Show TOC

Extending an OData Service Using Service Builder

Activities

You can extend OData services by customizing the generated source code to suit your needs and purposes.

Using the redefine technique in Service Builder, you can extend a model by creating a new target project, and then define it as a redefine of the service you want to extend.

The redefine technique is executed in the Service Builder.

Provider Classes

When you redefine a service, you obtain a new OData service that includes a new Model Provider Class (MPC), and a new Data Provider Class (DPC).

The MPC includes the MPC of the source service, and the DPC inherits from the DPC of the source service.

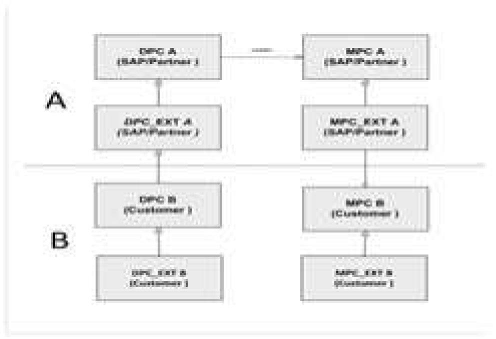


Illustration of Generated Provider Classes

In the illustration above, two sets of classes (MPC A, and DPC A) are generated for the source service A, and another set of classes (MPC B, and DPC B) are generated for the target service B.

While the MPC for the target service is included in the MPC of the source service (MPC B is included in MPC_EXT A), the DPC generated for the target service is inherited from the DPC of the source service, (DPC_EXT B inherits from DPC B, which also inherits from DPC_EXT A).

Use Cases for Redefining Services

Below, you will find explanation for the extensibility functionality based on specific use cases and an example.

The following are the use cases:

- Use case 1: Adding a new property to an existing entity

- Use case 2: Adding a new entity
- Use case 3: Adding a new Function import
- Use case 4: Adding an association and a navigation property

In the description of the use cases, we refer to the following example service, `zSales_Order_Example`, which contains two entity sets:

- `SalesOrderSet`: Data of sales orders (header)
- `SalesOrderItemSet`: Data of items ordered

For our example, you will extend the source service by creating a new project, `ZEXTEND_SALES_ORDER_EXAMPLE`, which is referred to as the target service.

Use Case 1: Adding a New Property to an Existing Entity

The required process for adding a new property to your target service, `ZEXTEND_SALES_ORDER_EXAMPLE`, depends on how the service is implemented.

In the sections that follow, you will find explanations for the following:

- DDIC binding that matches the data source attributes
- The entity in the data model is connected to a DDIC structure
- The entity in the data model is not connected to a data dictionary (DDIC) structure

Data Dictionary Structure Binding that Matches the Data Source Attributes

The following is the process for extending an entity in a model that is connected to a Data Dictionary (DDIC) structure.

When the connected DDIC structure matches the data source structure (or its attributes by name), and the service has been implemented using the `MOVE` command, (either single `MOVE`, or `MOVE-CORRESPONDING`) to assign a value to `ER_ENTITY`, it will be sufficient to extend the DDIC structure, and to extend the model as described in steps 1 to 3 of the section, “The Entity in the Data Model is Connected to a Data Dictionary Structure”, without changing the code.

Below is an example code for the entity set, `SALES_ORDER_ITEM Get Entity`, where the `MOVE` command is used, `(MOVE ls_et_order_item TO er_entity`, to fill the `er_entity`; once the structure is extended, it is automatically moved in the code.

Syntax

```
METHOD salesorderitemss_get_entity.

*-------------------------------------------------------------

*Data declaration

*-------------------------------------------------------------

DATA lv_sales_order_id TYPE zif_zsales_order_get=>char25.

DATA lv_sales_order_item TYPE zif_zsales_order_get=>char25.DATA
et_order_item TYPE zif_zsales_order_get=>zsales_order_item_table.

DATA ls_et_order_item TYPE LINE OF
zif_zsales_order_get=>zsales_order_item_table

DATA ls_converted_keys LIKE er_entity.

DATA lv_source_entity_set_name TYPE string.

*-------------------------------------------------------------

*Map the runtime request to the RFC - Only mapped attributes

*-------------------------------------------------------------

*Get all input information from the technical request context object

io_tech_request_context->get_converted_keys(

IMPORTING

es_key_values = ls_converted_keys

).

*Maps key fields to function module parameters

lv_source_entity_set_name = io_tech_request_context-
>get_source_entity_set_name( ).

lv_sales_order_item = ls_converted_keys-salesorderitempos.

lv_sales_order_id = ls_converted_keys-salesorderid.

*-------------------------------------------------------------

*Call RFC function module

*-------------------------------------------------------------

TRY.

CALL FUNCTION 'ZSALES_ORDER_GET'

EXPORTING

iv_sales_order_item = iv_sales_order_item
```

```
iv_sales_order_id = iv_sales_order_id
IMPORTING
et_order_item = et_order_item
EXCEPTIONS system_failure = 1000 message
lv_exc_msg
OTHERS = 1002.
*-----------------------------------------------------------------------
---*
* - Post Backend Call -
*-----------------------------------------------------------------------
---*
* Map properties from the backend to the Gateway output response
structure
* In GetEntity operation, we support only read for the first entry in
the response table
READ TABLE et_order_item INTO ls_et_order_item INDEX 1.
MOVE ls_et_order_item TO er_entity.
ENDMETHOD.
```

