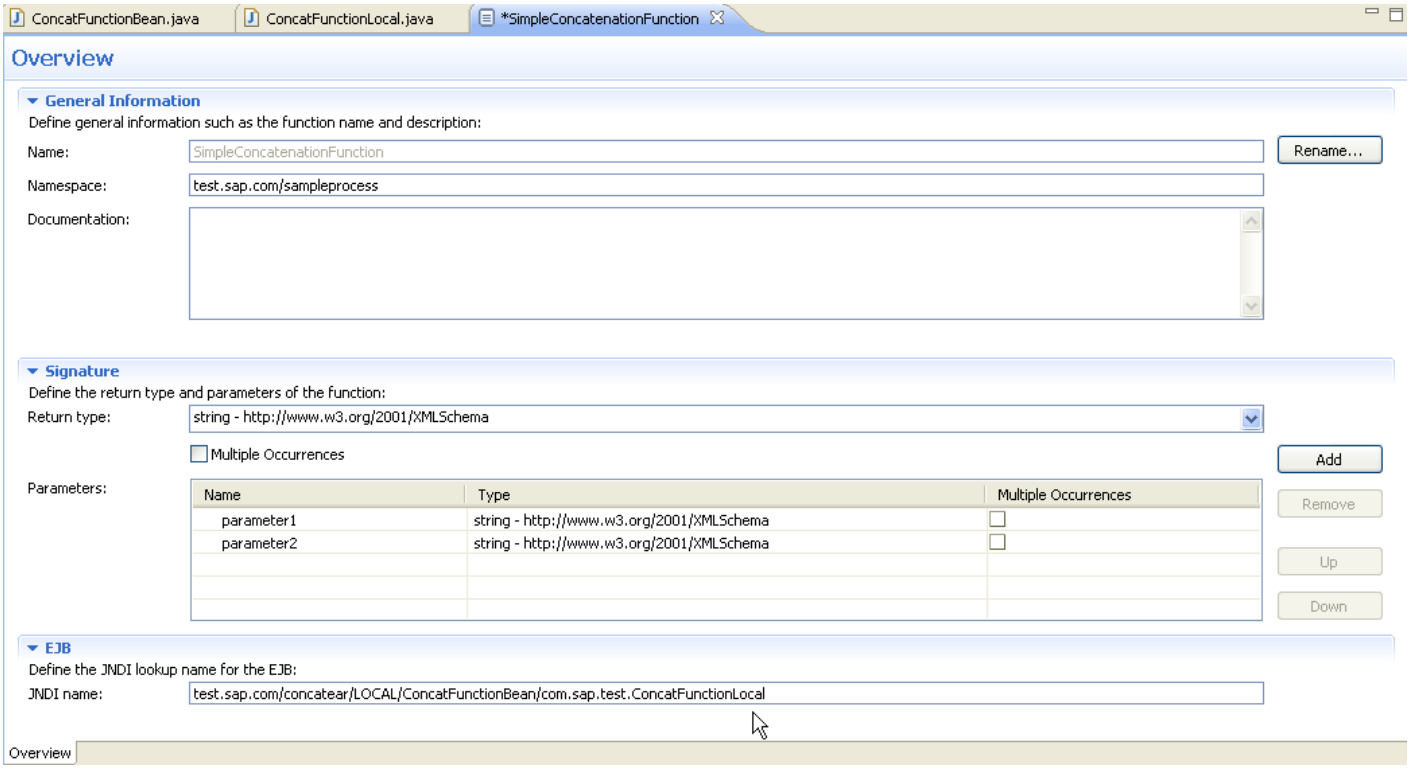
Open the *Process Composer* perspective and create the new EJB function to be used by your business process:



