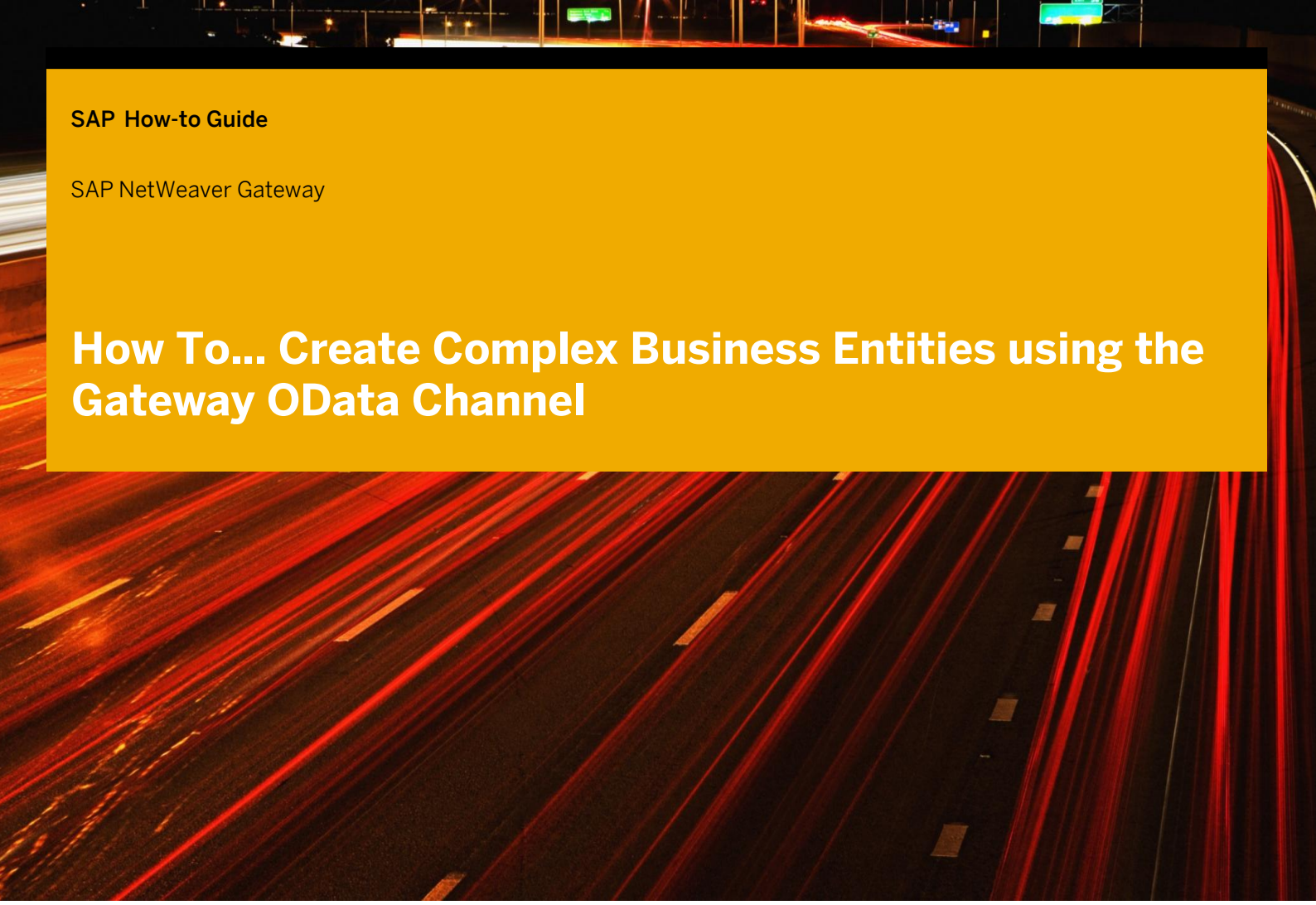


SAP How-to Guide

SAP NetWeaver Gateway

How To... Create Complex Business Entities using the Gateway OData Channel

Applicable Releases:

SAP NetWeaver Gateway 2.0 ≥ SP03

SAP NetWeaver 7.02 ≥ SP07

Version 2.0

February 2012

The Best-Run Businesses Run SAP™

Document History

Document Version	Description
1.00	First official release of this guide
2.00	Updated to account for changes delivered in SP03

Typographic Conventions

Type Style	Description
Example Text	Words or characters quoted from the screen. These include field names, screen titles, pushbuttons labels, menu names, menu paths, and menu options. Cross-references to other documentation
Example text	Emphasized words or phrases in body text, graphic titles, and table titles
`Example text`	File and directory names and their paths, messages, names of variables and parameters, source text, and names of installation, upgrade and database tools.
`Example text`	User entry texts. These are words or characters that you enter in the system exactly as they appear in the documentation.
`<Example text>`	Variable user entry. Angle brackets indicate that you replace these words and characters with appropriate entries to make entries in the system.
`EXAMPLE TEXT`	Keys on the keyboard, for example, `F2` or `ENTER`.

Icons

Icon	Description
	Caution
	Note or Important
	Example
	Recommendation or Tip

Table of Contents

1. Business Scenario

Some business objects, such as Sales Orders or Purchase Orders, consist of header and line item data. Thus, when creating such business objects, it makes sense to use hierarchical or nested data in the creation request. SAP NetWeaver Gateway can be used to create and expose a service that allows for the creation of these business objects.

2. Background Information

In SAP NetWeaver Gateway, the OData Channel provides *deep insert* functionality to accommodate the creation of an entity along with its associated entities in one request. The deep insert functionality is provided in ABAP interface /IWBEP/IF_MGW_APPL_SRV_RUNTIME, method CREATE_DEEP_ENTITY.

This document covers how to create and test an SAP NetWeaver Gateway service that can be used to QUERY, READ, and CREATE a Sales Order in an ECC system (ECC 6.0 in this guide). However, the main focus is on the CREATE operation using the deep insert functionality provided in the OData channel starting with SAP NetWeaver Gateway 2.0, SP2.

3. Prerequisites

The following are the prerequisites to complete the steps in this guide:

- You have access to an SAP NetWeaver 7.02 SP6 or higher system into which the SAP NetWeaver Gateway ABAP add-ons have been installed.
- Backend SAP ECC 6.0 or higher with the IW_BEP Gateway add-on installed.
- Connections between SAP NetWeaver Gateway system and backend SAP application are configured.
- You have at least a basic understanding of ABAP development.
- Firefox and the Firefox REST add-on for testing.

  Other REST clients (e.g. WFetch) can be used as well, but the instructions in this guide are specifically for the Firefox REST client.
- If you are using Internet Explorer to test the QUERY and READ operations as specified in this guide, make sure it is not configured to show "friendly error messages".

  To configure, go to Tools → Internet Options → Advanced → Uncheck "Show friendly HTTP error messages". This will help you analyze any errors if they occur.

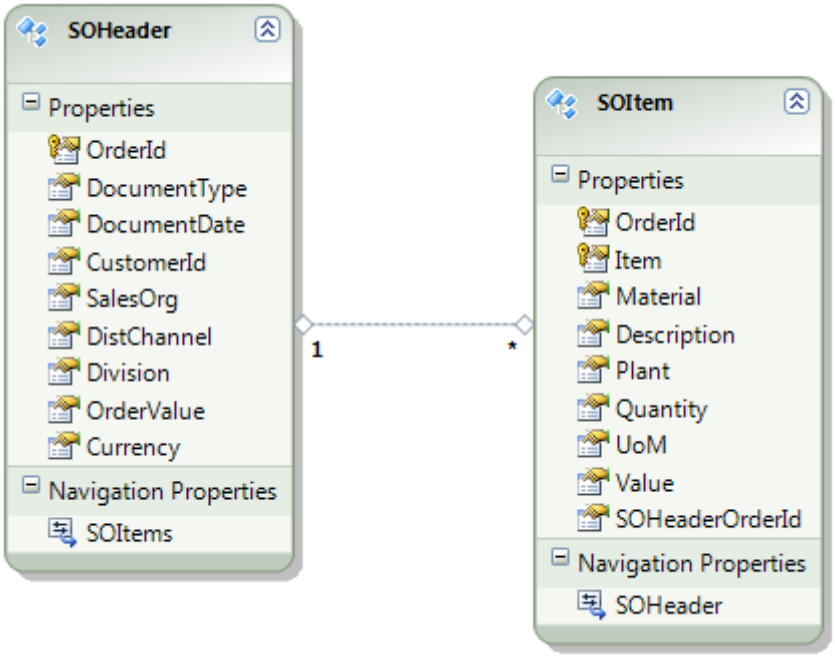
For more information on these topics, refer to the following:

- SAP NetWeaver Gateway Developer Guide – Backend OData Channel
- OData – Creating New Entries
- SAP NetWeaver Gateway Page on SCN
- SAP NetWeaver Gateway How-To Guides on SCN

More information on OData can be found at http://www.odata.org.

4. Step-by-Step Procedure

From a data modeling standpoint, the Sales Order business object in ECC will be represented by two entity types – SOHeader for the sales order header data and SOItem for sales order item data. As shown in the figure below, a relationship or *association* is established between the SOHeader entity type and SOItem entity type where for every SOHeader, there can be many SOItems.



You might also notice that each entity type has a *navigation property* that basically allows for navigation from one end of an association to another. In practical terms, this means that given an instance of a sales order header, you can navigate to all of its associated sales order items. Also, vice versa, given a sales order item, you can navigate to its associated sales order header.

It is important to understand the relationship shown in the data model as described above since using the deep insert feature in the Gateway OData channel is based on the underlying associations and navigation properties established between these entities.

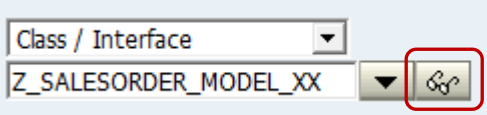
The following sections walk you through the steps of creating and implementing the model and runtime classes necessary to produce a Gateway service that executes the creation of a sales order. Only specifics as it relates to the deep insert functionality will be elaborated on. Basics on the Gateway modeling and runtime can be found in other Gateway How-To Guides available on SDN.

4.1 Create the Model Class

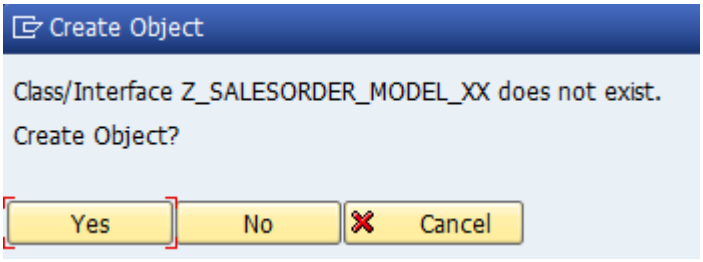
The Model class defines the metadata of your OData Channel service. This section shows you how to create this class and which super class it inherits from.

1. Logon to the SAP backend system and go to the ABAP Development Workbench – transaction SE80.
2. In the Object Navigator, choose the object type *Class/Interface* from the drop down.

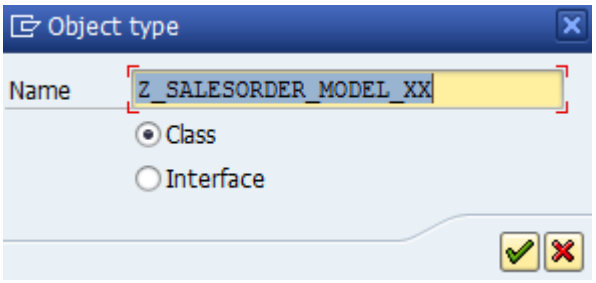
3. Enter `Z_SALESORDER_MODEL_<XX>` as the name of the Data Model and click on the Display (eye glasses) button. Replace *<XX>* with some unique identifier (e.g. your initials or student number).



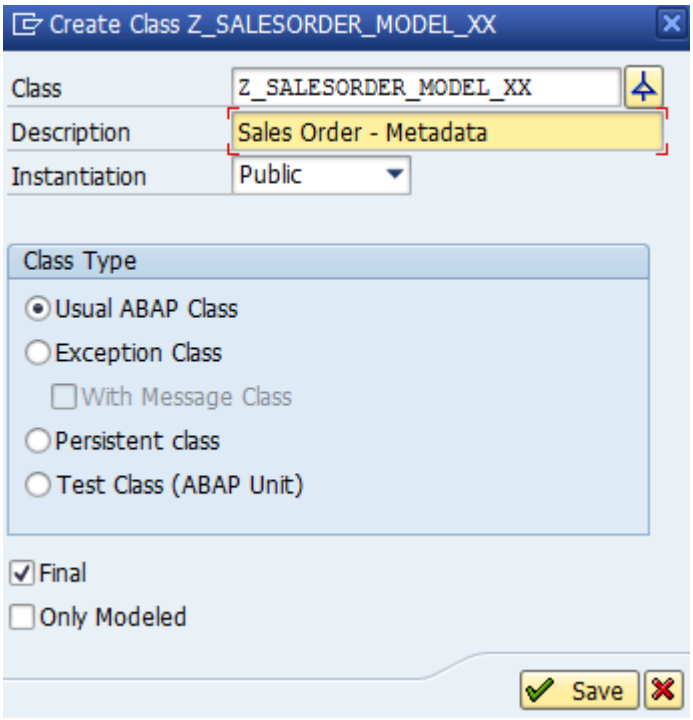
4. Select *Yes* when prompted to create the class.



