Creating and using Custom Data Types in SAP NetWeaver BPM 7.2

Applies to:

SAP NetWeaver Composition Environment 7.2 and above. For more information, visit the Business Process Modeling homepage.

Summary

For building a highly sophisticated BPMN process model, often standard data types and SAP provided pre-built functions may not be sufficient and would require the use of custom data types and functions. To develop custom functions on top of the custom data types, Service Data Objects (SDOs) needs to be used. The articles listed under the reference section of this document are few of the documents available in SDN that touches upon the use of SDO, but with the standard data types. So, this article is aimed at filling the gap that exists in this topic by providing details on the creation and use of **custom data types** in a BPMN process model and on **leveraging SDOs** to build **custom functions (**EJB mapping functions) that uses the custom data types.

Author: Balakrishnan.B

Company: Incture Technologies

Created on: 18 April 2011

Author Bio

Balakrishnan is a BPM Architect involved in building BPM solutions using the key SAP NetWeaver components (BPM, BRMS, CAF – Core & GP, EP, WebDynpro, VC, MDM) and open Java EE 5.0 standards (EJB 3.0, JAX – WS 2.0, JPA 1.0). His current areas of work include scalability and performance optimization.

Table of Contents

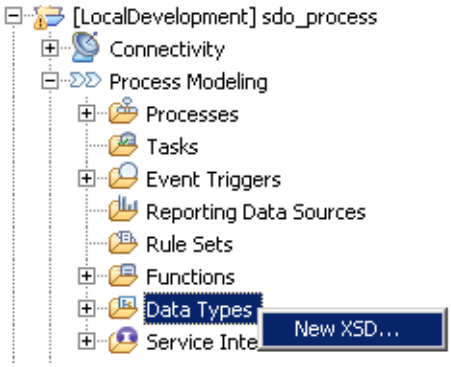
The Case

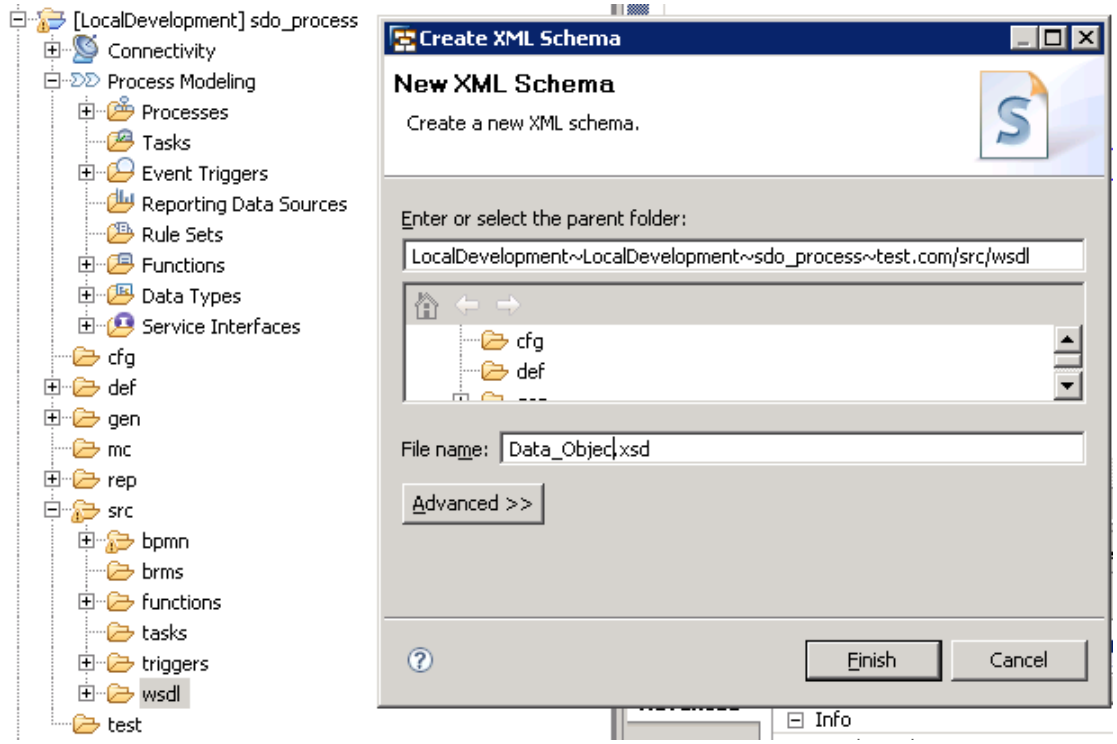
Typically BPM cuts across different business facets and as a result, the underlying BPMN process models also carry with it the need for handling scenario specific business objects. This effectively means that the process models should also contain a more relevant representation of the business object in the process context through out its lifecycle. To do that, SAP NetWeaver BPM provides features to create custom data types. Not just that. SAP NetWeaver BPM also provides APIs to manipulate the data contained within the data objects mapped with custom data types. This article provides a step-by-step approach for creating and using custom data types and manipulating the data objects (*with the help of SDOs*) to produce a streamlined data transfer between the BPMN process models and the underlying computational logic.