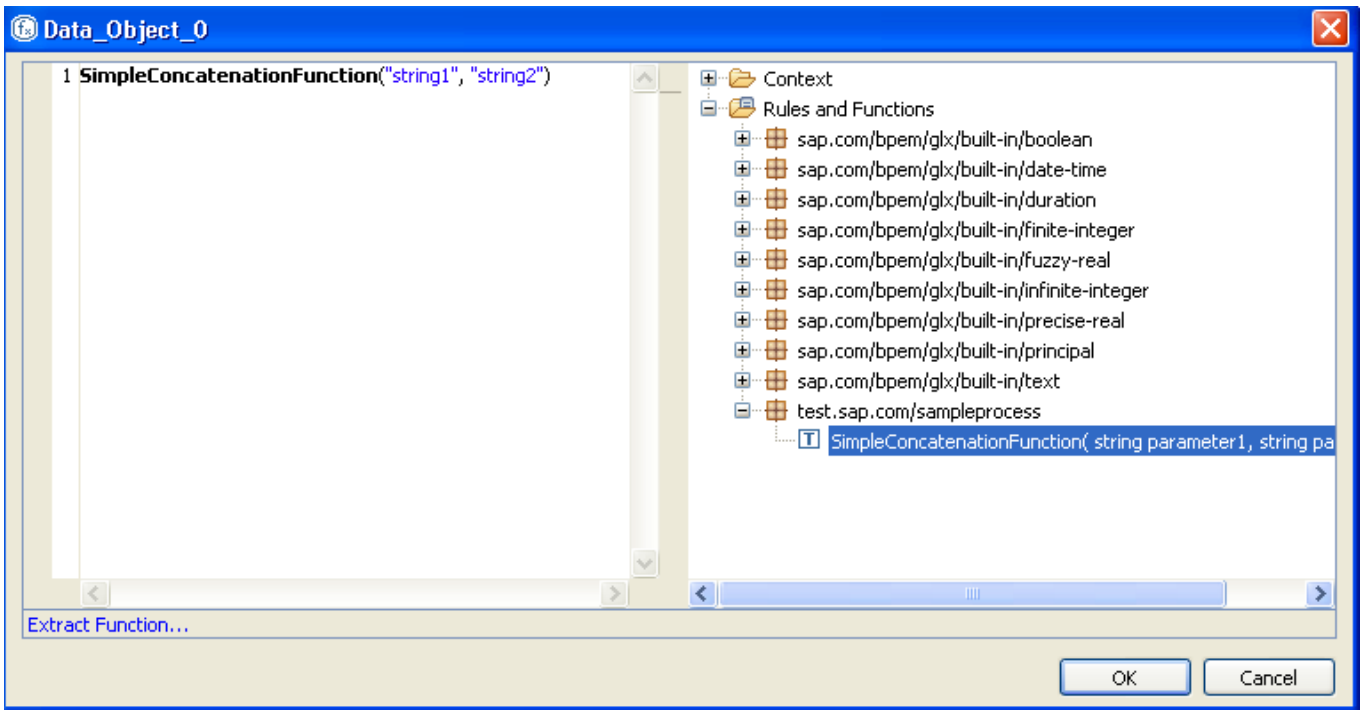
The names and types of the function parameters have to match the ones you defined in the Java implementation of your EJB.



Enter the above found JNDI lookup name in the *JNDI name* field of the EJB function. You can see the *JNDI name* field when you open the EJB function in the function editor.



From now on you are able to use this function within a mapping



Example Filter Function

This is another example showing how to iterate through list and how to work with complex types.

This function receives a list of Principals, filters them and returns the filtered list of Principals.

Principal type has the following structure:

Principal

principalId

type

To filter this list, it is necessary to give the function these arguments: list, field by which list will be filtered(in this case principalId or type) and value for this field.