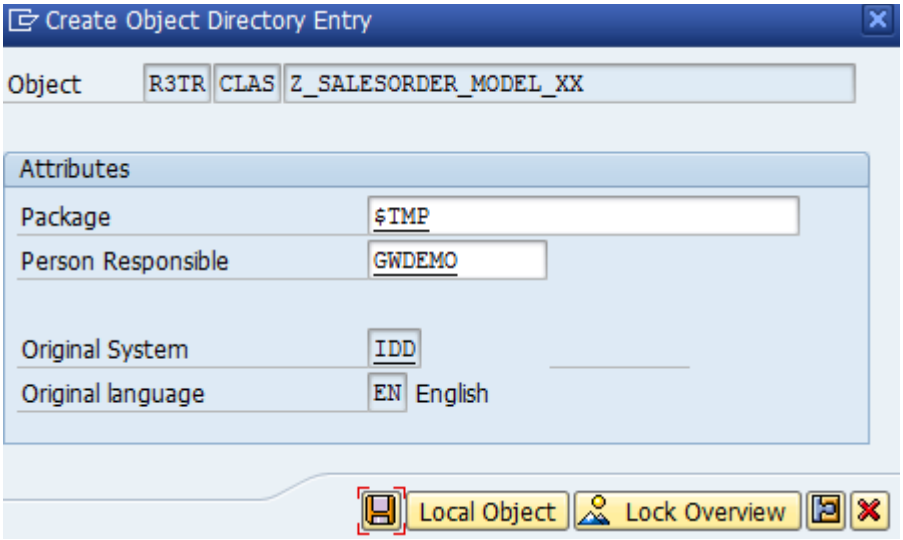
5. Accept the default *Class* option and click  to continue.



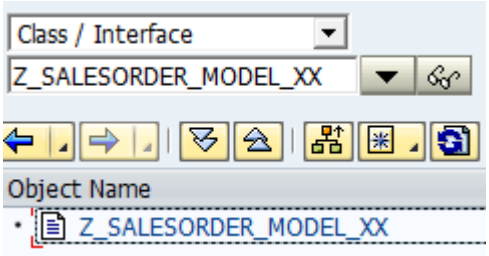
6. Enter a description and click *Save*.



7. Specify a package to which your objects can be saved for transport purposes. Select an appropriate package or select *Local Object* to save as local objects that will not be transported.

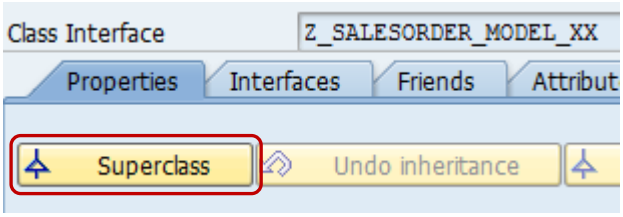


8. Double-click on Z_SALESORDER_MODEL_XX.

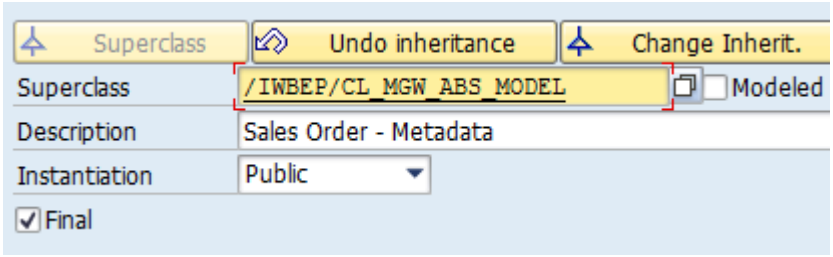


9. Switch to edit mode by clicking on the Display<->Change icon.

10. On the right hand side, click on the Properties tab.
11. Click on the *Superclass* button.

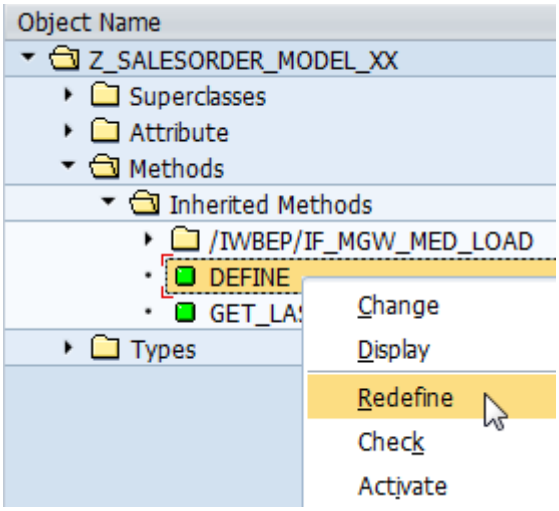


12. Enter `/IWBEP/CL_MGW_ABS_MODEL` in the Superclass field. Accept default options for the rest.



13. Save your class.

14. Now, among other nodes, you should see a *Methods* node under your class on the left panel. Expand it, right-click on method *DEFINE* and select *Redefine*.
15. Now redefine the Define method.

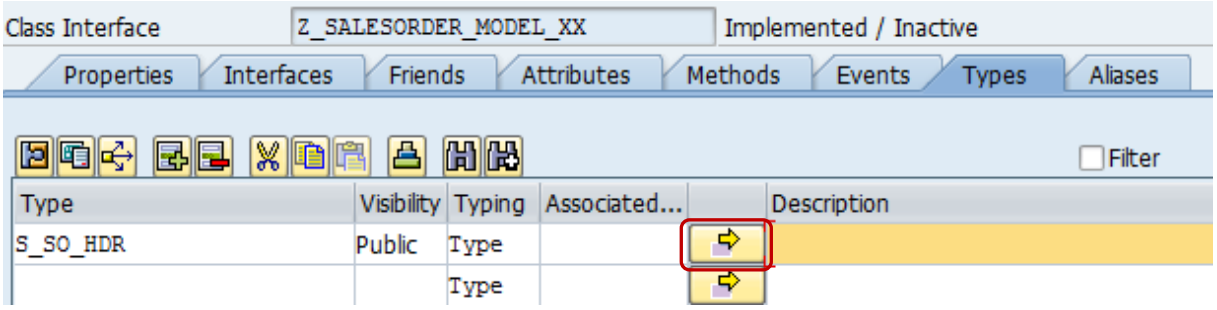


16. The code for this method is given below.

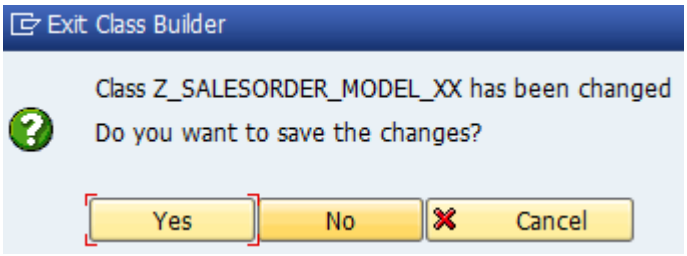
```
METHOD define.

  DATA: lo_entity_type    TYPE REF TO /iwbep/if_mgw_odata_entity_typ,
        lo_entity_set     TYPE REF TO /iwbep/if_mgw_odata_entity_set,
        lo_association    TYPE REF TO /iwbep/if_mgw_odata_assoc,
        lo_nav_property   TYPE REF TO /iwbep/if_mgw_odata_nav_prop,
        lo_property       TYPE REF TO /iwbep/if_mgw_odata_property.

*--------------------------------------------------------------------------------
* Sales Order Header
*--------------------------------------------------------------------------------

* Create the SalesOrderHeader data object
  lo_entity_type = model->create_entity_type( 'SOHeader' ).

* Define the properties of the SalesOrderHeader object
  lo_property = lo_entity_type->create_property( 'OrderId' ).

* set it as a key element
  lo_property->set_is_key( ).

  lo_property = lo_entity_type->create_property( 'DocumentType' ).
  lo_property = lo_entity_type->create_property( 'DocumentDate' ).
  lo_property = lo_entity_type->create_property( 'CustomerId' ).
  lo_property = lo_entity_type->create_property( 'SalesOrg' ).
  lo_property = lo_entity_type->create_property( 'DistChannel' ).
  lo_property = lo_entity_type->create_property( 'Division' ).
  lo_property = lo_entity_type->create_property( 'OrderValue' ).
  lo_property = lo_entity_type->create_property( 'Currency' ).

* Bind your structure that represents the above properties
  lo_entity_type->bind_structure( 'Z_SALESORDER_MODEL_XX=>S_SO_HDR' ).

* Name for the collection
  lo_entity_type->create_entity_set( 'SOHeaders' ).

*--------------------------------------------------------------------------------
* Sales Order Item
*--------------------------------------------------------------------------------

* Create the SalesOrderItem data object
```

```
  lo_entity_type = model->create_entity_type( 'SOItem' ).

* Define the properties of the SalesOrderItem object
  lo_property = lo_entity_type->create_property( 'OrderId' ).

* set it as a key element
  lo_property->set_is_key( ).

  lo_property = lo_entity_type->create_property( 'Item' ).

* set it as a key element
  lo_property->set_is_key( ).

  lo_property = lo_entity_type->create_property( 'Material' ).
  lo_property = lo_entity_type->create_property( 'Description' ).
  lo_property = lo_entity_type->create_property( 'Plant' ).
  lo_property = lo_entity_type->create_property( 'Quantity' ).
  lo_property = lo_entity_type->create_property( 'UoM' ).
  lo_property = lo_entity_type->create_property( 'Value' ).

* Bind your structure that represents the above properties
  lo_entity_type->bind_structure( 'Z_SALESORDER_MODEL_XX=>S_SO_ITEM' ).

  lo_entity_type->create_entity_set( 'SOItems' ).

*******************************************************************************
*   ASSOCIATIONS
*******************************************************************************
  "Define Association SO Header to SO Items
  lo_association = model->create_association(
                           iv_association_name = 'SOHeader_SOItems'
                           iv_left_type        = 'SOHeader'
                           iv_right_type       = 'SOItem'
                           iv_right_card       =  cardinality_feed
                           iv_left_card        =  cardinality_entity ).

  "Define Association SO Item to SO Header
  lo_association = model->create_association(
                           iv_association_name = 'SOItem_SOHeader'
                           iv_left_type        = 'SOItem'
                           iv_right_type       = 'SOHeader'
                           iv_right_card       =  cardinality_entity
                           iv_left_card        =  cardinality_entity ).

*******************************************************************************
*   NAVIGATION PROPERTIES
*******************************************************************************
  "Navigation Properties for entity SO Header
  lo_entity_type = model->get_entity_type( iv_entity_name = 'SOHeader' ).

  "Add Navigation Property from role SOHeader to role SOItems
  lo_entity_type->create_navigation_property(
                   iv_property_name    = 'SOItems'
                   iv_association_name = 'SOHeader_SOItems' ).

  "Navigation Properties for entity SO Item
  lo_entity_type = model->get_entity_type( iv_entity_name = 'SOItem' ).

  "Add Navigation Property from role SOItem to SOHeader
  lo_entity_type->create_navigation_property(
                   iv_property_name    = 'SOHeader'
                   iv_association_name = 'SOItem_SOHeader' ).


ENDMETHOD.
```

17. Save the method.
18. Double-click on Z_SALESORDER_MODEL_XX to display the class definition page again and then click on the *Types* tab.
19. For `S_SO_HDR` for Type and select *Public* for Visibility. Then click on the Direct Type Entry button.



20. Save when prompted.



21. Find the line of code:

```
types S_SO_HDR .
```

and replace it with the following:

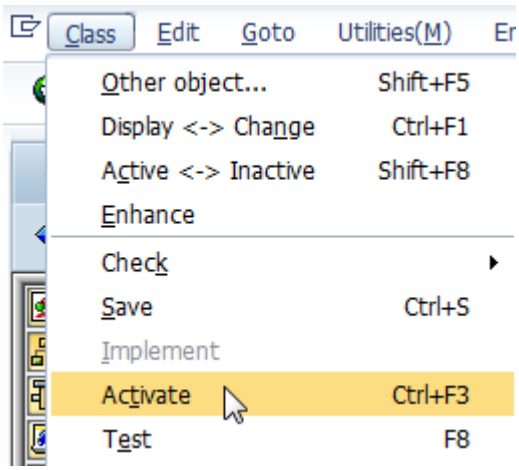
```
types:
  begin of s_so_hdr,
    OrderId              type vbak-vbeln,
    DocumentType         type vbak-auart,
    DocumentDate         type vbak-audat,
    CustomerId           type vbak-kunnr,
    SalesOrg             type vbak-vkorg,
    DistChannel          type vbak-vtweg,
    Division             type vbak-spart,
    OrderValue           type vbak-netwr,
    Currency             type vbak-waerk,
  end of s_so_hdr .

types: t_so_hdr type standard table of s_so_hdr.

types:
  begin of s_so_item,
    OrderId          type vbap-vbeln,
    Item             type vbap-posnr,
    Material         type vbap-matnr,
    Description      type vbap-arktx,
    Plant            type vbap-werks,
    Quantity         type vbap-kwmeng,
    UoM              type vbap-vrkme,
    Value            type vbap-netwr,
  end of s_so_item .


types: t_so_item type standard table of s_so_item.
```

22. Save the type definitions.
23. Click the Back arrow on the top toolbar.

Four type definitions should now be listed.