Note: It is assumed that the reader is conversant with NWDS and creating BPMN process models. So the basic steps involved in creating process models are not covered in this article.

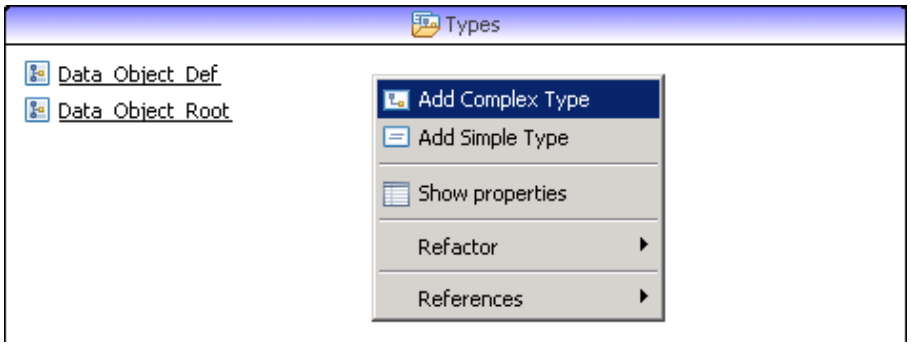
Creating Custom Data Types in a BPMN Process Model

Assuming that there is a need to have a data object in the process context which should be capable of holding multiple objects / instances of an element having several attributes, let's create a new Process Composer Development Component (DC) and navigate to the node "Data Types" under it. Use the right click context menu to choose "New XSD". In the pop-up that comes up, key-in the name of the custom data type that you want to create, say *Data_Object* and press finish.

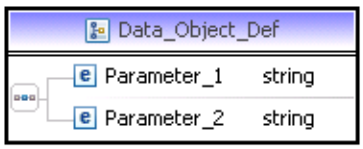




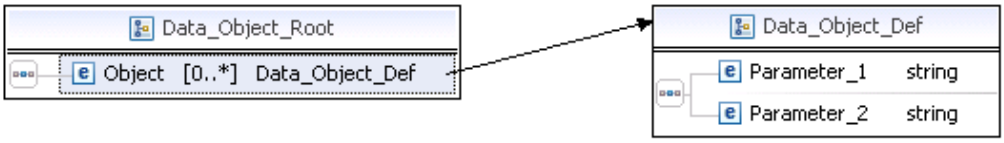
The newly created data type would open up in a separate editor. In the “Types” section of the editor, right click to choose “Add Complex Type” from the context menu and create two complex types *Data_Object_Def* and *Data_Object_Root*, for instance.

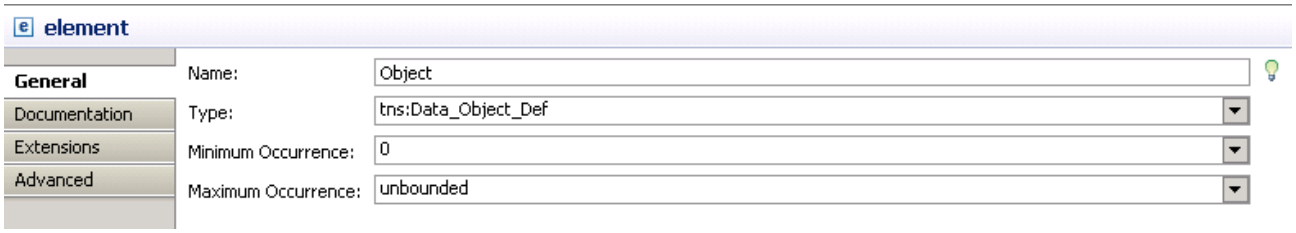


Double click to open the complex type “*Data_Object_Def*” and right click the type to choose the option “Add Element” to add two elements (*say Parameter_1* and *Parameter_2*) with “*string*” as the data type, as shown below.