Ensure that, the new property (ABAP field name) name is the same as the name of the data source attribute.

You can enable existing conversion exits for each new property that you add. See the section on *Enabling DDIC Conversion Exits*.

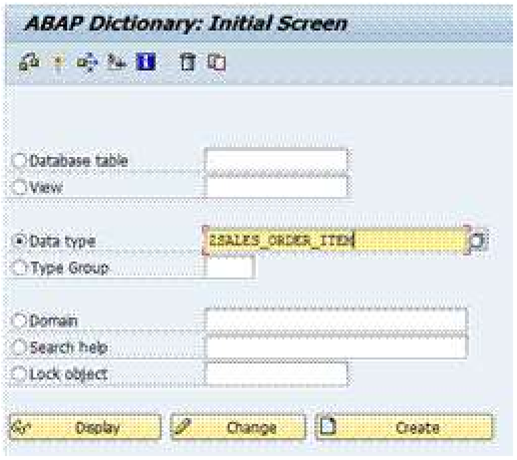
The Entity in the Data Model is Connected to a Data Dictionary Structure

You enhance the structure by adding a new attribute to the DDIC structure. For the new attribute, you must create a new property in the specific entity of the model.

Prerequisites: The DDIC can be enhanced.

The following is the process for extending an entity in a model that is connected to a DDIC structure:

1. Add a new field to the DDIC structture
    1. Enter the transaction `SE11`, and choose *Data Type*, and then enter the name of the structure you want to enhance.

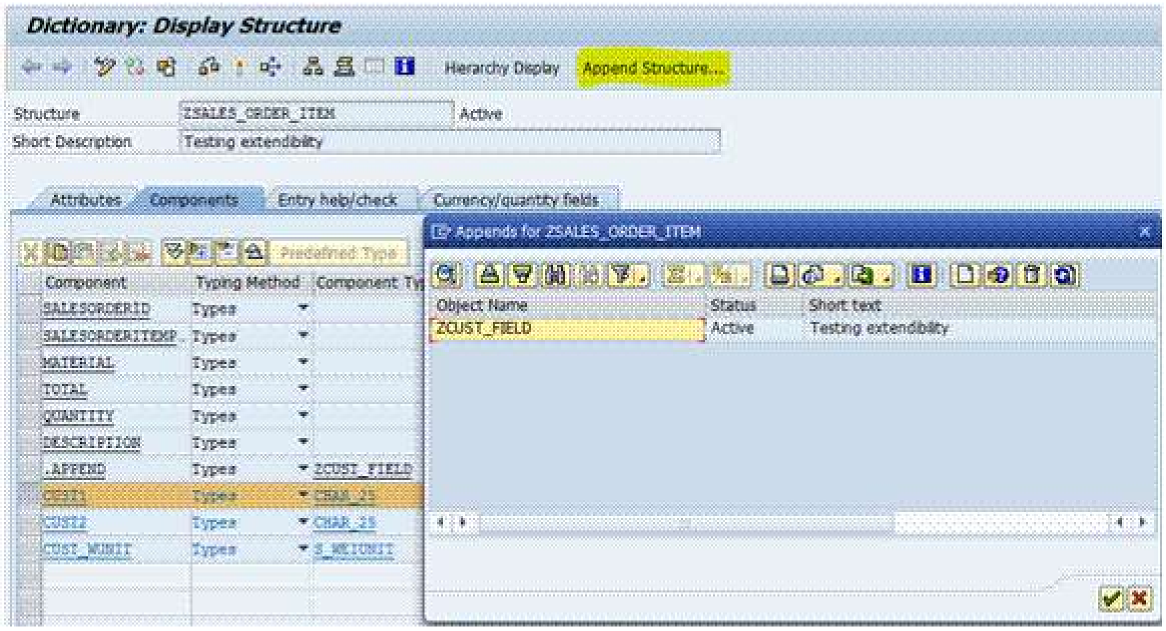

ABAP Dictionary: Initial Screen

Database table	
View	
(●) Data type	ZSALES_ORDER_ITEM
Type Group	
Domain	
Search help	
Lock object	