Type	Visibility	Typing	Associated...		Description
S_SO_HDR	Public				
T_SO_HDR	Public				
S_SO_ITEM	Public				
T_SO_ITEM	Public				

24. From the *Class* menu, select *Activate*.



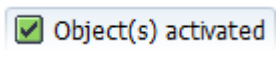
Note that a toolbar button is also available to activate.

A listed of inactive objects should be selected for activation. Click

Object name

D..	Object	Obj. name		User
	CINC	Z_SALESORDER_MODEL_XX=========CCDEF		GWDEMO
	CINC	Z_SALESORDER_MODEL_XX=========CCIMP		GWDEMO
	CINC	Z_SALESORDER_MODEL_XX=========CCMAC		GWDEMO
	CPRI	Z_SALESORDER_MODEL_XX		GWDEMO
	CPRO	Z_SALESORDER_MODEL_XX		GWDEMO
	CPUB	Z_SALESORDER_MODEL_XX		GWDEMO
	METH	Z_SALESORDER_MODEL_XX	DEFINE	GWDEMO
	METH	Z_SALESORDER_MODEL_XX	GET_LAST_MODIFIED	GWDEMO

25. Click  to continue activation.
26. Successful activation is confirmed on the lower left status bar.



Your model class is now complete.

4.2 Create and Implement Data Runtime Classes

In this section all the necessary runtime data classes are defined and implemented. The main runtime class is where you implement your business logic and is the class called when the service is executed. It is based on superclass `/IWBEP/CL_MGW_ABS_DATA` and will be used to manage calls to other entity specific classes that execute the various Query and CRUD (Create, Read, Update, Delete) operations for the service. As mentioned earlier, only the Query, Read, and Create operations will be covered in this guide.

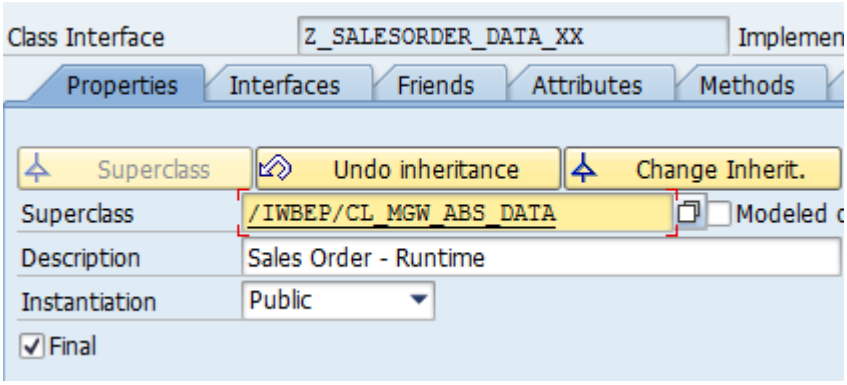
4.2.1 Create the Main Data Runtime Class

This section guides you through the initial setup of the main runtime class.

Note that, in this guide, it is in this class where the deep insert functionality will be implemented – specifically by redefining the CREATE_DEEP_ENTITY method of inherited interface /IWBEP/IF_MGW_APPL_SRV_RUNTIME.

1. In the SAP backend system, go to the ABAP Development Workbench – transaction SE80.
2. Create a new class called `Z_SALESORDER_DATA_<XX>` and assign a package to which your objects can be saved for transport purposes.

   Hint: Look at steps 3 – 7 in the previous section for assistance on creating a class.
3. Double-click on Z_SALESORDER_DATA_XX.
4. Switch to edit mode by clicking on the Display<->Change icon.
5. On the right hand side, click on the Properties tab.
6. Click on the *Superclass* button.
7. Enter `/IWBEP/CL_MGW_ABS_DATA` in the Superclass field. Accept default options for the rest.



8. Save your class.
9. Activate the class.

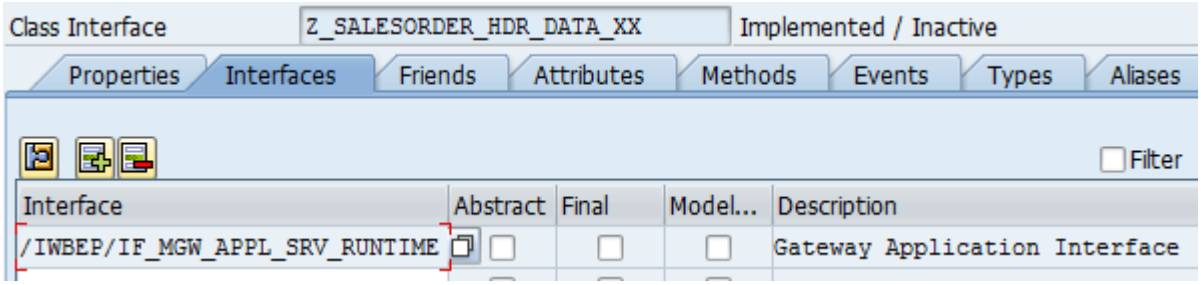
For now, we will leave the main data runtime class unimplemented and create two entity specific data classes in the next section that will be called from this main data class.

4.2.2 Create the Entity Specific Data Classes

Two entity specific data classes corresponding to Sales Order Header (SOHeader) and Sales Order Item (SOItem) are defined in this section. These classes implement the interface /IWBEP/IF_MGW_APPL_SRV_RUNTIME. The respective Query and Read operations for each entity will be implemented within redefined methods of these classes.

1. In the SAP backend system, go to the ABAP Development Workbench – transaction SE80.

2. Create a new class called `Z_SALESORDER_HDR_DATA_<XX>` and assign a package to which your objects can be saved for transport purposes.
3. Double-click on Z_SALESORDER_HDR_DATA_XX.
4. Switch to edit mode and click on the Interfaces tab.
5. Enter `/IWBEP/IF_MGW_APPL_SRV_RUNTIME` and press <Enter>.



6. Save and activate your changes.
7. Repeat steps 2 to 6 to create another class – Z_SALESORDER_ITEM_DATA_<XX>.

   You should now have two, unimplemented data runtime classes for each of our entity types – Sales Order Header (SOHeader) and Sales Order Item (SOItem).

4.2.3 Implement the Entity Specific Data Classes to Enable the Query and Read Operations

The implementation of the Query and Read operations for each entity type is done in this section.

1. In the ABAP Development Workbench (transaction SE80), display class Z_SALESORDER_HDR_DATA_XX (double-click on it to make sure its information is shown on the main panel).
2. Click on the Types tab and switch to edit mode if necessary.
3. Enter `S_VBAK` for the Type, select *Private* for Visibility, and click on the Direct Type Entry  button.
4. Confirm that changes should be saved.
5. Find the line of code:

```
  types S_VBAK .
```

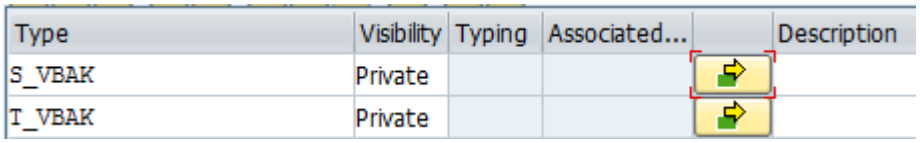
   and replace it with the following:

```
  types:
    begin of s_vbak,
      vbeln   type vbak-vbeln,
      auart   type vbak-auart,
      audat   type vbak-audat,
      kunnr   type vbak-kunnr,
      vkorg   type vbak-vkorg,
      vtweg   type vbak-vtweg,
      spart   type vbak-spart,
      netwr   type vbak-netwr,
      waerk   type vbak-waerk,
    end of s_vbak .


  types: t_vbak type standard table of s_vbak.
```

6. Save the type definitions.

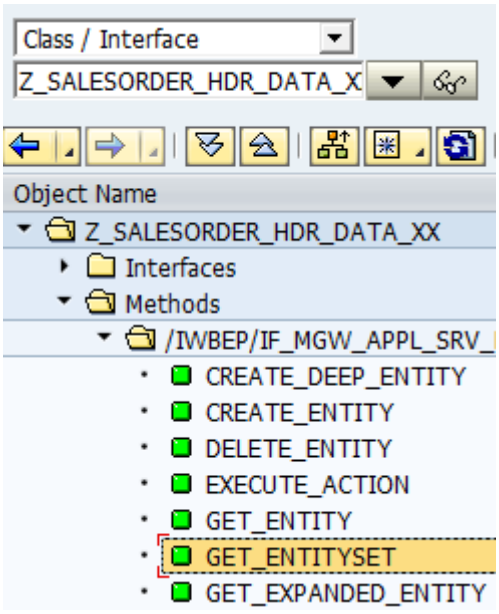
7. Click the Back arrow on the top toolbar.

Two type definitions should now be listed.

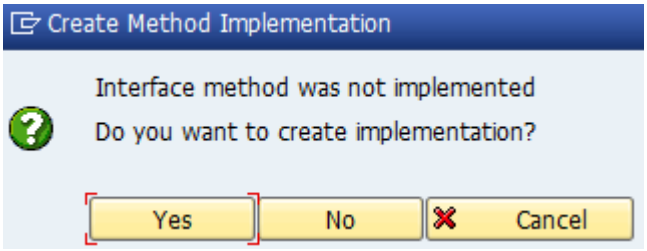


Type	Visibility	Typing	Associated...		Description
S_VBAK	Private				
T_VBAK	Private				

8. Now, expand the class, find method GET_ENTITYSET, and double-click on it.



9. Confirm that you want to create an implementation.



10. Switch to edit mode if necessary.
11. The code for this method is provided below.

```
method /IWBEP/IF_MGW_APPL_SRV_RUNTIME~GET_ENTITYSET.

  data:
    ls_vbak   type s_vbak,
    lt_vbak   type standard table of s_vbak,
    ls_sohdr  type z_salesorder_model_xx=>s_so_hdr,
    lt_sohdrs type z_salesorder_model_xx=>t_so_hdr,
    lv_max    type int4.

  field-symbols:
    <fs_key>      type /iwbep/s_mgw_name_value_pair,
    <fs_sohdrs>   type z_salesorder_model_xx=>t_so_hdr.

* Limit number of items returned to constant value.  $top value will override
if provided.
  lv_max = 100.
```

```
* Create the output data structure and assign its fields to
* the field symbol
  create data er_entityset type z_salesorder_model_xx=>t_so_hdr.
  assign er_entityset->* to <fs_sohdrs>.

  if is_paging-top is not initial.
    lv_max = is_paging-top.
  endif.

  select vbeln auart audat kunnr vkorg vtweg spart netwr waerk
    into corresponding fields of table lt_vbak
    from vbak
   up to lv_max rows.

  loop at lt_vbak into ls_vbak.
    ls_sohdr-OrderId              = ls_vbak-vbeln.
    ls_sohdr-DocumentType         = ls_vbak-auart.
    ls_sohdr-DocumentDate         = ls_vbak-audat.
    ls_sohdr-CustomerId           = ls_vbak-kunnr.
    ls_sohdr-SalesOrg             = ls_vbak-vkorg.
    ls_sohdr-DistChannel          = ls_vbak-vtweg.
    ls_sohdr-Division             = ls_vbak-spart.
    ls_sohdr-OrderValue           = ls_vbak-netwr.
    ls_sohdr-Currency             = ls_vbak-waerk.

    append ls_sohdr to <fs_sohdrs>.
  endloop.


endmethod.
```

12. Save the method.
13. Now repeat steps 8 – 11 to implement method GET_ENTITY using the following code.

```
method /IWBEP/IF_MGW_APPL_SRV_RUNTIME~GET_ENTITY.

  data: ls_sohdr type z_salesorder_model_xx=>s_so_hdr.

  field-symbols:
    <key>        type /iwbep/s_mgw_name_value_pair,
    <fs_sohdr>   type s_vbak.

* Create the output data structure and assign its fields to
* the field symbol
  create data er_entity type z_salesorder_model_xx=>s_so_hdr.
  assign er_entity->* to <fs_sohdr>.

  " In which PO Header are we interested?
  read table it_key_tab assigning <key> index 1.

  " Read the details of that particular PO Header from EKKO
  select single vbeln auart audat kunnr vkorg vtweg spart netwr waerk
    into corresponding fields of <fs_sohdr>
    from vbak
   where vbeln = <key>-value.

  if sy-subrc <> 0.
*   throw exception
  endif.


endmethod.
```

14. Save the method.
15. Activate class Z_SALESORDER_HDR_DATA_XX (select all class related objects if not selected).
16. Repeat steps 1 – 15 for class Z_SALESORDER_ITEM_DATA_XX in order to implement the GET_ENTITYSET and GET_ENTITY methods using the following code provided.

