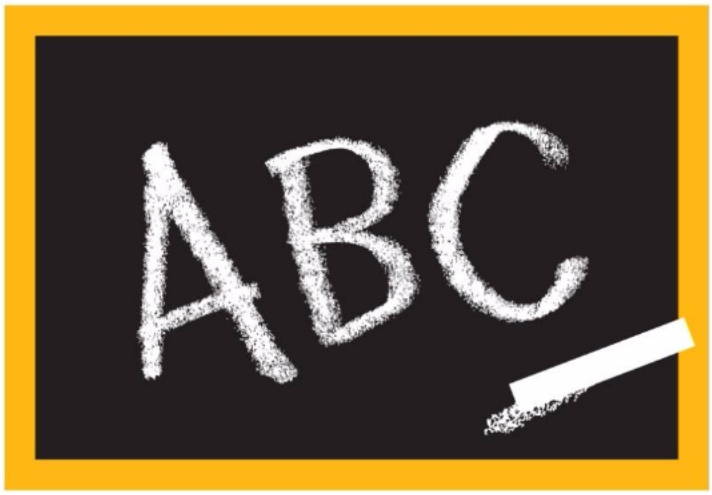


End to End Development Example in SAP® NetWeaver 7.4 & SAP® HANA

SAP NetWeaver 7.4 Support Package 5

Author: Jasmin Gruschke – jasmin.gruschke@sap.com

Target Audience

- Developers
- Consultants

For Public usage
Document version 1.00 – April, 2014

Contents

1 Remarks before you start

- This demo can be executed on an SAP NetWeaver AS ABAP 7.4 SP05 or higher running on a SAP HANA database SPS6 or higher.

- All screenshots have been made in an AS ABAP system with System ID (SID) “HANAABAP” installed in the HANA Database Schema “SAPHANAABAP”. Please consider to adapt this based on the SID and schema of your system.

- All ABAP Entities have been created in the “$TMP” package of the user “DEVELOPER”.

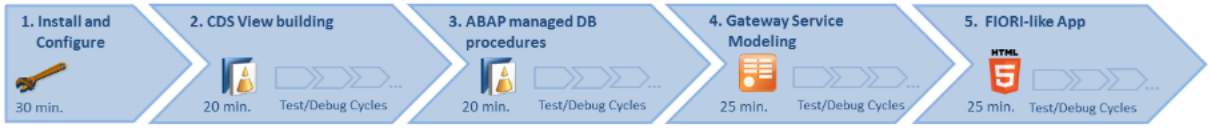
- You will need a user on SAP NetWeaver AS ABAP with the following roles assigned:
  - SAP_BC_DWB_ABAPDEVELOPER
  - SAP_BC_DWB_WBDISPLAY
  - /IWFND/RT_DEVELOPER (For Gateway Service Development)

- A dedicated HANA User is not necessary, however, in case you would like to check on created artifacts, you need a HANA user with role ABAP_DEV assigned.