ABAP Dictionary screen

2. Choose *Display*. In our example, the structure is, `ZSALES_ORDER_ITEM` .
3. Click *GoTo* in the menu, and select *Append Structure* F5 to create the append structure with the additional fields.

   If you have already appended to the structure you can extend it. For example, we created an append structure called, `ZCUST_FIELD`, and added fields `CUST1` and `CUST2`, to it.


Dictionary: Display Structure

Structure ZSALES_ORDER_ITEM Active
Short Description Testing extendibility

Component	Typing Method	Component Ty
SALESORDERID	Types	
SALESORDERITEMP	Types	
MATERIAL	Types	
TOTAL	Types	
QUANTITY	Types	
DESCRIPTION	Types	
.APPEND	Types	ZCUST_FIELD
CUST1	Types	CHAR_25
CUST2	Types	CHAR_25
CUST_WUNIT	Types	S_WEIUNIT

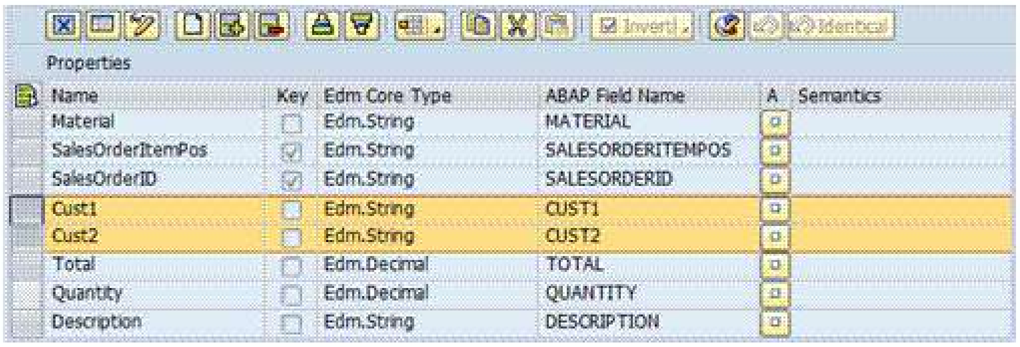
Appends for ZSALES_ORDER_ITEM

Object Name	Status	Short text
ZCUST_FIELD	Active	Testing extendibility

Append structure

4. When you add the fields to the structure, rename the fields using your own suffix or prefix to prevent naming conflicts.

2. Add a new property for the new field to the entity of the model.



Properties

Name	Key	Edm Core Type	ABAP Field Name	A	Semantics
Material	☐	Edm.String	MATERIAL		
SalesOrderItemPos	✓	Edm.String	SALESORDERITEMPOS		
SalesOrderID	✓	Edm.String	SALESORDERID		
Cust1	☐	Edm.String	CUST1		
Cust2	☐	Edm.String	CUST2		
Total	☐	Edm.Decimal	TOTAL		
Quantity	☐	Edm.Decimal	QUANTITY		
Description	☐	Edm.String	DESCRIPTION		

Newly defined properties

In your target service, add the new property to entity of the model and save it.

In our example, we added the fields, `CUST1` and `CUST2`, to the entity type, `SALESORDER`.

3. Save and generate the project for the target service. The new classes of the target service are generated.
4. Update the code for the methods of the specific extended entity in the DPC (extension) class for the target service, `ZEXTEND_SALES_ORDER_EXAMPLE`.

   To update the methods, you can use one of the following options:

   - Redefine the class and the methods for the CRUDQ operations of the specific Entity Set.

     Copy the code from the original method and update it by providing the additional fields.
   - Call the super class and add the additional fields in your code.