Types

```
  types:
    begin of s_vbap,
      vbeln   type vbap-vbeln,
      posnr   type vbap-posnr,
      matnr   type vbap-matnr,
      arktx   type vbap-arktx,
      werks   type vbap-werks,
      kwmeng  type vbap-kwmeng,
      vrkme   type vbap-vrkme,
      netwr   type vbap-netwr,
    end of s_vbap .

  types: t_vbap type standard table of s_vbap with default key.
```

GET_ENTITYSET

```
method /IWBEP/IF_MGW_APPL_SRV_RUNTIME~GET_ENTITYSET.

  data:
    ls_vbap    type s_vbap,                              " Structure for one SO
Item (ABAP fields)
    lt_vbap    type standard table of s_vbap,            " Table of SO Item
(ABAP fields)
    ls_soitem  type z_salesorder_model_xx=>s_so_item,    " Structure for one SO
Item (service fields)
    lt_soitems type z_salesorder_model_xx=>t_so_item,    " Table of SO Item
(service fields)
    lv_id      type vbap-vbeln,
    lv_kline type /iwbep/s_mgw_name_value_pair,
    lv_max     type int4.

  field-symbols:
    <fs_key>     type /iwbep/s_mgw_name_value_pair,
    <fs_items>   type z_salesorder_model_xx=>t_so_item.

* Limit number of items returned to constant value.  $top value will override
if provided.
  lv_max = 1000.

* Create the output data structure and assign its fields to
* the field symbol
  create data er_entityset type z_salesorder_model_xx=>t_so_item.
  assign er_entityset->* to <fs_items>.

* Is there a navigation path that needs to be followed?
  if it_navigation_path is initial.
    " No, so in this case, we are simply obtaining a
    " list of all available SO Headers based on criteria
    if is_paging-top is not initial.
      lv_max = is_paging-top.
    endif.

    select vbeln posnr matnr arktx werks kwmeng vrkme netwr
      into corresponding fields of table lt_vbap
      from vbap
     up to lv_max rows.
```

```
  else.
     " Yes, so now we need to read only items for a particular PO
     " based on underlying SO ID.  SO ID should be in it_key_tab.
     read table it_key_tab assigning <fs_key> index 1.

     select vbeln posnr matnr arktx werks kwmeng vrkme netwr
       into corresponding fields of table lt_vbap
       from vbap
      where vbeln = <fs_key>-value.
  endif.

  loop at lt_vbap into ls_vbap.
    ls_soitem-OrderId      = ls_vbap-vbeln.
    ls_soitem-Item         = ls_vbap-posnr.
    ls_soitem-Material     = ls_vbap-matnr.
    ls_soitem-Description  = ls_vbap-arktx.
    ls_soitem-Plant        = ls_vbap-werks.
    ls_soitem-Quantity     = ls_vbap-kwmeng.
    ls_soitem-UoM          = ls_vbap-vrkme.
    ls_soitem-Value        = ls_vbap-netwr.

    append ls_soitem to <fs_items>.
  endloop.


endmethod.
```

GET_ENTITY

```
method /IWBEP/IF_MGW_APPL_SRV_RUNTIME~GET_ENTITY.

  data:
    ls_item  type z_salesorder_model_xx=>s_so_item,
    lv_vbeln type vbap-vbeln,
    lv_posnr type vbap-posnr,
    lv_kline type /iwbep/s_mgw_name_value_pair,
    lv_lines type i.

  field-symbols:
    <fs_soitem>  type s_vbap.

* Create the output data structure and assign its fields to
* the field symbol
  create data er_entity type z_salesorder_model_xx=>s_so_item.
  assign er_entity->* to <fs_soitem>.

  " Have we been passed any key values?
  if it_key_tab is not initial.
    " Yup, how many key values are there?
    describe table it_key_tab lines lv_lines.

    " 2 key values are needed to locate a unique Item
    if lv_lines eq 2.
      read table it_key_tab into lv_kline with key name = 'OrderId'.
      if sy-subrc = 0.
        lv_vbeln = lv_kline-value.
      endif.

      read table it_key_tab into lv_kline with key name = 'Item'.
      if sy-subrc = 0.
        lv_posnr = lv_kline-value.
      endif.
    endif.
  endif.
```

```
  " Read the details of that particular SO Item from EKPO
  select single vbeln posnr matnr arktx werks kwmeng vrkme netwr
    into corresponding fields of <fs_soitem>
    from vbap
   where vbeln = lv_vbeln and
         posnr = lv_posnr.

  if sy-subrc <> 0.
*   throw exception
  endif.


endmethod.
```

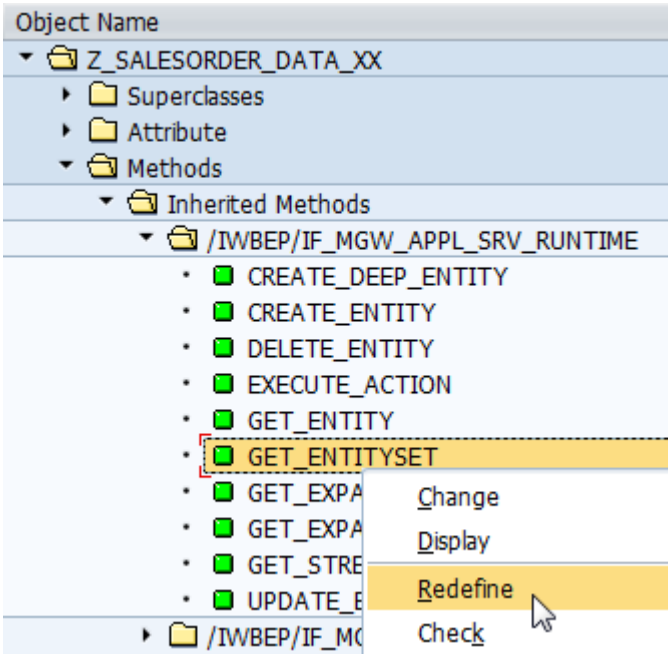
17. Save and activate class Z_SALESORDER_ITEM_DATA_XX (select all class related objects if not selected).

    Now the entity specific data runtime classes have been implemented for our purposes.

4.2.4 Implement the Data Runtime Class

Now we will go back and finish implementing the main data runtime class. Basically, the entity requested is determined and processing is directed to an entity specific implementation.

1. In the ABAP Development Workbench (transaction SE80), display class Z_SALESORDER_DATA_XX.
2. Now, expand the class, find method GET_ENTITYSET.
3. Right-click on it and select *Redefine*.



4. Implement the method using the following code:

```
method /IWBEP/IF_MGW_APPL_SRV_RUNTIME~GET_ENTITYSET.
* Reference the specific data provider class for each entity set
  data:
    lo_so_hdr   type ref to z_salesorder_hdr_data_xx,
    lo_so_item  type ref to z_salesorder_item_data_xx.

* What entity set has been requested?
* Be careful! The entity set names are case-sensitive!
  case iv_entity_name.
```

```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*   SOHeaders
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
    when 'SOHeader'.
      " Instantiate the data provider object for this entity set and the
export
      " data structure.
      " Here, we just happen to know that the correct data type can be found
in
      " the associated Model Provider class; however, such a reference is only
      " a convenience, not a requirement
      create:
        object lo_so_hdr,
        data er_entityset type z_salesorder_model_xx=>s_so_hdr.

      lo_so_hdr->/iwbep/if_mgw_appl_srv_runtime~get_entityset(
        exporting iv_entity_name           = iv_entity_name
                  iv_source_name           = iv_source_name
                  iv_entity_set_name       = iv_entity_set_name
                  it_filter_select_options = it_filter_select_options
                  it_order                 = it_order
                  is_paging                = is_paging
                  it_navigation_path       = it_navigation_path
                  it_key_tab               = it_key_tab
                  iv_filter_string         = iv_filter_string
                  iv_search_string         = iv_search_string
        importing er_entityset             = er_entityset ).

* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*   SOItems
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
    when 'SOItem'.
      " Instantiate the data provider object for this entity set and the
export
      " data structure.
      create:
        object lo_so_item,
        data er_entityset type z_salesorder_model_xx=>s_so_item.

      lo_so_item->/iwbep/if_mgw_appl_srv_runtime~get_entityset(
        exporting iv_entity_name           = iv_entity_name
                  iv_source_name           = iv_source_name
                  iv_entity_set_name       = iv_entity_set_name
                  it_filter_select_options = it_filter_select_options
                  it_order                 = it_order
                  is_paging                = is_paging
                  it_navigation_path       = it_navigation_path
                  it_key_tab               = it_key_tab
                  iv_filter_string         = iv_filter_string
                  iv_search_string         = iv_search_string
        importing er_entityset             = er_entityset ).

* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*   Something else...
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
    when others.
      " Add handler here
  endcase.


endmethod.
```

5. Save the method.
6. Now redefine method GET_ENTITY using the following code:

```
method /IWBEP/IF_MGW_APPL_SRV_RUNTIME~GET_ENTITY.

  data:
    lo_sohdr   type ref to z_salesorder_hdr_data_xx,
    lo_soitem  type ref to z_salesorder_item_data_xx.

* Implement call-throughs to the specific entity type data provider classes.
* What entity has been requested?  Be careful! Entity names are case
sensitive!
  case iv_entity_name.
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*   Sales Order Header
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
    when 'SOHeader'.
      create:
        object lo_sohdr,
        data   er_entity type z_salesorder_model_xx=>s_so_hdr.
      lo_sohdr->/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
        importing er_entity          = er_entity ).

* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*   Sales Order Item
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
    when 'SOItem'.
      create:
        object lo_soitem,
        data   er_entity type z_salesorder_model_xx=>s_so_item.
      lo_soitem->/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
        importing er_entity          = er_entity ).

    when others.
      " Add some other entity handler here
  endcase.

endmethod.
```

7. Save the method.

4.2.5 Implement CREATE_DEEP_ENTITY method for Deep Insert

The CREATE_DEEP_ENTITY method is used for the CREATE operation of an entity with deep or nested data provided in an inline format according to http://www.odata.org/developers/protocols/operations#CreatingnewEntries. Every deep insert request has to be handled by the implementation which determines whether it can fulfill the current deep insert request or not by a given expand expression.

The method signature is similar to the CREATE_ENTITY method except that it has an additional parameter – IO_EXPAND. The SAP NetWeaver Gateway framework resolves the inline data and translates it to an expand expression which is passed to the method via IO_EXPAND. It can be used to validate whether the current request including the inline data matches a given expand expression or to retrieve purely the expand string. If the expand expression matches, the data can be accessed via IO_DATA_PROVIDER similar to CREATE_ENTITY. The ES_DATA parameter expects a nested structure which contains the components for the inline entries.

The final response sent back to the requestor only contains the newly created sales order header entry without the inline item entries.

In our sales order case, sales order header data would be provided as the top level entry along with sales order item entries nested inline in the request. A corresponding nested structure representing the full sales order (header and item) should be defined in the implementation to accommodate the deep data structure that will be provided in the sales order creation request.

1. Still within class Z_SALESORDER_DATA_XX, redefine method CREATE_DEEP_ENTITY using the following code.

```
method /IWBEP/IF_MGW_APPL_SRV_RUNTIME~CREATE_DEEP_ENTITY.

  types: ty_t_soitem type standard table of z_salesorder_model_xx=>s_so_item
with default key.

* Represents full Sales Order structure - header with one of more items
  types: begin of ty_s_so.
           include type z_salesorder_model_xx=>s_so_hdr.
           types: SOItems type ty_t_soitem,
         end of ty_s_so.

  types: cx_mgw_busi_exception TYPE REF TO /iwbep/cx_mgw_busi_exception.

  data: ls_so              type ty_s_so,
        ls_item            type z_salesorder_model_xx=>s_so_item,
        lv_compare_result  type
/iwbep/if_mgw_odata_expand=>ty_e_compare_result.

  data: lv_soid     type BAPIVBELN-VBELN,
        ls_sohdr    type BAPISDHD1,
        ls_soitem   type BAPISDITM,
        lt_soitem   type standard table of BAPISDITM,
        ls_partner  type BAPIPARNR,
        lt_partner  type standard table of BAPIPARNR,
        lt_return   type bapirettab,
        lr_return   TYPE REF TO bapiret2,
        lx_busi_exc TYPE cx_mgw_busi_exception,
        lo_meco     TYPE REF TO /iwbep/if_message_container.

  constants: lc_soitems TYPE string VALUE 'SOItems'.

*   Validate whether the current request including the inline SOItem data
matches
    lv_compare_result = io_expand->compare_to( lc_soitems ).

*   Upon match, access data from IO_DATA_PROVIDER
    if lv_compare_result EQ /iwbep/if_mgw_odata_expand=>gcs_compare_result-
match_equals.
      io_data_provider->read_entry_data( IMPORTING es_data = ls_so ).

*     Move data into BAPI structure
      " Header data
      ls_sohdr-doc_type   = ls_so-DocumentType.
      ls_sohdr-sales_org  = ls_so-SalesOrg.
      ls_sohdr-distr_chan = ls_so-DistChannel.
      ls_sohdr-division   = ls_so-Division.

      " Item data
      loop at ls_so-SOItems into ls_item.
        ls_soitem-itm_number  = ls_item-Item.
        ls_soitem-material    = ls_item-Material.
        ls_soitem-plant       = ls_item-Plant.
        ls_soitem-target_qty  = ls_item-Quantity.
```

```
        append ls_soitem to lt_soitem.
      endloop.

      " Fill Partner table with one entry - Sold-to Party
      ls_partner-partn_role = 'AG'.
      ls_partner-partn_numb = ls_so-CustomerId.

      append ls_partner to lt_partner.

      CALL FUNCTION 'BAPI_SALESORDER_CREATEFROMDAT2'
        EXPORTING
*         SALESDOCUMENTIN               =
          ORDER_HEADER_IN               = ls_sohdr
*         ORDER_HEADER_INX              =
*         SENDER                        =
*         BINARY_RELATIONSHIPTYPE       =
*         INT_NUMBER_ASSIGNMENT         =
*         BEHAVE_WHEN_ERROR             =
*         LOGIC_SWITCH                  =
*         TESTRUN                       =
*         CONVERT                       = ' '
        IMPORTING
          SALESDOCUMENT                 = lv_soid
        TABLES
          RETURN                        = lt_return
          ORDER_ITEMS_IN                = lt_soitem
*         ORDER_ITEMS_INX               =
          ORDER_PARTNERS                = lt_partner
*         ORDER_SCHEDULES_IN            =
*         ORDER_SCHEDULES_INX           =
*         ORDER_CONDITIONS_IN           =
*         ORDER_CONDITIONS_INX          =
*         ORDER_CFGS_REF                =
*         ORDER_CFGS_INST               =
*         ORDER_CFGS_PART_OF            =
*         ORDER_CFGS_VALUE              =
*         ORDER_CFGS_BLOB               =
*         ORDER_CFGS_VK                 =
*         ORDER_CFGS_REFINST            =
*         ORDER_CCARD                   =
*         ORDER_TEXT                    =
*         ORDER_KEYS                    =
*         EXTENSIONIN                   =
*         PARTNERADDRESSES              =
                .

      READ TABLE lt_return INDEX 1 REFERENCE INTO lr_return.

      IF lr_return->*-type NE 'S'.

        lo_meco = mo_context->get_message_container( ).
        lo_meco->add_messages_from_bapi(
              it_bapi_messages          = lt_return
              iv_determine_leading_msg  =
/iwbep/if_message_container=>gcs_leading_msg_search_option-first
             ).

        CREATE OBJECT lx_busi_exc
          EXPORTING
            message_container = lo_meco.

        RAISE EXCEPTION lx_busi_exc.
      ELSE.
        COMMIT WORK.
         ls_so-OrderID  = lv_soid.
```

```
      ENDIF.

      copy_data_to_ref(
        EXPORTING
          is_data = ls_so
        CHANGING
          cr_data = er_deep_entity
      ).
    ENDIF.


endmethod.
```