The implementation of the *example filter function* can look like this:

```
package com.sap.test;

import java.util.Iterator;
import java.util.List;

import javax.ejb.Stateless;
import javax.xml.namespace.QName;

import com.sap.glx.mapping.execution.api.function.Function;
import com.sap.glx.mapping.execution.api.invoker.SdoInvoker;
import com.sap.glx.sdo.api.SdoRenamingHelper;
import commonj.sdo.DataObject;
import commonj.sdo.Property;
import commonj.sdo.Type;

public class FilterFunctionBean implements FilterFunctionLocal {

   private static final String NAMESPACE_FUNCTION = "test.sap.com/sampleprocess";
   private static final String NAME_PROPERTY_INPUT_ARG1 =
SdoRenamingHelper.renameXsdElementToSdoProperty( new QName(NAMESPACE_FUNCTION,
"arg1"), false);
   private static final String NAME_PROPERTY_INPUT_ARG2 =
SdoRenamingHelper.renameXsdElementToSdoProperty( new QName(NAMESPACE_FUNCTION,
"arg2"), false);
   private static final String NAME_PROPERTY_INPUT_ARG3 =
SdoRenamingHelper.renameXsdElementToSdoProperty( new QName(NAMESPACE_FUNCTION,
"arg3"), false);
   private static final String NAME_PROPERTY_OUTPUT_RESULT =
SdoRenamingHelper.renameXsdElementToSdoProperty( new QName(NAMESPACE_FUNCTION,
"result"), false);

   public DataObject invokeSdo(DataObject input, InvocationContext invocationContext)
{
         // check sourceDO
         if (input == null) {
               throw new IllegalArgumentException("SourceDO must exist");
         }
```

```
        if (invocationContext == null) {
              throw new IllegalArgumentException("HelperContext must exist");
        }

        //input
        Type typeInput = input.getType();

        List<DataObject> listPrincipal =
input.getList(typeInput.getProperty(NAME_PROPERTY_INPUT_ARG1));
        String filterCriteria =
input.getString(typeInput.getProperty(NAME_PROPERTY_INPUT_ARG2));
        String filterValue =
input.getString(typeInput.getProperty(NAME_PROPERTY_INPUT_ARG3));

        //Way to check input argument is attribute or element
        if(filterCriteria.startsWith("@")){
              filterCriteria = SdoRenamingHelper.renameXsdAttributeToSdoProperty(
new QName(null, filterCriteria), false);
        } else {
              filterCriteria = SdoRenamingHelper.renameXsdElementToSdoProperty( new
QName(null, filterCriteria), false);
        }

        // function code
        for (Iterator<?> it = listPrincipal.iterator(); it.hasNext(); ) {
              DataObject principalDO = (DataObject) it.next();
              String principalField = principalDO.getString(
principalDO.getType().getProperty(filterCriteria) );
              if ( filterValue.compareTo( principalField ) !=0 ){
             it.remove();
              }
        }

        // output
        DataObject outputDO = invocationContext.createOutputDataObject();
        Property outputProperty =
outputDO.getType().getProperty(NAME_PROPERTY_OUTPUT_RESULT);
        outputDO.setList(outputProperty, listPrincipal);

        return outputDO;
   }


}
```

Example Complex Types

This is an example how to create complex types.

The return type is Principal.

```
package com.sap.test;

import javax.ejb.Stateless;
import javax.xml.namespace.QName;

import com.sap.glx.mapping.execution.api.function.Function;
import com.sap.glx.mapping.execution.api.invoker.SdoInvoker;
import com.sap.glx.sdo.api.SdoRenamingHelper;
import commonj.sdo.DataObject;
import commonj.sdo.Property;
import commonj.sdo.Type;

public class CreatePrincipalBean implements CreatePrincipalLocal {

	private static final String NAMESPACE_FUNCTION = "test.sap.com/sampleprocess";

	private static final String NAME_PROPERTY_INPUT_ARG1 =
SdoRenamingHelper.renameXsdElementToSdoProperty( new QName(NAMESPACE_FUNCTION, "arg1"),
false);
	private static final String NAME_PROPERTY_INPUT_ARG2 =
SdoRenamingHelper.renameXsdElementToSdoProperty( new QName(NAMESPACE_FUNCTION, "arg2"),
false);
	private static final String NAME_PROPERTY_OUTPUT_RESULT =
SdoRenamingHelper.renameXsdElementToSdoProperty( new QName(NAMESPACE_FUNCTION, "result"),
false);

	public DataObject invokeSdo(DataObject input, InvocationContext
invocationContext) {

	  // check sourceDO
	  if (input == null) {
		throw new IllegalArgumentException("SourceDO must exist");
	  }

	  if (invocationContext == null) {
		throw new IllegalArgumentException("HelperContext must exist");
	  }

	  //input
	  Type typeInput = input.getType();

	  String idValue =
input.getString(typeInput.getProperty(NAME_PROPERTY_INPUT_ARG1));
	  int typeValue =
input.getInt(typeInput.getProperty(NAME_PROPERTY_INPUT_ARG2));

	  // function code
	  DataObject rootObject = invocationContext.createOutputDataObject();
	  Property resultProperty =
rootObject.getType().getProperty(NAME_PROPERTY_OUTPUT_RESULT);
	  DataObject resultDO = rootObject.createDataObject( resultProperty );
	  String principalIdName = SdoRenamingHelper.renameXsdElementToSdoProperty(
new QName(null, "principalId"), false);
	  String typeName = SdoRenamingHelper.renameXsdElementToSdoProperty( new
QName(null, "type"), false);
```

```
        Property idProperty = resultDO.getType().getProperty(principalIdName);
        Property typeProperty = resultDO.getType().getProperty(typeName);
        resultDO.set(idProperty, idValue);
        resultDO.set(typeProperty, typeValue);


        return rootObject;


    }

}
```

Copyright