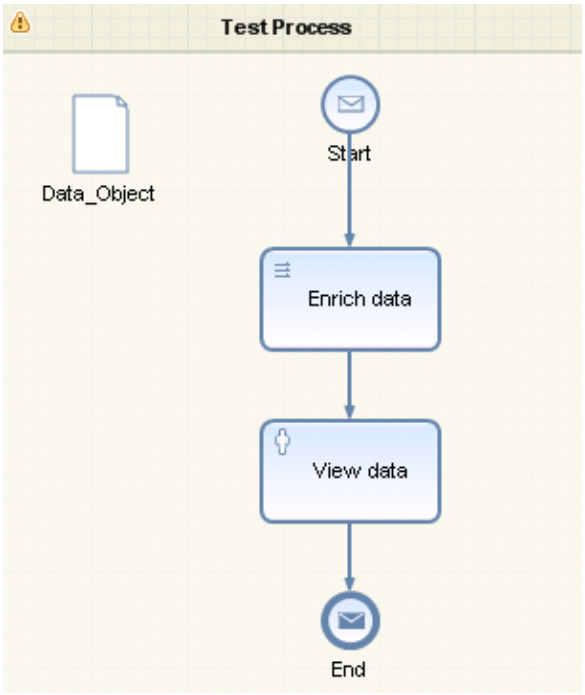
Similarly, define the complex type “*Data_Object_Root*” by double clicking and choosing “Add Element” to add an element with name “*Object*” and set its data type to “*Data_Object_Def*” and the minimum and maximum occurrences as “0” and “unbounded” respectively. With this, a data type with an unbounded structure containing two attributes has been created.





Note: You may need to use the option “Browse” to choose a reference to the data type “Data_Object_Def” as the type for the element “Object” under the complex type “Data_Object_Root”.

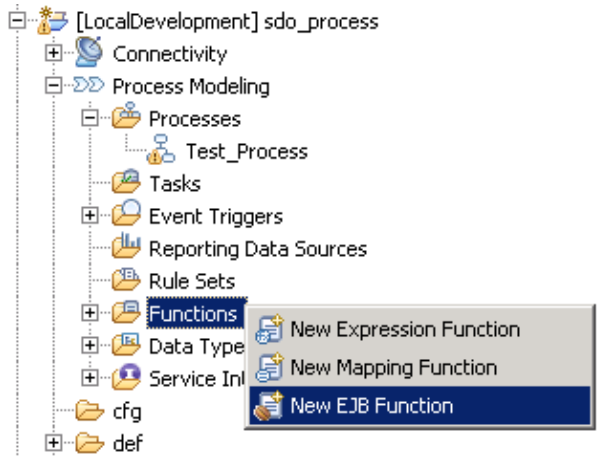
This data type (*Data_Object_Root*) can now be used in the process context by assigning it to a data object, say *Data_Object*. All said and done, now how to populate this data object with the relevant values in the process context during the runtime? The following section answers that. Let's consider the below process model to take the context forward.

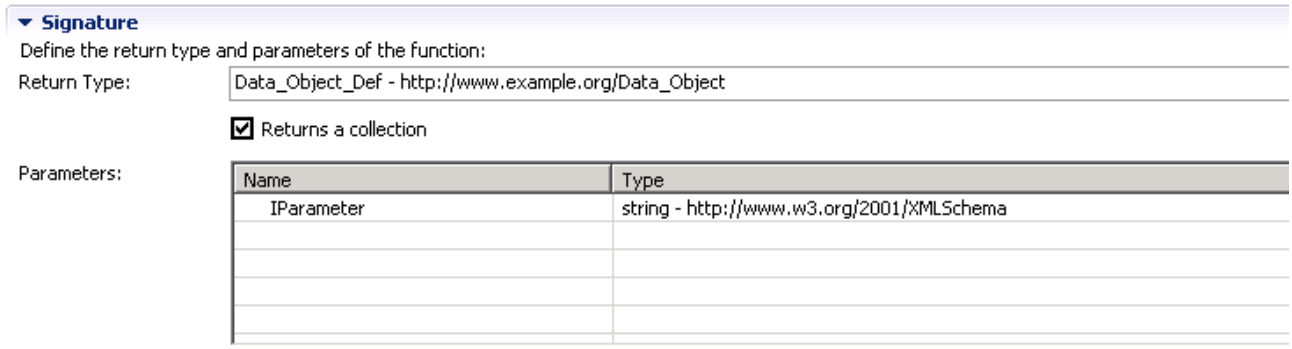


This process model contains a data object mapped to the data type "Data_Object" and contains a mapping function that calls the logic to fill the data object with the relevant values. Even though this model doesn't make any real business sense, it is good enough to demonstrate the approach.