2. Save the method.
3. Activate class Z_SALESORDER_DATA_XX (select all class related objects if not selected).

Relevant coding is now complete. We just need to assign the model class to the runtime class via configuration and then register the service on Gateway.

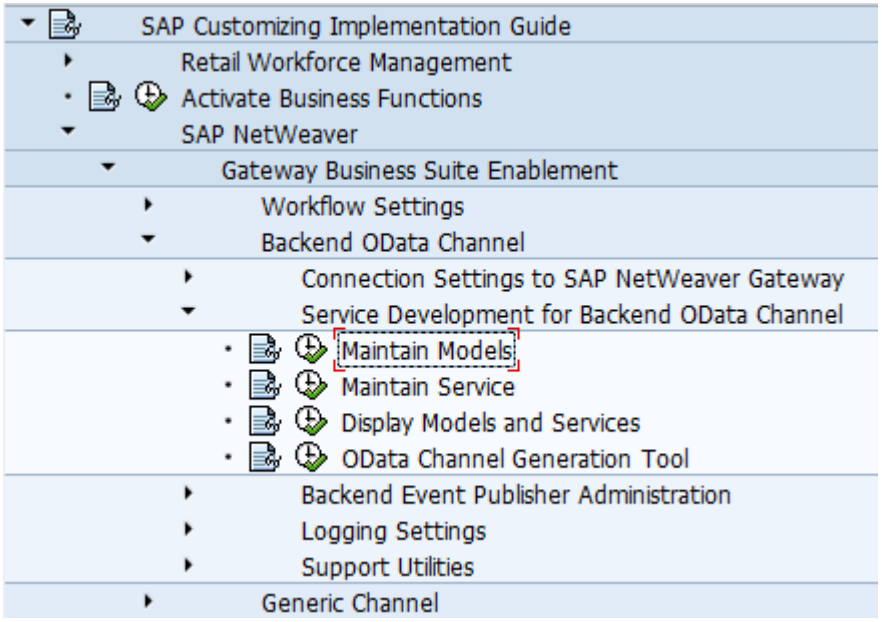
4.3 Configure the Runtime Data and Model Provider Classes to Act as a Gateway Service

The configuration that now needs to be performed is to associate the Model Provider class (`Z_SALESORDER_MODEL_XX`) with the runtime Data Provider class (`Z_SALESORDER_DATA_XX`). This is done by creating two wrapper objects.

- The Model Provider class is wrapped in a Technical Model.
- The Runtime Data Provider class is wrapped in a Service Group.

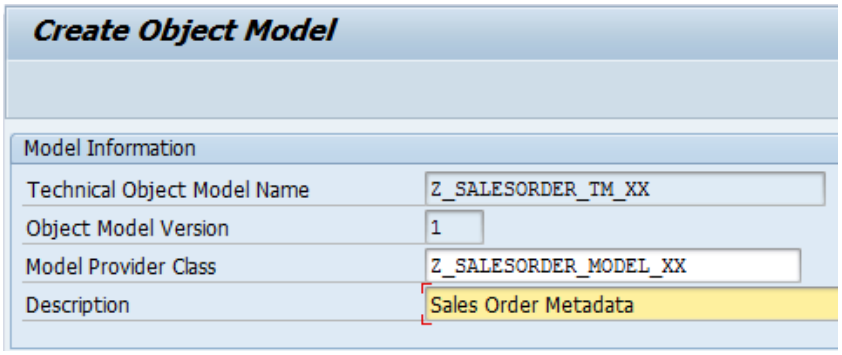
The Technical Model is then assigned to the service group.

1. Start transaction `SPRO` on your SAP backend system and enter the SAP Reference IMG.
2. Follow the path *SAP NetWeaver → Gateway Business Suite Enablement → Backend OData Channel → Service Development for Backend OData Channel*:

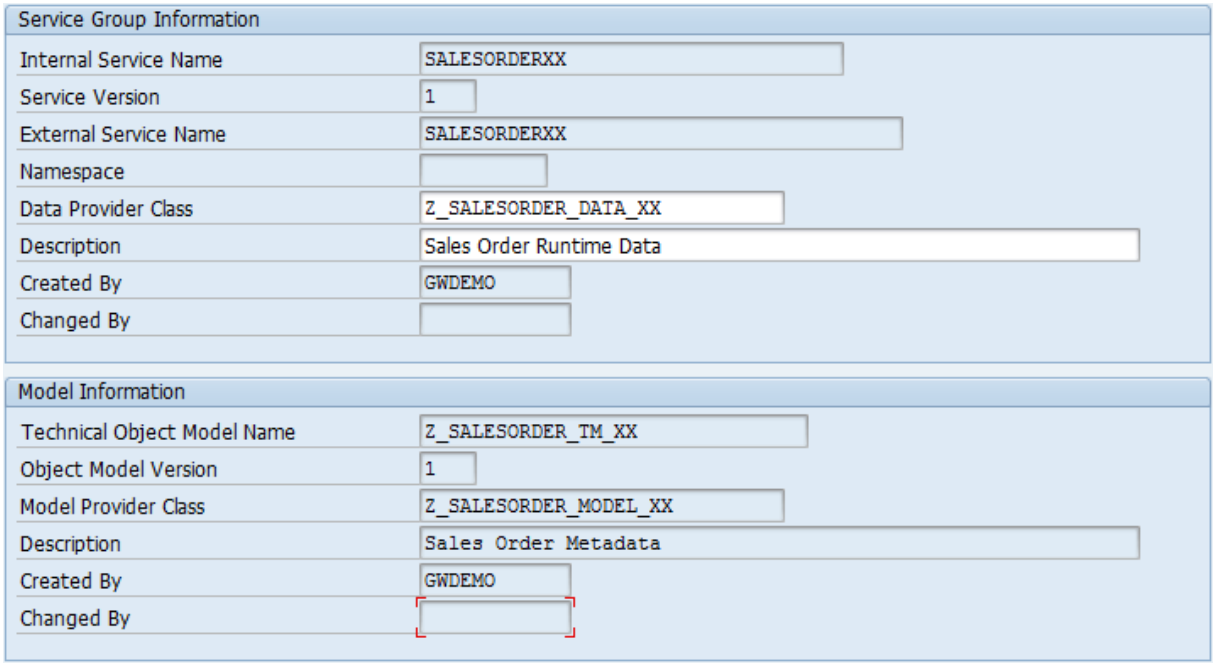


3. Start the *Maintain Models* activity.
4. Here you will need to enter the Technical Model Name and a version number. At the moment, the version number is always 1. Enter a Technical Model Name of `Z_SALESORDER_TM_XX` and press the *Create* button.

5. On the following screen, enter the name of the metadata class that is being wrapped and give it some meaningful description. In our case, the metadata class is `Z_SALESORDER_MODEL_XX`.



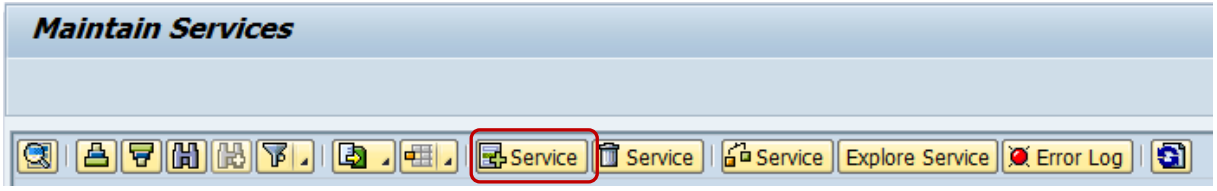
6. Save the changes, select an appropriate package, and back out to the IMG tree structure.
7. Now run the *Maintain Service* activity. This is where the Runtime Data Provider class is wrapped using an Internal Name. There is a quirk in the configuration transaction that means whatever name you enter as the Internal Service Name, is assigned to the External Service Name. Since the External Service Name is the name that will appear in the URL that runs this service, we should choose a name that will be meaningful to the end user.
8. Enter `SalesOrderXX` as the Internal Name and 1 as the version number and press the *Create* button.
9. On the following screen, enter the name of the Runtime Data Provider class `Z_SALESORDER_DATA_XX` and a meaningful description.
10. You must now save your configuration otherwise you will get an error message on the following pop-up screen.
11. Since we have already created a Technical Model to wrap our metadata class, we need to press the *Assign Model* button, NOT the *Create Model* button.
12. On the following pop-up, enter the name of the Technical Model we created above (`Z_SALESORDER_TM_XX`), the version number (which at the moment, is always 1) and press the save button.
13. Now press Save, and your completed configuration should look like this:



4.4 Expose the Gateway Service to the Outside World

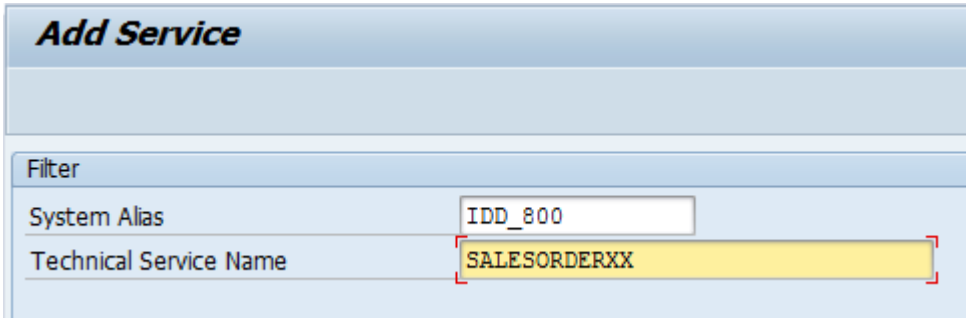
Having created the configuration that connects the Model Provider and Runtime Data Provider classes to form a Gateway Service, it is now necessary to expose that Gateway Service to the outside world.

1. Log on to the Gateway system (i.e. the ABAP system containing the `GW_CORE` component).
2. Run transaction SPRO and navigate to *SAP NetWeaver → Gateway → OData Channel Administration → General Settings → Maintain Services* or go directly to the transaction using transaction code **`/IWFND/MAINT_SERVICE.`**
3. First we need to add our service to this Gateway system. Start by clicking on the "Add Service" button:

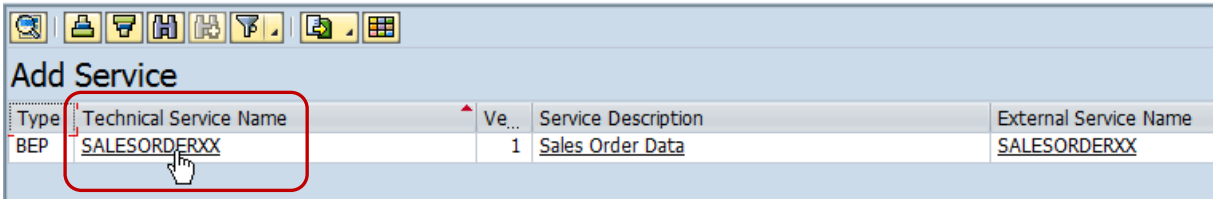


4. Here, you need to know the system alias of the ABAP system in which you developed and configured the Gateway Service.

   Enter your System Alias and also enter `SALESORDERXX` for Technical Service Name to minimize the list of services returned.