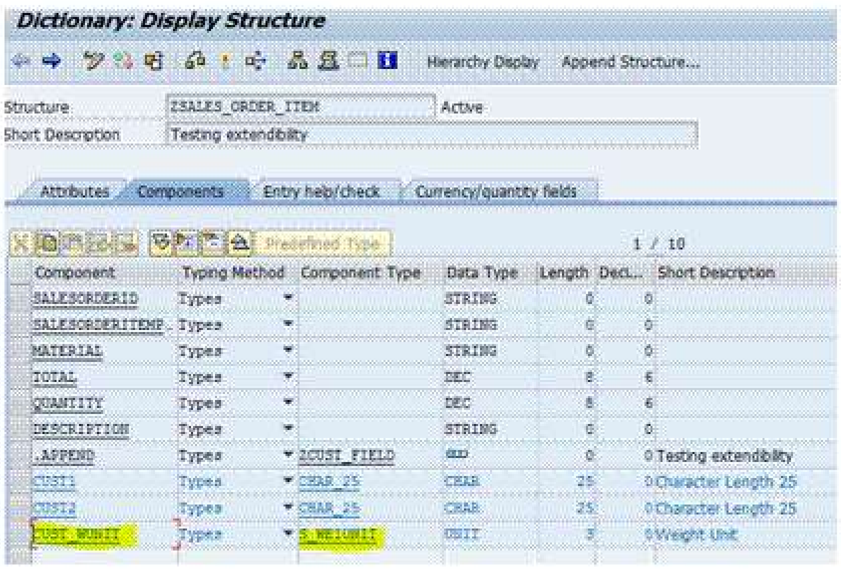
   See the section, *Update the code in the methods and provide the additional fields*.

   If the service has been implemented using SAP tools, such as, `Explicit Enhancement Point`, or BADI, you can use it, instead of redefining the class method.

You can enable existing conversion exits for each new property that you add. See the section on *Enabling DDIC Conversion Exits*.

Enabling DDIC Conversion Exits

In our example, the entity type, `ZSales_Order_Item`, has a new property `CUST_WUNIT`, with data element that has a conversion exit.



Structure: ZSALES_ORDER_ITEM Active
Short Description: Testing extendibility

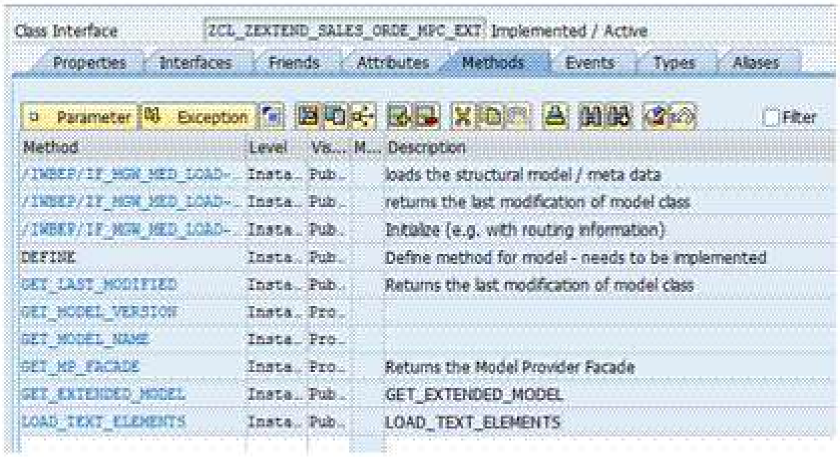
1 / 10

Component	Typing Method	Component Type	Data Type	Length	Deci...	Short Description
SALESORDERID	Types		STRING	0	0	
SALESORDERITEMP	Types		STRING	0	0	
MATERIAL	Types		STRING	0	0	
TOTAL	Types		DEC	8	6	
QUANTITY	Types		DEC	8	6	
DESCRIPTION	Types		STRING	0	0	
.APPEND	Types	ZCUST_FIELD		0	0	Testing extendibility
CUST1	Types	CHAR_25	CHAR	25	0	Character Length 25
CUST2	Types	CHAR_25	CHAR	25	0	Character Length 25
CUST_WUNIT	Types	S_WEIUNIT	UNIT	3	0	Weight Unit

Using existing conversion exit in a DDIC strucutre

To enable the new property to use existing conversion exits at runtime, do the following:

1. Open the MPC extension class of the target service, `ZCL_ZEXTEND_SALES_ORDE_MPC_EXT`.
2. Select the method, *DEFINE*, and choose the *Redefine* button.