Using Service Data Objects to update the Data Object

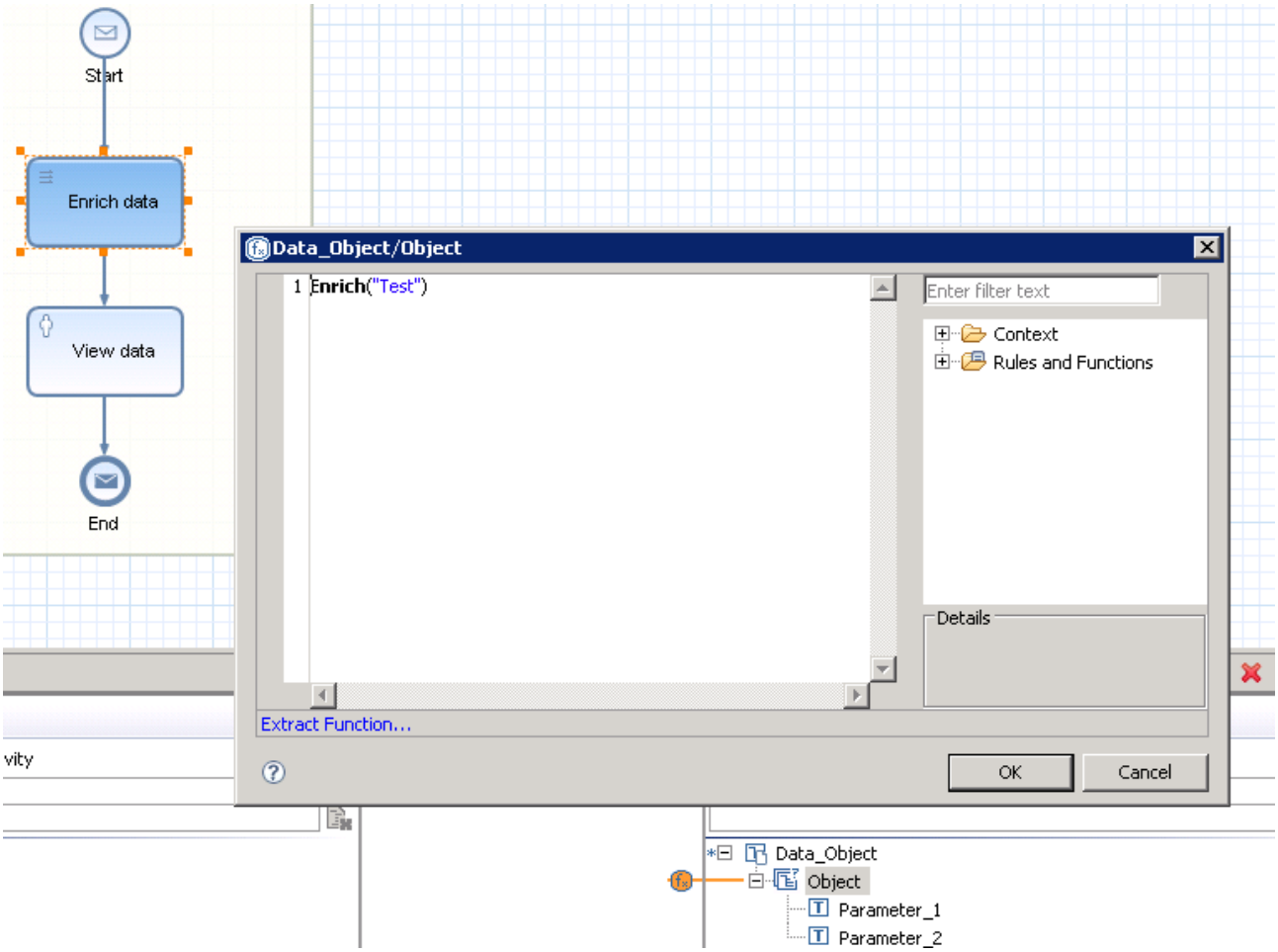
To update this "Data_Object", a custom EJB function is required. So, let's create an EJB mapping function, "Enrich" with the signature given below.





Hint: In order to enable the EJB mapping function to return back a collection of objects of type "Data_Object_Def", the return type should be marked with collection type as shown above and not with the type "Data_Object_Root" though the data object is mapped to this new data type Data_Object.

Then map this EJB function to the mapping activity as shown below.