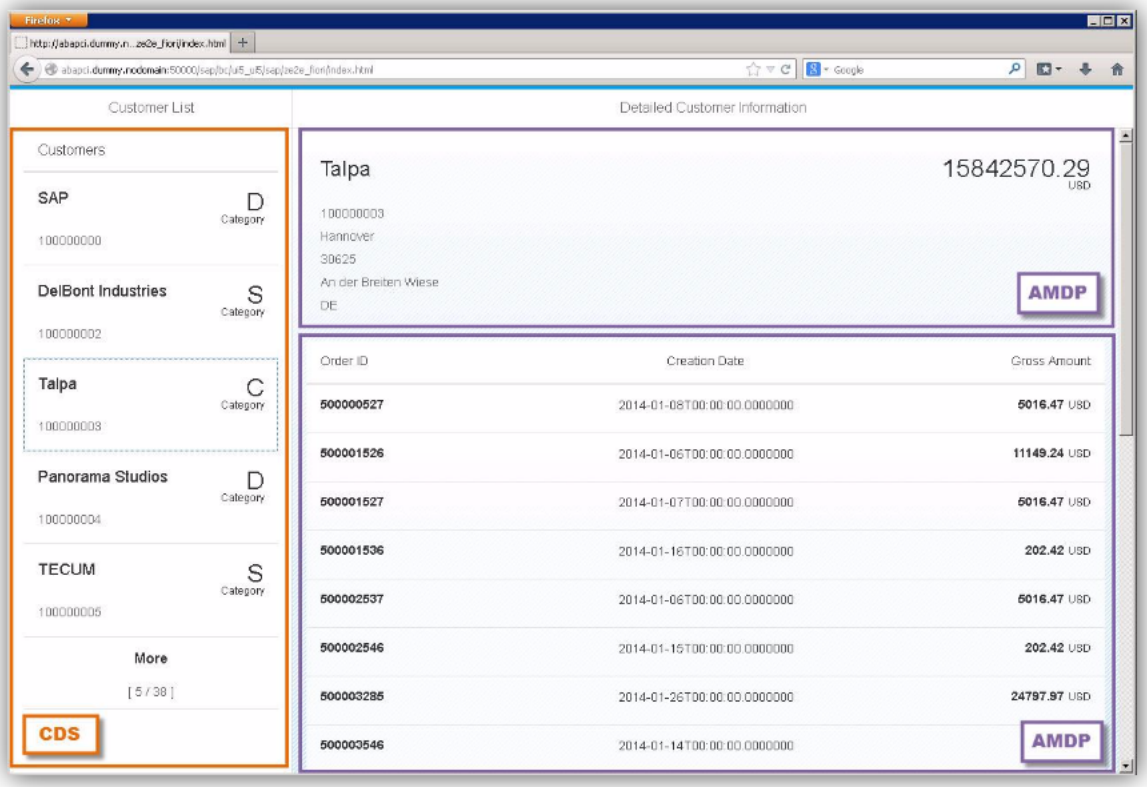
- The screenshots in this end-to-end guide have been created based on HANA Studio version 1.00.70 (Build id: 386119) and may differ with respect to other HANA releases.

- For more details, information and guides based on SAP NetWeaver AS ABAP and SAP HANA please visit our SCN Page: http://scn.sap.com/docs/DOC-35518.

2 What's inside this guide?



This document shows you an end-to-end development example from SAP HANA via ABAP to a Fiori-like application, which will look like:



On the left-hand side of the application, a list of customers is retrieved, showing the name and the ID of the customers as well as a category which depends on the number of open invoices.

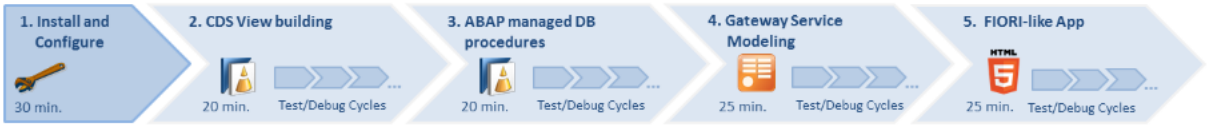
Selecting one of the customers from the list populates the right-hand side of the application. In the header part some more detailed information on the selected customer is shown, in particular the address information as well as the aggregated gross amount of all open invoices for this customer. In the lower part, the list of sales order invoices with their creation date and the gross amount per sales order invoice are listed. All gross amounts in the application are converted to USD.

From a technical point of view, the application is based on a SAP NetWeaver OData Service comprising the entity sets for the customer list (left-hand side) and the sales order invoice list (lower right-hand side) and an entity set for the detailed customer information (upper right-

hand side). While the customer list entity set is based on a CDS view, the information on the right-hand side is retrieved from ABAP-managed database procedures (AMDP).

If you now say "never heard from neither CDS nor AMDP" don't worry, you'll get to know these objects and their features while working yourself through this end-to-end guide ☺.