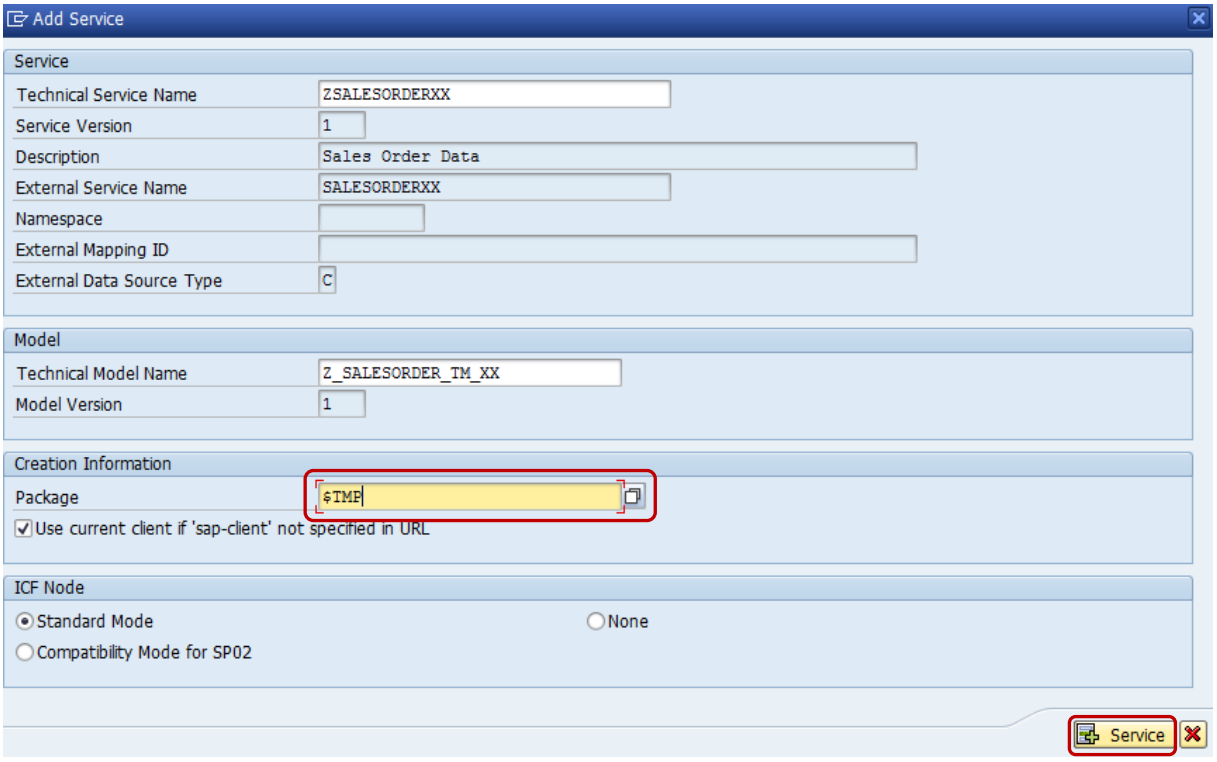


5. After pressing <Enter>, our exact service should be the only service displayed.

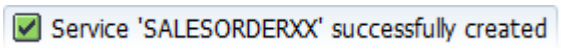


6. Click on the *SALESORDERXX* link under the Technical Service Name column.

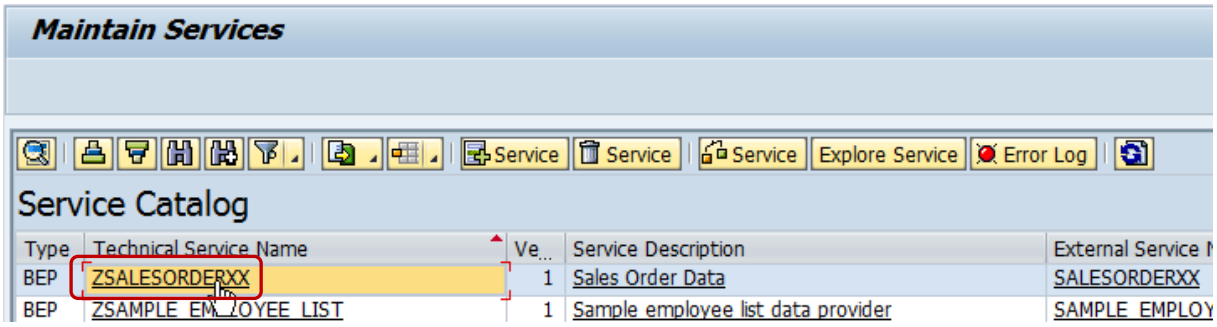
7. If necessary, enter the name of the relevant Package and then click on the Add Service button on the lower right.



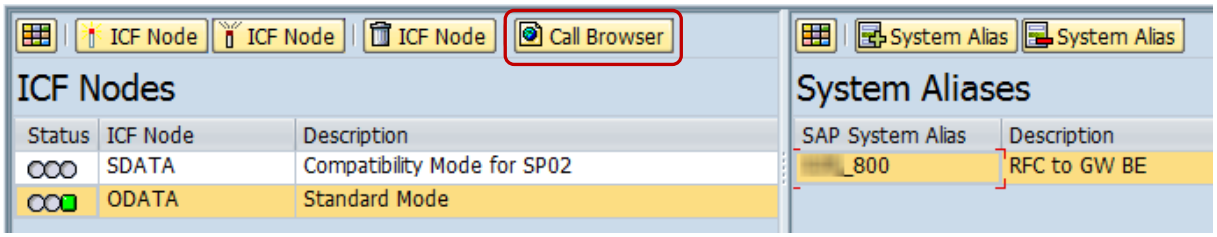
8. A confirmation message should appear on the bottom left upon successful addition of the service:



9. Go back to the Maintain Services screen. Look for your newly added service and click on *ZSALESORDERXX*.



10. You should see the service classified under "Standard Mode" and "ODATA" for ICF Node and a system alias should already be assigned.



Now the service is ready to be consumed by the outside world.

4.5 Test the Gateway Service

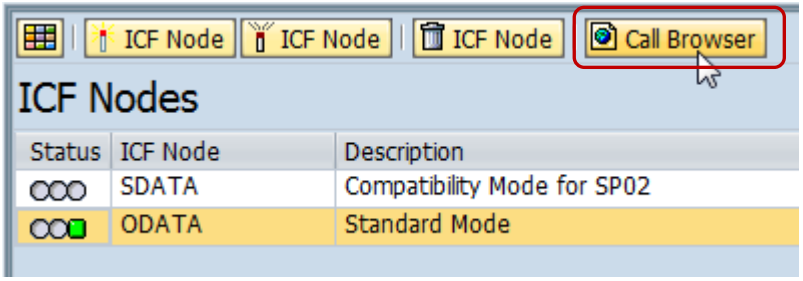
With the required model and runtime classes created, implemented, registered, and exposed to the outside world, testing of the service can now commence. Starting with SP03, a REST enabled client, such as Firefox REST client or WFetch, is needed for testing all the operations (e.g. Query, Read, Create, Update, Delete). A browser no longer suffices in displaying the XML response, even for GET related operations such as Query or Read.

4.5.1 View the Service Document and Metadata

The Service Document lists all the available feeds or collections so that clients can discover what they are and how to access them.

The service metadata is basically a description of the underlying data model and a direct reflection of the model class implementation of the DEFINE method.

1. With your service selected on the *Maintain Services* page, select the *Call Browser* button to launch a browser to display the service document.



The URL for the service document is:

```
http://<GWHost>:<port>/sap/opu/odata/sap/SALESORDERXX/
```

```
<?xml version="1.0" encoding="utf-8" ?>
- <app:service
    xml:base="http://[redacted]/sap/opu/odata/sap/SALESORDERXX/"
    xmlns:app="http://www.w3.org/2007/app"
    xmlns:atom="http://www.w3.org/2005/Atom"
    xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
    xmlns:sap="http://www.sap.com/Protocols/SAPData">
  - <app:workspace>
      <atom:title>Data</atom:title>
    - <app:collection sap:content-version="1" href="SOHeaders">
        <atom:title>SOHeaders</atom:title>
        <sap:member-title>SOHeader</sap:member-title>
      </app:collection>
    - <app:collection sap:content-version="1" href="SOItems">
        <atom:title>SOItems</atom:title>
        <sap:member-title>SOItem</sap:member-title>
      </app:collection>
    </app:workspace>
  </app:service>
```

2. Add `$metadata` to the URL for the Service Document to display the Service Metadata Document.

```
http://<GWHost>:<port>/sap/opu/odata/sap/SALESORDERXX/$metadata
```

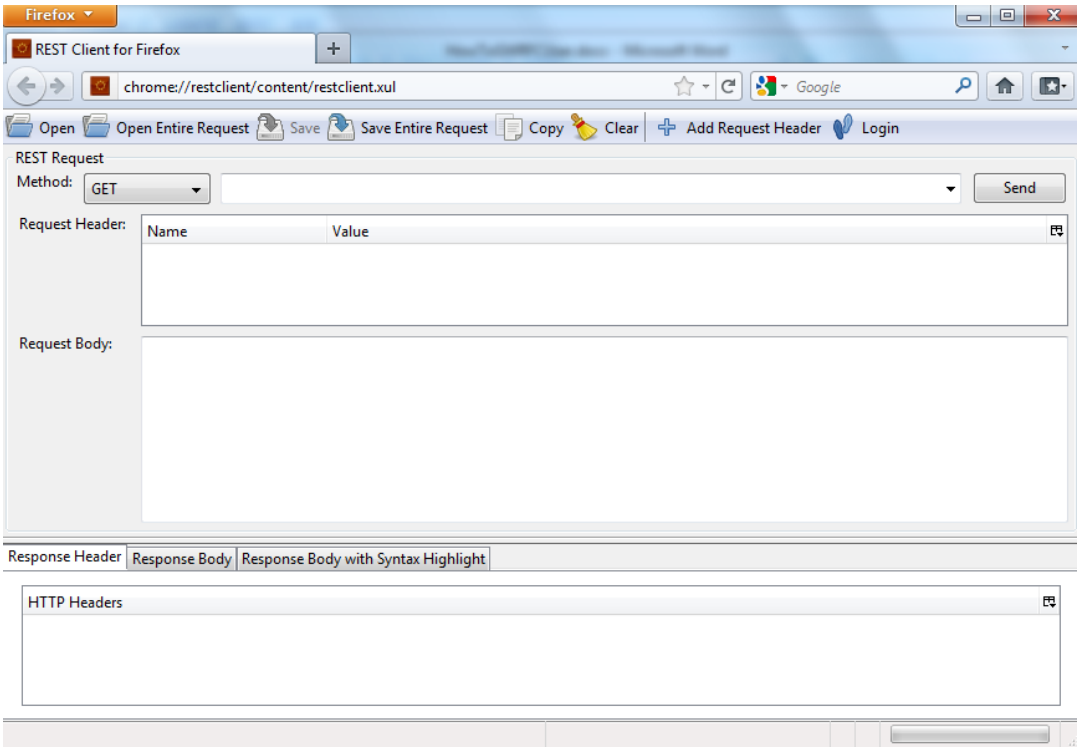
```
<?xml version="1.0" encoding="utf-8" ?>
- <edmx:Edmx Version="1.0" xmlns:edmx="http://schemas.microsoft.com/ado/2007/06/edmx"
    xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
    xmlns:sap="http://www.sap.com/Protocols/SAPData">
  - <edmx:DataServices m:DataServiceVersion="2.0">
    - <Schema Namespace="SALESORDERXX" xmlns="http://schemas.microsoft.com/ado/2008/09/edm">
      - <EntityType Name="SOHeader" sap:content-version="1">
        - <Key>
            <PropertyRef Name="OrderId" />
          </Key>
          <Property Name="OrderId" Type="Edm.String" Nullable="false" MaxLength="10" sap:label="Sales Document" />
          <Property Name="DocumentType" Type="Edm.String" MaxLength="4" sap:label="Sales Doc. Type" />
          <Property Name="DocumentDate" Type="Edm.DateTime" sap:label="Document Date" />
          <Property Name="CustomerId" Type="Edm.String" MaxLength="10" sap:label="Sold-to party" />
          <Property Name="SalesOrg" Type="Edm.String" MaxLength="4" sap:label="Sales Org." />
          <Property Name="DistChannel" Type="Edm.String" MaxLength="2" sap:label="Distr. Channel" />
          <Property Name="Division" Type="Edm.String" MaxLength="2" sap:label="Division" />
          <Property Name="OrderValue" Type="Edm.Decimal" Precision="15" Scale="2" sap:label="Net value" />
          <Property Name="Currency" Type="Edm.String" MaxLength="5" sap:label="Doc. Currency" sap:semantics="currency-
            code" />
          <NavigationProperty Name="SOItems" Relationship="SALESORDERXX.SOHeader_SOItems"
            FromRole="FromRole_SOHeader_SOItems" ToRole="ToRole_SOHeader_SOItems" />
        </EntityType>
      - <EntityType Name="SOItem" sap:content-version="1">
        - <Key>
            <PropertyRef Name="OrderId" />
            <PropertyRef Name="Item" />
          </Key>
          <Property Name="OrderId" Type="Edm.String" Nullable="false" MaxLength="10" sap:label="Sales Document" />
          <Property Name="Item" Type="Edm.String" Nullable="false" MaxLength="6" sap:label="Item" />
          <Property Name="Material" Type="Edm.String" MaxLength="18" sap:label="Material" />
          <Property Name="Description" Type="Edm.String" MaxLength="40" sap:label="Description" />
          <Property Name="Plant" Type="Edm.String" MaxLength="4" sap:label="Plant" />
          <Property Name="Quantity" Type="Edm.Decimal" Precision="15" Scale="3" sap:label="Order quantity" />
          <Property Name="UoM" Type="Edm.String" MaxLength="3" sap:label="Sales unit" sap:semantics="unit-of-measure" />
          <Property Name="Value" Type="Edm.Decimal" Precision="15" Scale="2" sap:label="Net value" />
          <NavigationProperty Name="SOHeader" Relationship="SALESORDERXX.SOItem_SOHeader"
            FromRole="FromRole_SOItem_SOHeader" ToRole="ToRole_SOItem_SOHeader" />
        </EntityType>
      - <Association Name="SOItem_SOHeader" sap:content-version="1">
```

4.5.2 Test the Query Operation

Testing the Query operation is just a matter of replacing `$metadata` with a collection name as shown in the Service Document. In our case the available collections are SOHeaders (for Sales Order Headers) and SOItems (Sales Order Items).