Class Interface: ZCL_ZEXTEND_SALES_ORDE_MPC_EXT Implemented / Active

Method	Level	Vis...	M...	Description
/IWBEP/IF_MGW_MED_LOAD~...	Insta...	Pub...		loads the structural model / meta data
/IWBEP/IF_MGW_MED_LOAD~...	Insta...	Pub...		returns the last modification of model class
/IWBEP/IF_MGW_MED_LOAD~...	Insta...	Pub...		Initialize (e.g. with routing information)
DEFINE	Insta...	Pub...		Define method for model - needs to be implemented
GET_LAST_MODIFIED	Insta...	Pub...		Returns the last modification of model class
GET_MODEL_VERSION	Insta...	Pro...		
GET_MODEL_NAME	Insta...	Pro...		
GET_MP_FACADE	Insta...	Pro...		Returns the Model Provider Facade
GET_EXTENDED_MODEL	Insta...	Pub...		GET_EXTENDED_MODEL
LOAD_TEXT_ELEMENTS	Insta...	Pub...		LOAD_TEXT_ELEMENTS

Redefining a method

3. Call the super class.
4. For each entity (that has a new property) with conversion exits that you want to use, assign the DDIC structure.

For example:

Syntax

```
METHOD define.

DATA lo_property TYPE REF TO /iwbep/if_mgw_odata_property.

DATA lo_entity_type TYPE REF TO /iwbep/if_mgw_odata_entity_typ.

super->define( ).

* Bind data structure 'zsales_order_items' to entity type
'SalesOrderItems'( Activate the new fields conversion exit )

lo_entity_type=model->get_entity_type('SalesOrderItems').

lo_entity_type->bind_structure

(iv_bind_conversions = abap_true iv_structure_name =
'zsales_order_items').

ENDMETHOD.
```

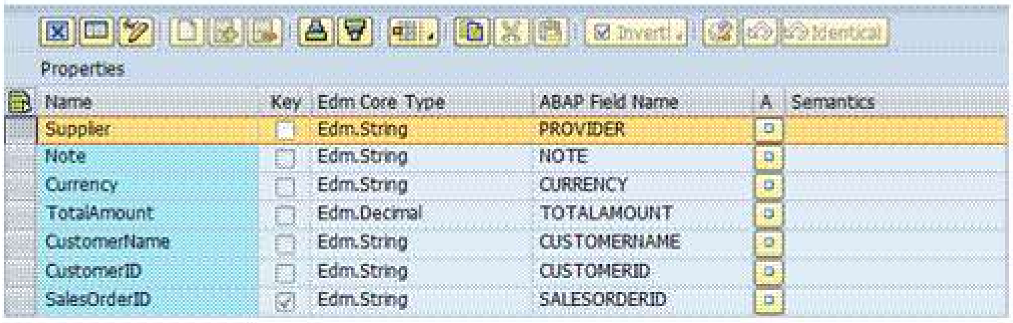
)1.

In the example above, the conversion exits assigned to data element, `S_WEIUNIT`, will now be available at runtime.

The Entity in the Data Model is not Connected to a Data Dictionary Structure

The following is the process for extending an entity in a model that is not connected to a DDIC structure:

1. Add a new property to the entity type and generate the service, `ZEXTEND_SALES_ORDER_EXAMPLE`. In our example, we added a new property, `Supplier`, to the entity type, `SALESORDER`.

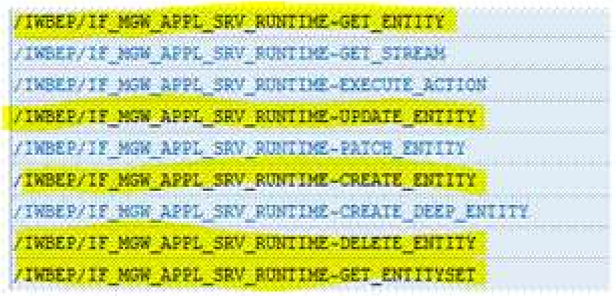


Properties

Name	Key	Edm Core Type	ABAP Field Name	A	Semantics
Supplier	☐	Edm.String	PROVIDER	▫	
Note	☐	Edm.String	NOTE	▫	
Currency	☐	Edm.String	CURRENCY	▫	
TotalAmount	☐	Edm.Decimal	TOTALAMOUNT	▫	
CustomerName	☐	Edm.String	CUSTOMERNAME	▫	
CustomerID	☐	Edm.String	CUSTOMERID	▫	
SalesOrderID	✓	Edm.String	SALESORDERID	▫	