3 Technical Prerequisites



To follow this end-to-end guide you need an Application Server ABAP 7.4 SP5 (or higher) on SAP HANA SPS7 and an Eclipse-based development environment including ABAP Development Tools for SAP NetWeaver as well as the UI Development Toolkit for HTML5 (SAPUI5 Tools).

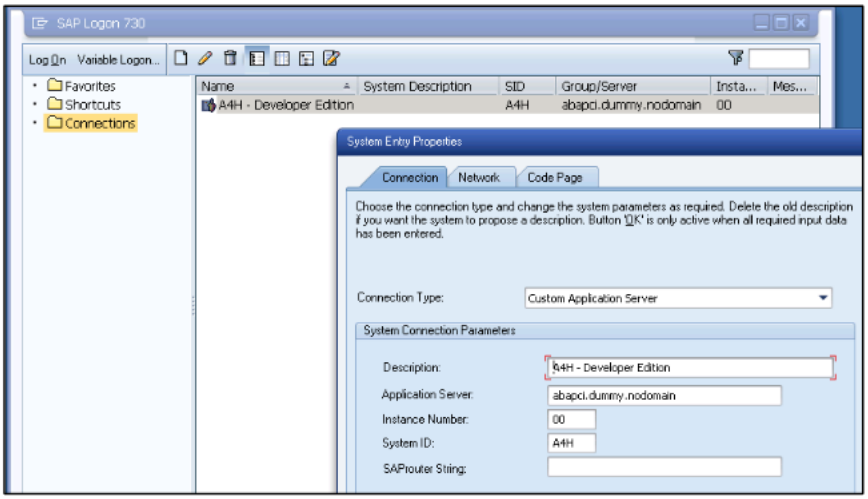
Concerning the backend system, the easiest method is to set up and run the developer edition of the AS ABAP 7.4, which is provided as virtual appliance by the SAP Cloud Appliance Library (see http://scn.sap.com/docs/DOC-52323). Choosing this variant, there are no additional system configuration steps. If you are working on other backend systems, please consult the system configuration steps described in the Appendix (System Configuration & Example Data Generation).

Using the virtual appliance as described in the document above, you can connect to your frontend image via a remote desktop connection. On the frontend image, you can find a SAP HANA studio installation including all necessary tools (ABAP development tools, UI tools). If you prefer a local Eclipse installation, please see the Appendix (Installation Guides) or the Additional Information chapter in the mentioned SCN document.

Check some preconditions

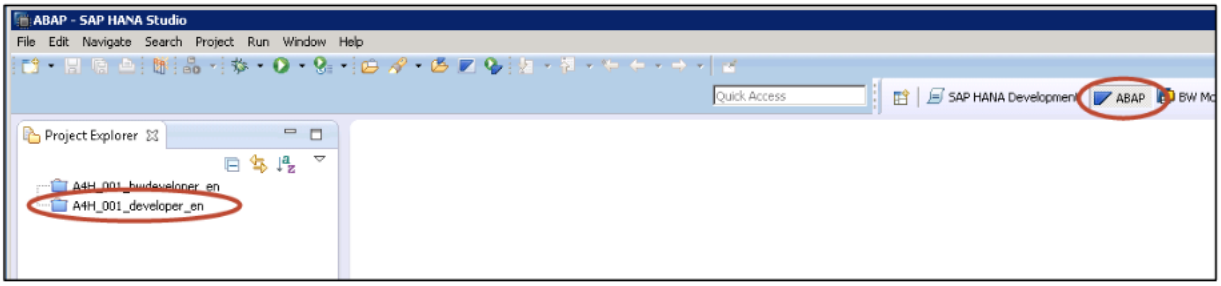
As mentioned above, if you are using the AS ABAP 7.4 SP5 as virtual appliance, you don't have to do any system configurations. However, it's worthwhile to check some preconditions.

1. Open the SAP Logon application and check that the ABAP system (in our case the connection to application server abapci.dummy.nodomain instance number 00) is maintained.