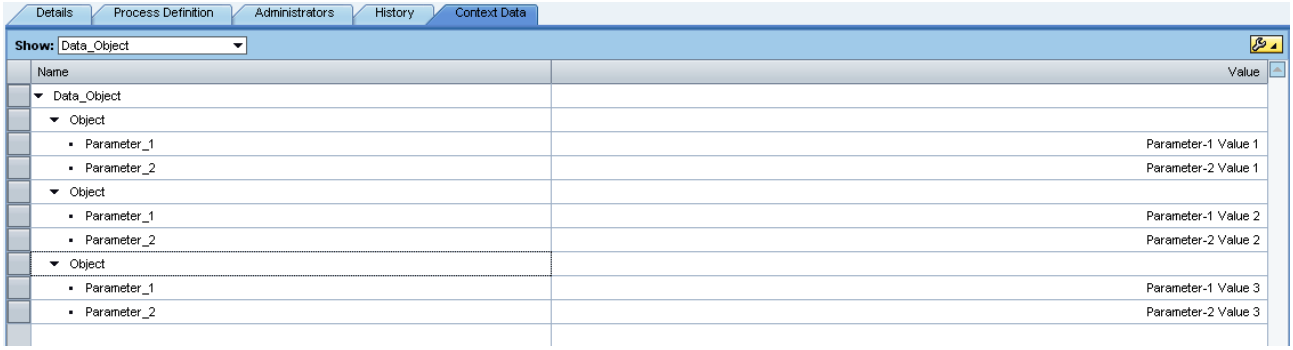
Further details on how to create and use EJB functions in a BPMN process model is available in this "How-to" document. Now, the key information that is required is "how to programmatically populate this object with values"? The details along with the relevant code piece are given below.

```
// BPM Process DC name space identifier
   private static final String PROCESS_NAMESPACE = "com.test/sdo_process";

   // EJB Function input parameter name space identifier
   private static final String INPUT_PARAMETER_ID = "IParameter";
   private static final String INPUT_PARAMETER = SdoRenamingHelper
                .renameXsdElementToSdoProperty(new QName(PROCESS_NAMESPACE,
                              INPUT_PARAMETER_ID), false);

   // Parameter names
   final static String PARAMETER1_IN_DATA_OBJECT =
"$http://www.example.org/Data_Object:Parameter_1";
   final static String PARAMETER2_IN_DATA_OBJECT =
"$http://www.example.org/Data_Object:Parameter_2";

   public SdoFunctionHelper() {
   }
   @Override
   public DataObject invokeSdo(DataObject inputDO,
                InvocationContext invocationContext) {

// Use this input parameter for any computational logic that may be
          // required. Here it is not being used.
Type typeInput = inputDO.getType();
          String invokerType = inputDO.getString(typeInput
                        .getProperty(INPUT_PARAMETER));

// Create an instance of the output data object.
          DataObject rootObject = invocationContext.createOutputDataObject();
          Property parameter1InDataObject = null;
          Property parameter2InDataObject = null;

          // Loop as you may need to populate the collection
          for (int i = 0; i < 3; i++) {
                // Create an instance of the object corresponding to the structure
                // Data_Object_Def
                DataObject resultDo = rootObject.createDataObject(0);


                // Your logic to populate the field values goes here…
                parameter1InDataObject =
resultDo.getType().getProperty(PARAMETER1_IN_DATA_OBJECT);
                parameter2InDataObject = resultDo.getType().getProperty(
                              PARAMETER2_IN_DATA_OBJECT);

                resultDo
                              .set(parameter1InDataObject, "Parameter-1 Value " + (i
+ 1));
                resultDo
                              .set(parameter2InDataObject, "Parameter-2 Value " + (i
+ 1));
          return rootObject;
```

Key Elements in the Code Piece

- The parameter name should be constructed in the following way.
  `$http://www.example.org/<Data Type Name>:<Parameter Name>.`
- Remember that in the signature of the EJB mapping function we have defined the return type as **collection of** Data_Object_Def. So, `DataObject rootObject = invocationContext.createOutputDataObject();` would create an instance of this object i.e List< Data_Object_Def >.
- `DataObject resultDo = rootObject.createDataObject(0);` would create an instance of this object Data_Object_Def.

With this in place, we are good to execute the process and see the collection getting populated into the data object of the process context. The result of this would look like the one shown below.



Related Content

Service Data Objects (SDO)

Service Data Objects preview application

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

For more information, visit the Business Process Modeling homepage.

Disclaimer and Liability Notice

This document may discuss sample coding or other information that does not include SAP official interfaces and therefore is not supported by SAP. Changes made based on this information are not supported and can be overwritten during an upgrade.

SAP will not be held liable for any damages caused by using or misusing the information, code or methods suggested in this document, and anyone using these methods does so at his/her own risk.

SAP offers no guarantees and assumes no responsibility or liability of any type with respect to the content of this technical article or code sample, including any liability resulting from incompatibility between the content within this document and the materials and services offered by SAP. You agree that you will not hold, or seek to hold, SAP responsible or liable with respect to the content of this document.