New property entry

2. Redefine the dispatching methods (methods that route to the type CRUDQ methods) of the operations.



```
/IWBEP/IF_MGW_APPL_SRV_RUNTIME~GET_ENTITY
/IWBEP/IF_MGW_APPL_SRV_RUNTIME~GET_STREAM
/IWBEP/IF_MGW_APPL_SRV_RUNTIME~EXECUTE_ACTION
/IWBEP/IF_MGW_APPL_SRV_RUNTIME~UPDATE_ENTITY
/IWBEP/IF_MGW_APPL_SRV_RUNTIME~PATCH_ENTITY
/IWBEP/IF_MGW_APPL_SRV_RUNTIME~CREATE_ENTITY
/IWBEP/IF_MGW_APPL_SRV_RUNTIME~CREATE_DEEP_ENTITY
/IWBEP/IF_MGW_APPL_SRV_RUNTIME~DELETE_ENTITY
/IWBEP/IF_MGW_APPL_SRV_RUNTIME~GET_ENTITYSET
```

Typed methods for operations

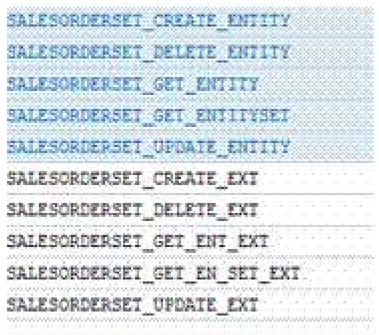
Open the DPC extension class of the target service, `ZEXTEND_SALES_ORDER_EXAMPLE`, and redefine the dispatching methods for `Create, Read, Update`, and `Query (CRUQ)` operations for the extended entity:

- /IWBEP/IF_MGW_APPL_SRV_RUNTIME~GET_ENTITY
- /IWBEP/IF_MGW_APPL_SRV_RUNTIME~CREATE_ENTITY
- /IWBEP/IF_MGW_APPL_SRV_RUNTIME~UPDATE_ENTITY
- /IWBEP/IF_MGW_APPL_SRV_RUNTIME~GET_ENTITYSET

For the dispatching method of each operation, enhance the code for the extended entity set and call the super method for all other entities.

Optional: To be consistent with SAP development, follow the next step and build the typed methods, and in the dispatching method route to the typed methods.

3. Copy the typed methods for the `CRUQ` operations that have been assigned to the entity set you want to extend.

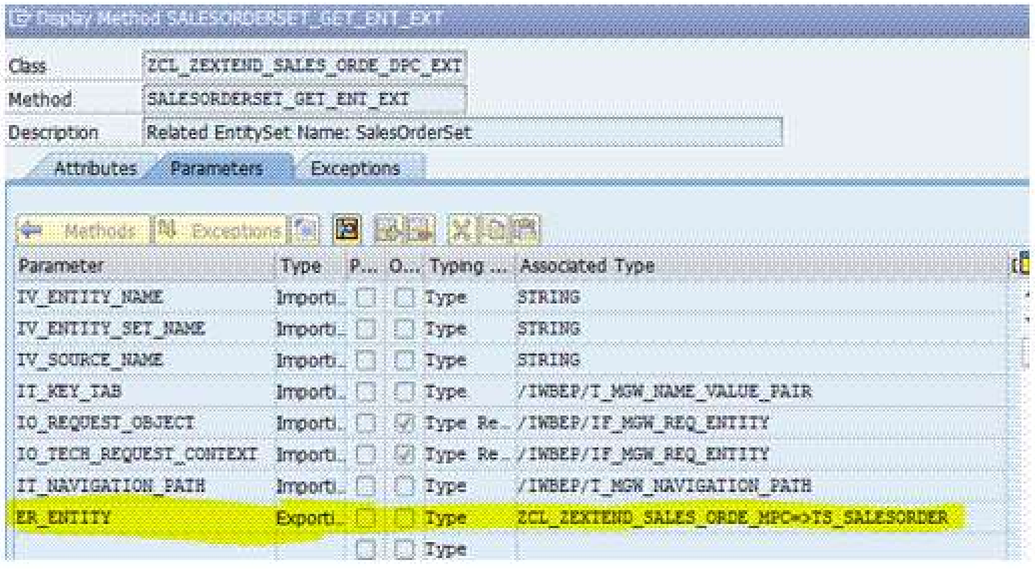


```
SALESORDERSET_CREATE_ENTITY
SALESORDERSET_DELETE_ENTITY
SALESORDERSET_GET_ENTITY
SALESORDERSET_GET_ENTITYSET
SALESORDERSET_UPDATE_ENTITY
SALESORDERSET_CREATE_EXT
SALESORDERSET_DELETE_EXT
SALESORDERSET_GET_ENT_EXT
SALESORDERSET_GET_EN_SET_EXT
SALESORDERSET_UPDATE_EXT
```

Example of method names for the operations

4. Provide a new name, by adding a suffix or prefix, such as, `_EXT`. In the example, we copied the methods in the source service, `ZSALES_ORDER_EXAMPLE`, to the new methods in the target service,

and added the suffix, `_EXT`.