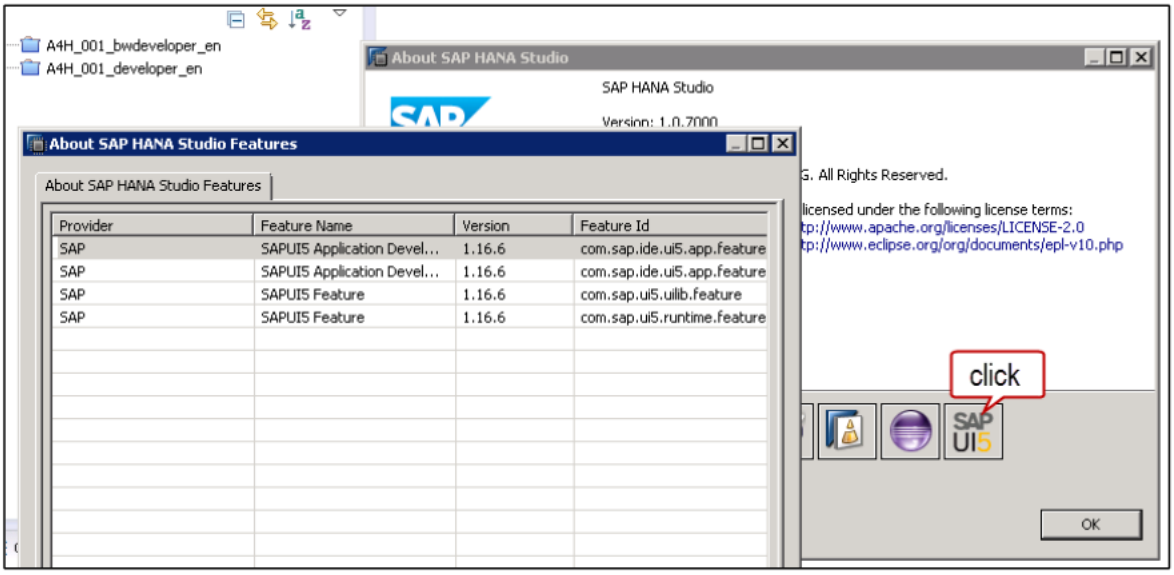


2. In the ABAP Perspective of the SAP HANA Studio, check that the ABAP Project for the backend system mentioned in the previous step exists. The ABAP Project belongs to user

DEVELOPER (client 001) and the passwort is (initially) identical to the master password of the virtual appliance instance (see http://scn.sap.com/docs/DOC-52323).



3. Please also verify that the UI Development Toolkit for HTML5 (SAPUI5 Tools) are installed, e.g. via the context menu *Help > About SAP HANA Studio*, where you should be able to find the SAP UI5 development toolkit with version 1.16.6.



4 Scenario Description

The scenario you are going to implement is part of the reference scenario delivered with AS ABAP 7.4. Based on the open item analysis, the scenario will give you several business figures like a simple customer classification and open invoice amounts per customer (including currency conversion).

Data Model

The data model you will be using consists of four tables. Each table has a primary key (datatype `GUID`) called `NODE_KEY` and the key component `CLIENT`.