In order to test the service operations, a simple HTTP client tool is required. In this guide, Firefox (with REST add-on) will be used as an example, but there are other tools available such as WFetch from Microsoft. Either of these can be downloaded for free. The following steps assume that have the Firefox browser and have installed the REST Client add-on.

3. Open the Firefox REST client.



4. Use the following URL to execute the Query operation.

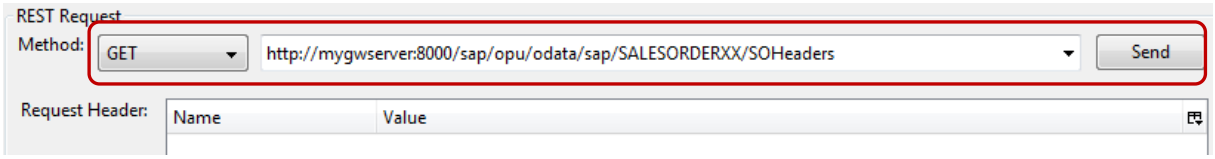
   Notice that it's similar to the URL to access the metadata – just a matter of replacing "$metadata" with "SOHeaders".

   ```
   http://<GWHost>:<port>/sap/opu/odata/sap/SALESORDERXX/SOHeaders
   ```

5. User credentials can be supplied in advanced by pressing the *Login* button 

or can be supplied when prompted at execution time.

6. Set the Method to *GET* and press the *Send* button.



   Note that in the backend implementation, we have hardcoded a limit to the number of query results to a maximum of 100 for simplicity. However, the query string option $top can be used to override this default either to restrict the results or to increase the number of results as well (e.g. **`$top=10`** or **`$top=1000`**).

7. Try this for Sales Order Items as well.

   ```
   http://<GWHost>:<port>/sap/opu/odata/sap/SALESORDERXX/SOItems
   ```

4.5.3 Test the Read Operation

The read operation will return the details of one instance from the collection – so in our case, one specific Sales Order Header or one specific Sales Order Item. Testing the Read operation is just a matter of building on the Query URL and adding one or more key fields. However, OData makes this even easier by providing the relative links to each specific entity within the Query results.

1. From one of the entries returned from the Query, find a link to read the details of the specific entry.

```
- <atom:entry>
    <atom:category term="SALESORDERXX.SOHeader" scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme"/>
  - <atom:content type="application/xml">
    - <m:properties>
        <d:OrderId>0000004969</d:OrderId>
        <d:DocumentType>TA</d:DocumentType>
        <d:DocumentDate>1997-01-02T00:00:00</d:DocumentDate>
        <d:CustomerId>0000001390</d:CustomerId>
        <d:SalesOrg>1000</d:SalesOrg>
        <d:DistChannel>10</d:DistChannel>
        <d:Division>00</d:Division>
        <d:OrderValue>5500.00</d:OrderValue>
        <d:Currency>DEM</d:Currency>
      </m:properties>
    </atom:content>
  - <atom:id>
      http://vmw3815.wdf.sap.corp:50009/sap/opu/odata/sap/SALESORDERXX/SOHeaders('0000004969')
    </atom:id>
    <atom:link href= "SOHeaders('0000004969')" rel="edit" title="SOHeader"/>
    <atom:link href= "SOHeaders('0000004969')/SOItems" rel="http://schemas.microsoft.com/ado/2007/08/dataservices/related/SOItems"
    type=feed" title="SOItems"/>
    <atom:title type="text">SOHeaders('0000004969')</atom:title>
    <atom:updated>2012-02-13T22:38:31Z</atom:updated>
  </atom:entry>
```

2. So the URL to call the Read operation for this entry is:

   `http://<GWHost>:<port>/sap/opu/odata/sap/SALESORDERXX/`**`SOHeaders(0000004969)`**

```
- <atom:entry xml:base="http://[redacted]/sap/opu/odata/sap/SALESORDERXX/">
    <atom:category term="SALESORDERXX.SOHeader" scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme"/>
  - <atom:content type="application/xml">
    - <m:properties>
        <d:OrderId>0000005969</d:OrderId>
        <d:DocumentType>TA</d:DocumentType>
        <d:DocumentDate>1999-08-10T00:00:00</d:DocumentDate>
        <d:CustomerId>0000001033</d:CustomerId>
        <d:SalesOrg>1000</d:SalesOrg>
        <d:DistChannel>12</d:DistChannel>
        <d:Division>00</d:Division>
        <d:OrderValue>89960.00</d:OrderValue>
        <d:Currency>DEM</d:Currency>
      </m:properties>
    </atom:content>
  - <atom:id>
      http://vmw3815.wdf.sap.corp:50009/sap/opu/odata/sap/SALESORDERXX/SOHeaders('0000005969')
    </atom:id>
    <atom:link href= "SOHeaders('0000005969')" rel="edit" title="SOHeader"/>
    <atom:link href= "SOHeaders('0000005969')/SOItems" rel="http://schemas.microsoft.com/ado/2007/08/dataservices/related/SOItems"
    type=feed" title="SOItems"/>
    <atom:title type="text">SOHeaders('0000005969')</atom:title>
    <atom:updated>2012-02-13T22:47:17Z</atom:updated>
  </atom:entry>
```

3. You will notice another link (highlighted above) that allows you to navigate to the corresponding Item entries. In that case, the URL would be:

   `http://<GWHost>:<port>/sap/opu/odata/sap/SALESORDERXX/`**`SOHeaders(0000004969)/SOItems`**

4.5.4 Test the Expanded Query and Read Operations

One way to get an idea of what the request payload for a deep insert request should consist of is to call the Read operation with the $expand OData system query option (OData – $expand). This allows, for example, to Read a full sales order – i.e., a sales order header along with all of its corresponding item entries in an inline, deep format.

This is possible due to the associations and navigation properties set up in our data model. The one-to-many association between SOHeader and SOItems along with the navigation property, SOItems, allowing for navigation from SOHeader to SOItems, is the basis for reading or creating our sales order header entries "deeply" with inline sales order item entries.

The central runtime interface, /IWBEP/IF_MGW_APPL_SRV_RUNTIME, also provides the GET_EXPANDED_ENTITY and GET_EXPANDED_ENTITYSET methods for reading or querying entity data in a deep/inline format.

Note

SAP NetWeaver Gateway framework provides a default implementation of these methods which can completely handle expand requests in a generic way. However, you do have the option to redefine the methods with your own implementation if necessary (e.g. in case of performance-critical scenarios or if you experience a bad response time).

1. To test getting a full, single sales order (e.g. a sales order header and all its corresponding items), use the following URL:

   `http://<GWHost>:<port>/sap/opu/odata/sap/SALESORDERXX/`**`SOHeaders(0000004969)?$expand=SOItems`**

   Note

   The resulting payload from this read is essentially the same format that will be used for the deep insert request.

2. To test getting all sales order headers, each with all its associated sales order items, use the following URL:

   `http://<GWHost>:<port>/sap/opu/odata/sap/SALESORDERXX`**`/SOHeaders?$expand=SOItems`**

   Tip

   Given the large amount of data potentially returned from this call, the use of the system query option $top is advised in this case.

4.5.5 Test the Create Operation (Deep Insert)

Starting with Gateway 2.0/SP03, Cross-Site Request Forgery (CSRF) protection using *token* exchange and validation is enabled by default for all data modifying requests (e.g. Create, Update, Delete).

This means that a valid CSRF token must first be retrieved using a non-modifying request (e.g. using the GET method). Then it can be sent along with the subsequent modifying request and validated before normal processing continues.

For more information on CSRF protection mechanisms in SAP NetWeaver Gateway, please visit the SAP Help Portal: Cross-Site Request Forgery Protection.

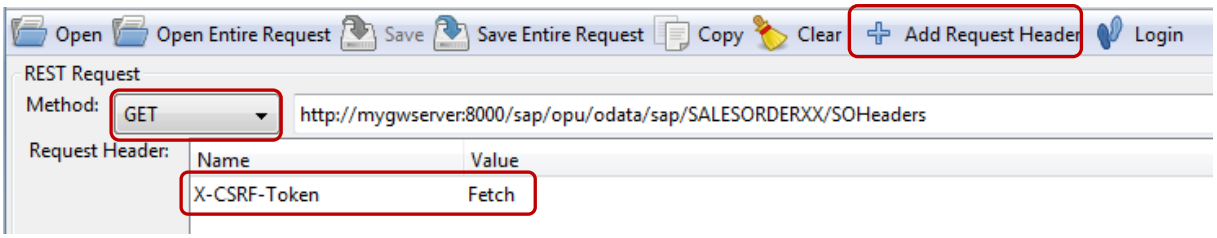
Retrieve the CSRF Token First

1. Open the Firefox REST client.
2. Set the method to GET.
3. Set the URL to the same one used for the QUERY operation:

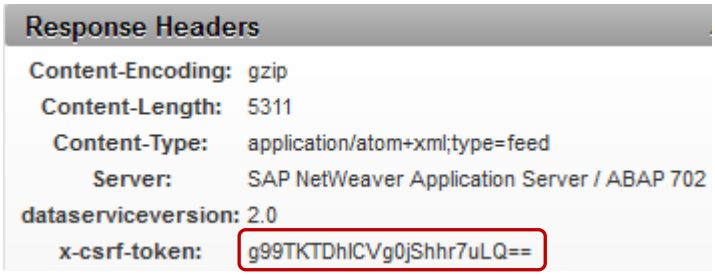
   Example

   `http://mygwserver:8000/sap/opu/odata/sap/ SALESORDERXX/SOHeaders`

4. Click the *Add Request Header* button and add the HTTP header *X-CSRF-Token* with a value of `Fetch` as shown:



5. From the Firefox main menu, choose *Tools* →*Web Developer* →*Web Console* so that you can copy the token from the Firefox window.
6. Click on the *Send* button to execute the query.
7. On the Web Console at the top, click on the link that appears corresponding to the request just executed.
8. In the pop-up window, scroll to Response Headers at the bottom and copy the token value returned.



Close the pop-up window.

Execute the Create Operation

1. Set the method to *POST* and change the value for header *X-CSRF-Token* to the token value that was retrieved in the previous step (Hint: Just double-click on "Fetch" to edit the value).
2. Add a new HTTP header called *Content-Type* with value `application/atom+xml` and keep the URL set to the Query URL.
3. The URL should still be set to the same one used for the QUERY operation:

   `http://<GWHost>:<port>/sap/opu/odata/sap/SALESORDERXX/SOHeaders`
4. Next you need to supply the actual data that will be used to create the user.

   The XML format used is basically the same as the XML result returned when performing a Read operation with the $expand option – for example, to retrieve Sales Order header and its item data together.

   A sample XML dataset is provided below.

   Copy the sample XML below to the Request Body of the Firefox REST client. Note that these are just sample values. They should be replaced with relevant values for your particular system.

```
<?xml version="1.0" encoding="UTF-8"?>
<atom:entry
    xmlns:atom="http://www.w3.org/2005/Atom"
    xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
    xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata">
    <atom:content type="application/xml">
        <m:properties>
            <d:OrderId>0</d:OrderId>
```

```
            <d:DocumentType>TA</d:DocumentType>
            <d:CustomerId>0000001033</d:CustomerId>
            <d:SalesOrg>1000</d:SalesOrg>
            <d:DistChannel>12</d:DistChannel>
            <d:Division>00</d:Division>
            <d:DocumentDate m:null="true" />
            <d:OrderValue m:null="true" />
            <d:Currency m:null="true" />
        </m:properties>
    </atom:content>
    <atom:link
        rel="http://schemas.microsoft.com/ado/2007/08/dataservices/related/SOItems"
        type="application/atom+xml;type=feed"
        title="SALESORDERXX.SOHeader_SOItems">
        <m:inline>
            <atom:feed>
                <atom:entry>
                    <atom:content type="application/xml">
                        <m:properties>
                            <d:OrderId>0</d:OrderId>
                            <d:Item>000010</d:Item>
                            <d:Material>M-05</d:Material>
                            <d:Plant>1200</d:Plant>
                            <d:Quantity m:Type="Edm.Decimal">100.000</d:Quantity>
                            <d:Description m:null="true" />
                            <d:UoM m:null="true" />
                            <d:Value m:null="true" />
                        </m:properties>
                    </atom:content>
                </atom:entry>
                <atom:entry>
                    <atom:content type="application/xml">
                        <m:properties>
                            <d:OrderId>0</d:OrderId>
                            <d:Item>000020</d:Item>
                            <d:Material>M-06</d:Material>
                            <d:Plant>1200</d:Plant>
                            <d:Quantity m:Type="Edm.Decimal">200.000</d:Quantity>
                            <d:Description m:null="true" />
                            <d:UoM m:null="true" />
                            <d:Value m:null="true" />
                        </m:properties>
                    </atom:content>
                </atom:entry>
            </atom:feed>
        </m:inline>
    </atom:link>
</atom:entry>
```

Note

Even though every field in your Gateway data model may not be relevant for the CREATE operation, they must be supplied in the request with either a valid value or with its *null* attribute set to "true".