Class	ZCL_ZEXTEND_SALES_ORDE_DPC_EXT
Method	SALESORDERSET_GET_ENT_EXT
Description	Related EntitySet Name: SalesOrderSet

Parameter	Type	P...	O...	Typing ...	Associated Type
IV_ENTITY_NAME	Importi..	☐	☐	Type	STRING
IV_ENTITY_SET_NAME	Importi..	☐	☐	Type	STRING
IV_SOURCE_NAME	Importi..	☐	☐	Type	STRING
IT_KEY_TAB	Importi..	☐	☐	Type	/IWBEP/T_MGW_NAME_VALUE_PAIR
IO_REQUEST_OBJECT	Importi..	☐	✓	Type Re..	/IWBEP/IF_MGW_REQ_ENTITY
IO_TECH_REQUEST_CONTEXT	Importi..	☐	✓	Type Re..	/IWBEP/IF_MGW_REQ_ENTITY
IT_NAVIGATION_PATH	Importi..	☐	☐	Type	/IWBEP/T_MGW_NAVIGATION_PATH
ER_ENTITY	Exporti..	☐	☐	Type	ZCL_ZEXTEND_SALES_ORDE_MPC=>TS_SALESORDER
		☐	☐	Type	

Copied methods

5. Update the method interfaces with the model provider class (MPC extension) types in your new project.

   Once you have the new methods, you can update their interfaces.

   > Note
   >
   > Note that the new methods are not inherited, therefore you can update their interfaces.

   For each method, copy the type structure in the MPC extension class of the source service, `ZSALES_ORDER_EXAMPLE,` and replace the type structure in the MPC extension class in the target service, `ZEXTEND_SALES_ORDER_EXAMPLE`.

   Make changes to the methods of the operations as follows:

   - CREATE operation

     Change the `ER_ENTITY` type

     In the source service, `ZSALES_ORDER_EXAMPLE,` the MPC extension class displays, `ZCL_ZSALES_ORDER_EXAMP_MPC=>TS_SALESORDER`.

     In the target service, `ZEXTEND_SALES_ORDER_EXAMPLE`, edit the MPC extension class to,`ZCL_ZEXTEND_SALES_ORDE_MPC=>TS_SALESORDER`.

   - DELETE operation: No changes are required for this operation, since you cannot copy the method.
   - READ (GET_ENTITY) operation

     Change the `ER_ENTITY` type as follows:

  In the source service, `ZSALES_ORDER_EXAMPLE`, the MPC extension class displays as, `ZCL_ZSALES_ORDER_EXAMP_MPC=>TS_SALESORDER`.

  In the target service, `ZEXTEND_SALES_ORDER_EXAMPLE`, edit the MPC extension class to,`ZCL_ZEXTEND_SALES_ORDE_MPC=>TS_SALESORDER`.

- QUERY (GET_ENTITYSET) operation

  Change the `ET_ENTITY` type from:

  In the source service, `ZSALES_ORDER_EXAMPLE`, the MPC extension class displays as, `ZCL_ZSALES_ORDER_EXAMP_MPC=>TT_SALESORDER`.

  In the target service, `ZEXTEND_SALES_ORDER_EXAMPLE`, edit the MPC extension class to, `ZCL_ZEXTEND_SALES_ORDE_MPC=>TT_SALESORDER`.

- UPDATE operation

  Change the `ER_ENTITY` type as follows:

  In the source service, `ZSALES_ORDER_EXAMPLE`, the MPC extension class displays as, `ZCL_ZSALES_ORDER_EXAMP_MPC=>TS_SALESORDER`.

  In the target service, `ZEXTEND_SALES_ORDER_EXAMPLE`, edit the MPC extension class to, `ZCL_ZEXTEND_SALES_ORDE_MPC=>TS_SALESORDER`.