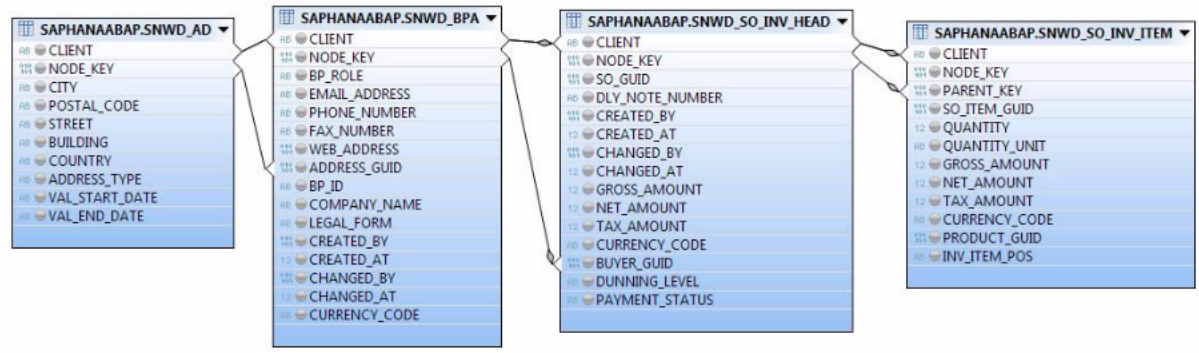
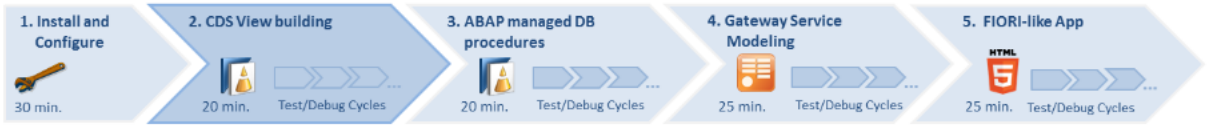


Table `SNWD_BPA` contains the relevant information on the business partner complemented by a 1:1 relation to the business partner address data given in table `SNWD_AD`. The information on sales order invoices is given in tables `SNWD_SO_INV_HEAD` (header data) and `SNWD_SO_INV_ITEM` (position/item data). The business partner table can be connected via a 1:n relation to the invoice header table field `BUYER_GUID`. The invoice header table can further be connected to the sales order item table in a 1:n relation using fields `SNWD_SO_INV_HEAD.NODE_KEY` to `SNWD_SO_INV_ITEM.PARENT_KEY`. One special feature of the data model is that the invoice items table contains all items of a given invoice in different currencies.

Full Reference Application

The scenario described here is just a small part of the full ABAP for SAP HANA reference scenario. More details can be found in transaction `SEPM_OIA` in your AS ABAP or on SCN http://scn.sap.com/docs/DOC-35518.

5 CDS View Building in ABAP



As mentioned in chapter What's inside this guide?, the final application displays a list of customers including the customer name, the customer ID, and the customer classification category. While this could be performed by fetching the data into an internal table on the application server layer and perform all necessary calculation there, we'll follow the code-to-data paradigm, i.e. we push data-intensive calculation logic to the database layer and only fetch the (display) relevant data to the application server. More information on the code-to-data paradigm can be found in http://scn.sap.com/community/abap/hana/blog/2014/02/03/abap-for-hana-code-push-down.

The task at hand could be done using advanced functionality of Open SQL (see http://help.sap.com/abapdocu_740/en/index.htm?file=ABENNEWS-740_SP05-OPEN_SQL.htm). We've chosen an alternative approach; we'll make use of a new feature in the AS ABAP 7.4, namely a Core Data Services (CDS) view.

In general, Core Data Services (CDS) is an enhancement of SQL which allows a simple and harmonized way for the definition and consumption of semantically rich data models natively in HANA applications – independent of the consumption technology. The enhancements compare to SQL include:

- Annotations to enrich the data models with metadata
- Associations on a conceptual level, replacing joins with simple path expressions
- Expressions used for calculations and queries in the data model

You may say: We don't want to create a data model in this end-to-end guide, which is entirely true. However, we can make use of the advanced ABAP view building capabilities to define a CDS view, facilitating the query on the data model described in chapter Data Model.

CDS views, like the well-known dictionary views created and maintained in transaction `SE11`, are managed by the ABAP data dictionary. During activation, a database view is created on the HANA layer, yet only the ABAP CDS view (defined in a so-called DDL source) has to be transported via the ABAP Change and Transport System (CTS). Moreover, the functionality provided by CDS views can be used on all SAP supported databases, you don't have to worry when transporting these objects in a heterogeneous system landscape.