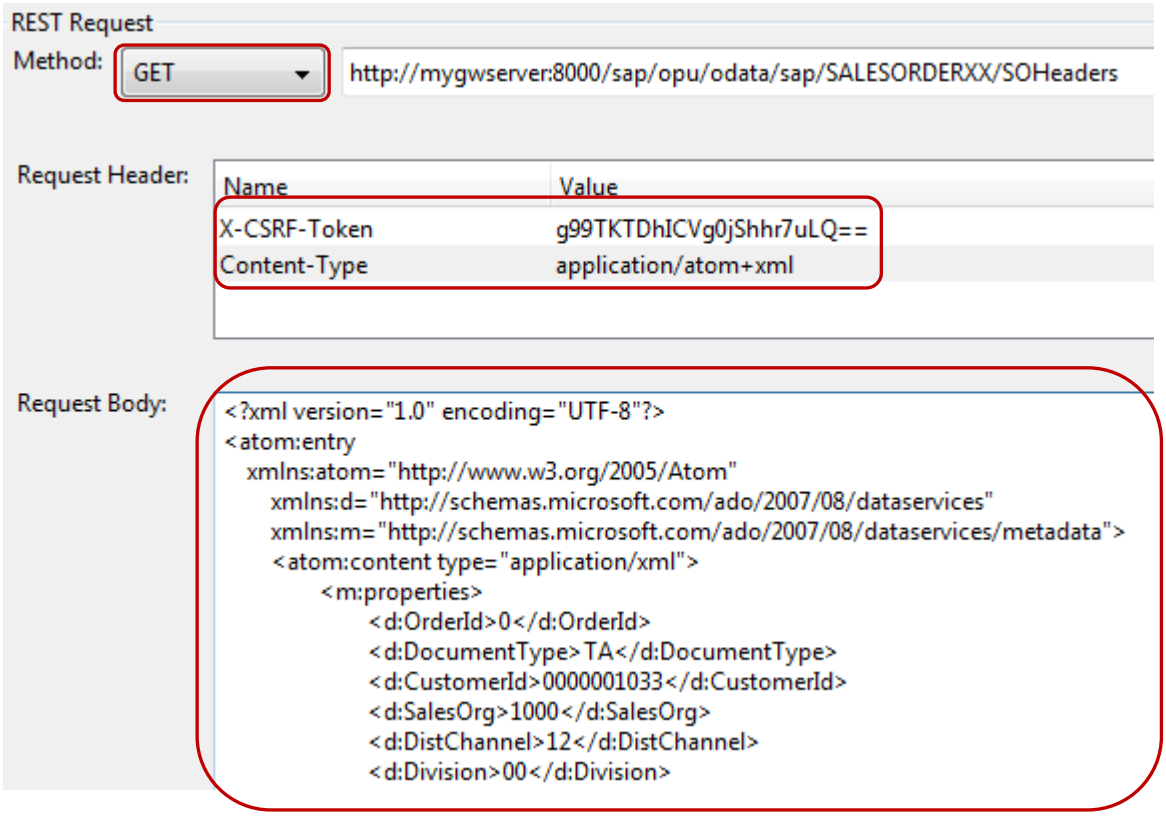
For example, in the sample above, the property "Description" is not needed for the CREATE operation, but it still must be included with its *null* attribute set to "true" (*<d:Description m:null="true" />*). Otherwise, a HTTP 500 error can result with error message "The Data Services Request could not be understood due to malformed syntax".

Note

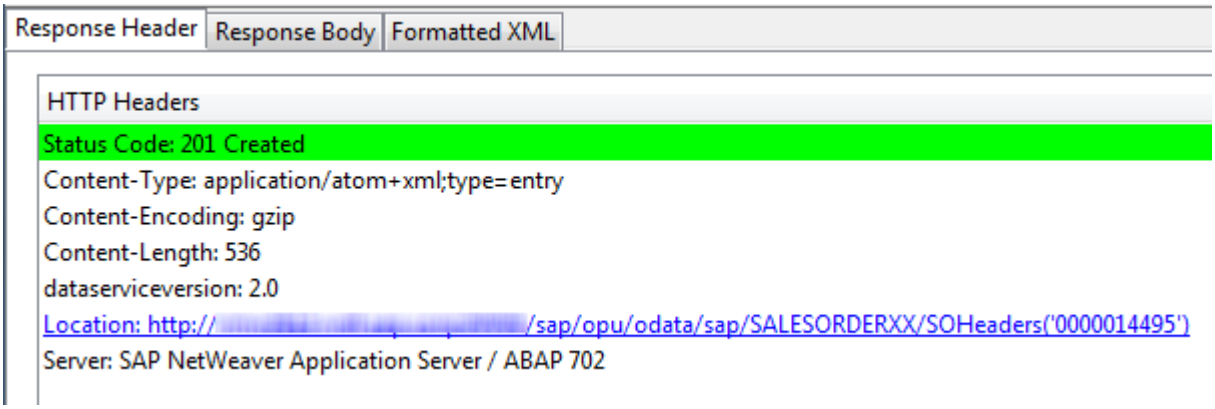
In SP03, for POST requests where keys are generated on the server, a value for the key(s) must be supplied in the request. Setting the null attribute in this case is not allowed.

For example, in the sample above, "OrderId" is included in the request and set with a dummy value. The actual Order ID will be generated by the SAP system.

5. Now that the request is ready, click on the *Send* button.

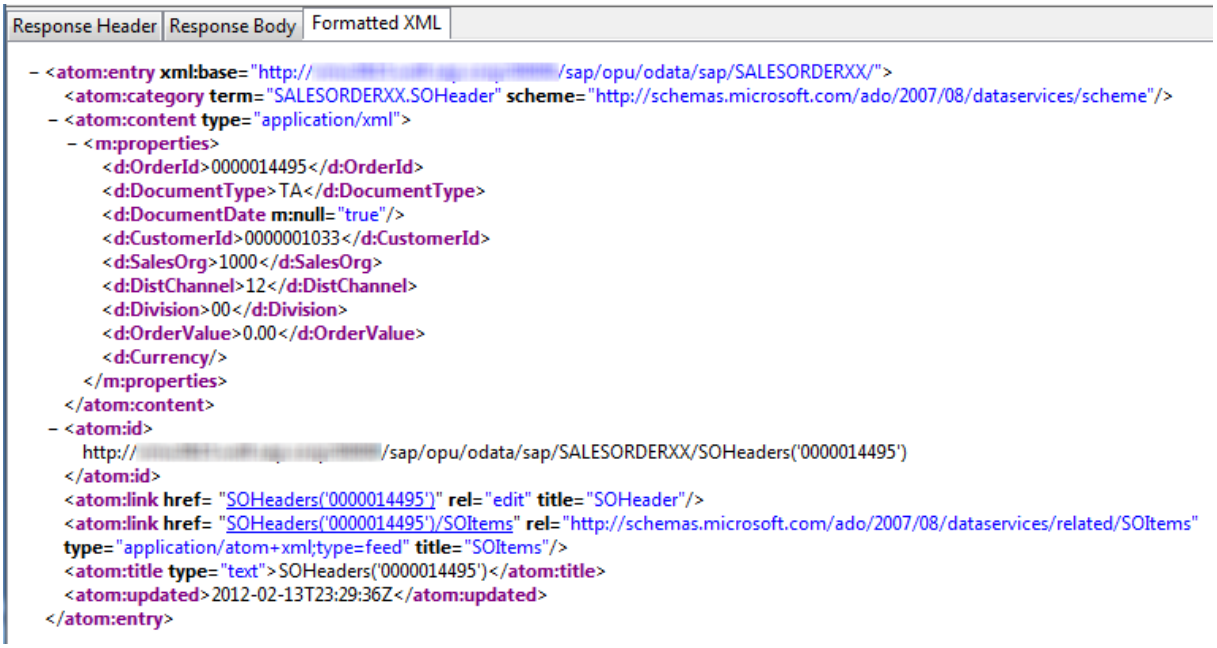


6. Upon successful creation, you should receive an HTTP "Status Code: 201 Created" in the Response Header.



An OData standard is that after every CREATE operation has completed, the client should automatically perform a READ operation. In the Response Header section, look at the *Location* parameter in the HTTP header returned to the client. This is the URL to perform the READ operation.

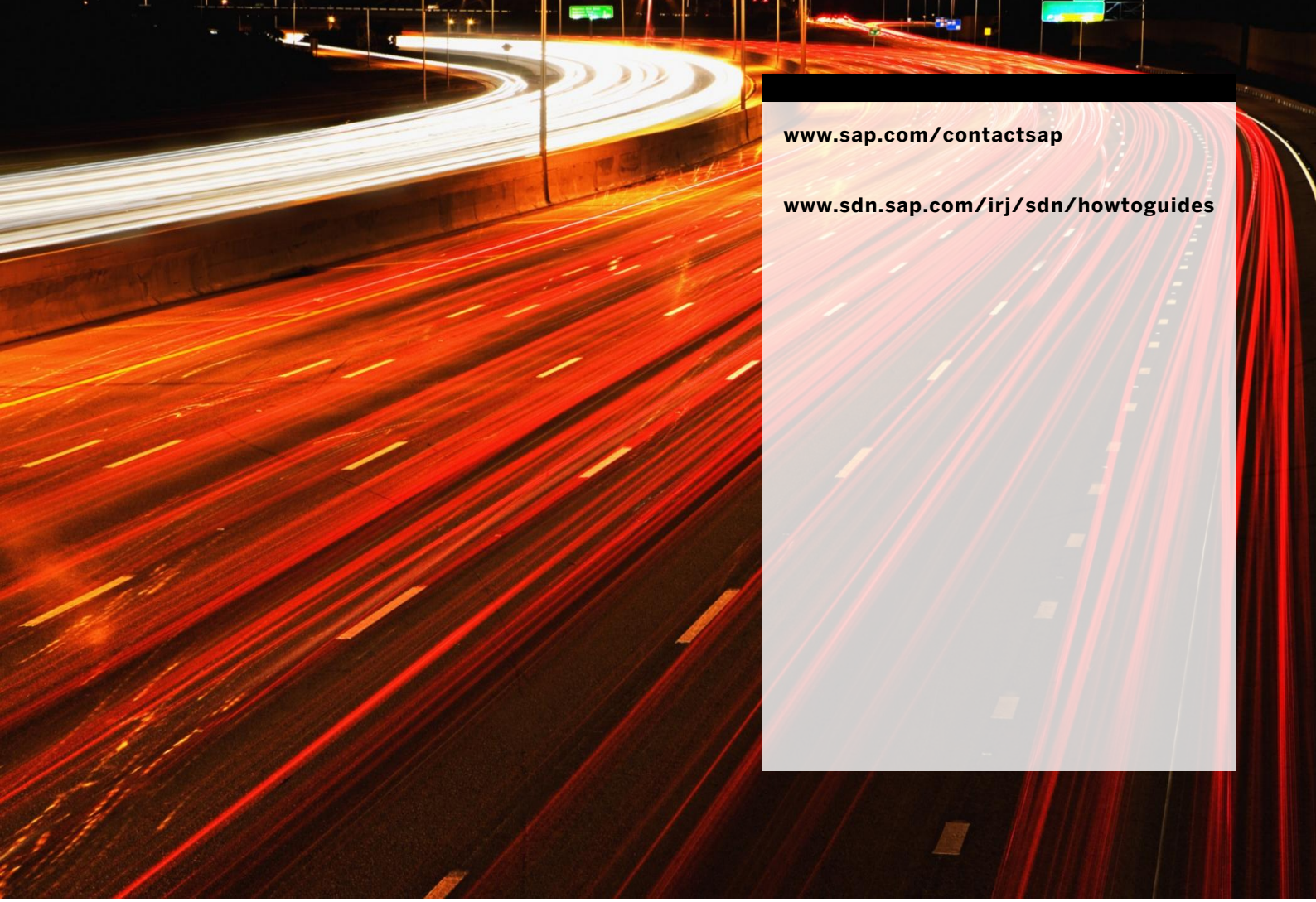
7. Now select the *Formatted XML* tab. In this section, you see the resulting XML returned after the client has performed the automatic READ.



5. Summary

Starting with SAP NetWeaver Gateway 2.0/SP2, the OData channel allows for reading and creating complex business entities in one request. With the new "deep insert" functionality, business objects such as Sales Orders or Purchase Orders, consisting of header and line item data, can now be created using Gateway services. Conversely, using the OData system query option, *$expand*, entities along with any associated entities can be retrieved inline using one request.

This document covered all the steps necessary to create and test an SAP NetWeaver Gateway service that can be used to query, read, and create a sales order in an ECC system, maintaining a focus on the CREATE operation where the deep insert functionality is needed.



www.sap.com/contactsap

www.sdn.sap.com/irj/sdn/howtoguides