6. Update the code in the methods, and provide the additional fields.

   You have two main options to update the code. The option you choose depends on the logic you want to implement:

   - Call the original method of the entity operation; and then provide the code for the additional field.
   - Copy the code for the method of the specific Entity Set, and the specific operation, from the original method; and update the code for the additional fields.

   If you choose to implement the first option, you will be using SAP code with your own code for the additional fields. The disadvantage is that, it you may call the backend twice; once for the SAP provided fields, and then again for the additional fields.

7. Redefine the dispatching methods.

   The next process is to redefine each dispatching method in the DPC (extension) class.

   There are typed methods with MPC type, and dispatching methods. At runtime the dispatching methods, one method per each CRUDQ operation, are called with a generic interface that dispatches the call to specific typed methods.

   The specific call is based on the Entity Set name.

   In each dispatch method, verify that the specific entity is the extended entity, and then route the entity to the new methods, if not, route it to a super dispatching method.

Syntax

Below is an example code for Get Entity:

```
METHOD /iwbep/if_mgw_appl_srv_runtime~get_entity.
DATA salesorderset_get_entityset
TYPE zcl_zextend_sales_orde_mpc=>ts_salesorder.
DATA lv_entityset_name TYPE string.
DATA lr_entity TYPE REF TO data.
lv_entityset_name = io_tech_request_context->get_entity_set_name( ).
CASE lv_entityset_name.
*-------------------------------------------------------------------------
--- *
* EntitySet - SalesOrderSet
*-------------------------------------------------------------------------
---
* WHEN 'SalesOrderSet'.
* Call the entity set generated method
salesorderset_get_ent_ext(
EXPORTING iv_entity_name = iv_entity_name
iv_entity_set_name = iv_entity_set_name
iv_source_name = iv_source_name
it_key_tab = it_key_tab
it_navigation_path = it_navigation_path
io_tech_request_context = io_tech_request_context
IMPORTING er_entity = salesorderset_get_entityset ).
IF salesorderset_get_entityset IS NOT INITIAL.
* Send specific entity data to the caller interface
copy_data_to_ref(
EXPORTING
is_data = salesorderset_get_entityset
CHANGING
cr_data = er_entity ).
ELSE.
```

```
* In case of initial values - unbind the entity reference er_entity =
lr_entity.
ENDIF.
*-----------------------------------------------------------------------
---*
* Other enteties => call SAP model
*-----------------------------------------------------------------------
---
WHEN OTHERS.
TRY.
CALL METHOD super->/iwbep/if_mgw_appl_srv_runtime~get_entity
EXPORTING
iv_entity_name = iv_entity_name
iv_entity_set_name = iv_entity_set_name
iv_source_name = iv_source_name
it_key_tab = it_key_tab it_navigation_path = it_navigation_path
io_tech_request_context = io_tech_request_context
IMPORTING
er_entity = er_entity.
CATCH /iwbep/cx_mgw_busi_exception.
* todo
CATCH /iwbep/cx_mgw_tech_exception.
* todo
ENDTRY.
ENDCASE.
ENDMETHOD.
```

Use Case 2: Adding a New Entity

The following are the processes for adding a new entity to extend the OData service:

1. In your target service, `ZEXTEND_SALES_ORDER_EXAMPLE`, create and add the new entity, and then generate the project.

   For more information, see Entity Types
2. Manually create new methods for the CRUDQ operations you require for the newly created entity.

You can copy existing methods generated for another entity and use it to update your new entity, and add the relevant code.

3. Redefine the dispatching method.

   The dispatching methods are inherited from the DPC source service, and these methods do not recognize the new entity; therefore you must redefine these methods in your target service.

   See the section on “Redefine the dispatching methods”.

Use Case 3: Adding a New Function Import

The following are the enhancement processes for extending an OData service by adding a new function import to the data model:

1. Add a new function import into the model.
2. Manually redefine the function import method,`</IWBEP/IF_MGW_CORE_SRV_RUNTIM =>EXEC_SERVICE_OPERATION>`, and write your own code.

Use Case 4: Adding an Association and a Navigation Property

The following are the enhancement processes for extending an OData service by adding an association and a navigation property:

1. In the target service, add the new association, navigation property, and referential constraint to the data model.
2. Update the code in the query and read operations of the target entity.

More Information

Function Imports

Redefining OData Service (GW)

Creating a Service Builder Project

Defining Properties

Associations

`http://help.sap.com/saphelp_45b/helpdata/en/34/8e732a6df74873e10000009b38f9b8/content.htm`

MOVE-CORRESPONDING in ABAP - The SAP Programming Language.