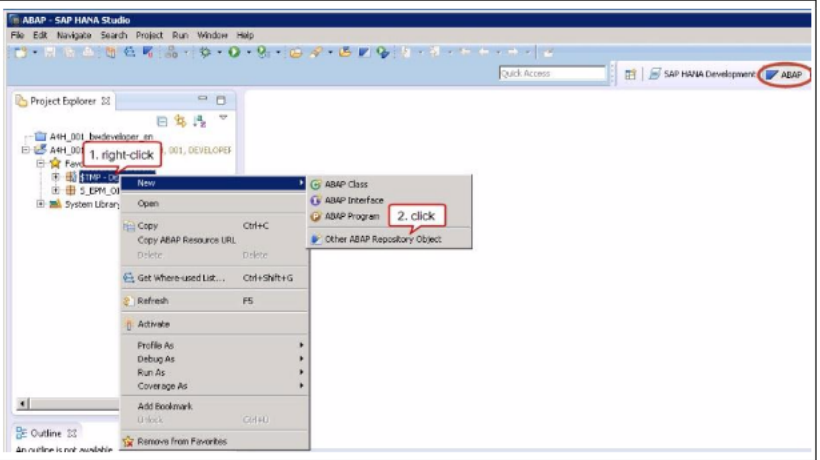
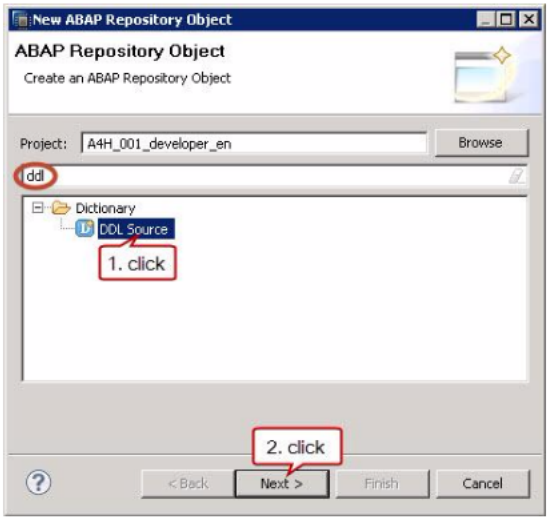
More detailed information on CDS View building can be found in http://scn.sap.com/community/abap/eclipse/blog/2014/02/04/new-data-modeling-features-in-abap-for-hana and references therein.

Let's create the CDS View(s)

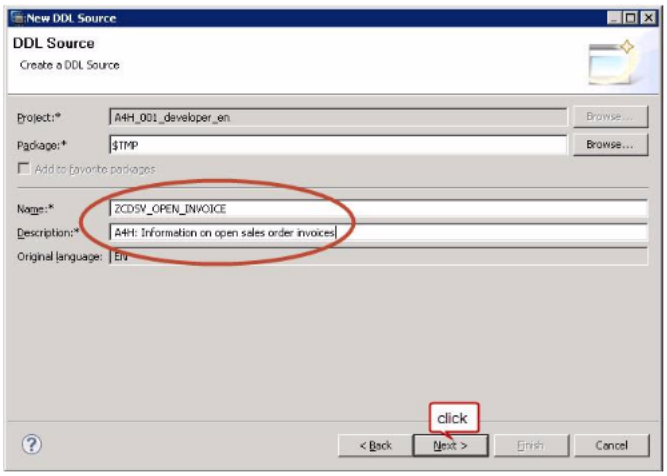
In our development example, we'll create two CDS view:

- `ZCDSV_OPEN_INVOICE`, used to calculate the number of unpaid sales order invoices per customer
- `ZCDSV_CUST_CLASSIFICATION`, used to consume `ZCDSV_OPEN_INVOICE` and to additionally provide the customer name and ID information

For the definition of CDS views in the ABAP data dictionary, an ABAP DDL source object (`R3TR DDLS`) has to be created. These new objects can only be created and maintained with the ABAP Development Tools for SAP NetWeaver, the artist also known as ABAP in Eclipse, so we'll start the task in the ABAP perspective of our SAP HANA studio:

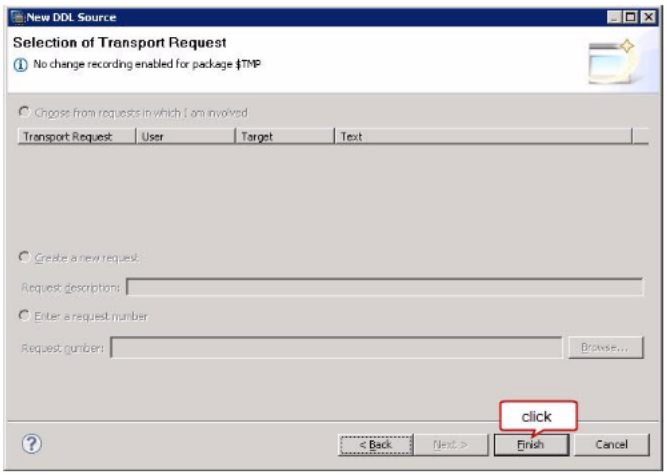
Description	Screen Shot
1. In the ABAP perspective of the SAP HANA studio, right-click on the `$TMP` folder of your ABAP project and select *New > Other ABAP Repository Object* from the context menu.	
2. In the creation dialog window, filter on "ddl", select *Dictionary > DDL Source* and continue with *Next*.	

3. In the New DDL Source dialog, enter the information:
   a. Name: `ZCDSV_OPEN_INVOICE`
   b. Description: A4H: Information on open sales order invoices

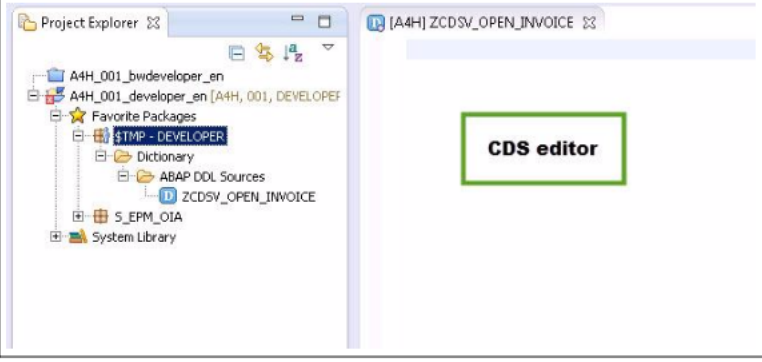


4. You develop in the `$TMP` package, therefore you're informed, that no transport request is necessary. Just confirm this information.

   Note: If you chose a transport-relevant package, you have to create/select a transport request.



5. You can find the DDL source in the project explorer view and an empty CDS editor window is opened.



Code snippet: `ZCDSV_OPEN_INVOICE`

```
@AbapCatalog.sqlViewName: 'ZV_CDS_INVC'
define view zcdsv_open_invoice as select from snwd_so_inv_head
{
    key snwd_so_inv_head.buyer_guid,
    'C' as category
}
where snwd_so_inv_head.payment_status <> 'P'
group by snwd_so_inv_head.buyer_guid
having count( distinct snwd_so_inv_head.node_key ) <= 2000

union all
```

```
select from snwd_so_inv_head
{
    key snwd_so_inv_head.buyer_guid,
    'D' as category
}
where snwd_so_inv_head.payment_status <> 'P'
group by snwd_so_inv_head.buyer_guid
having count( distinct snwd_so_inv_head.node_key ) >  2000
   and count( distinct snwd_so_inv_head.node_key ) <= 4000

union all

select from snwd_so_inv_head
{
    key snwd_so_inv_head.buyer_guid,
    'S' as category
}
where snwd_so_inv_head.payment_status <> 'P'
group by snwd_so_inv_head.buyer_guid
having count( distinct snwd_so_inv_head.node_key ) > 4000
```

The CDS view is built as union of three distinct select statements. According to the number of open (not-yet paid) sales orders invoices `No_of_Inv`, the customers are categorized according to:

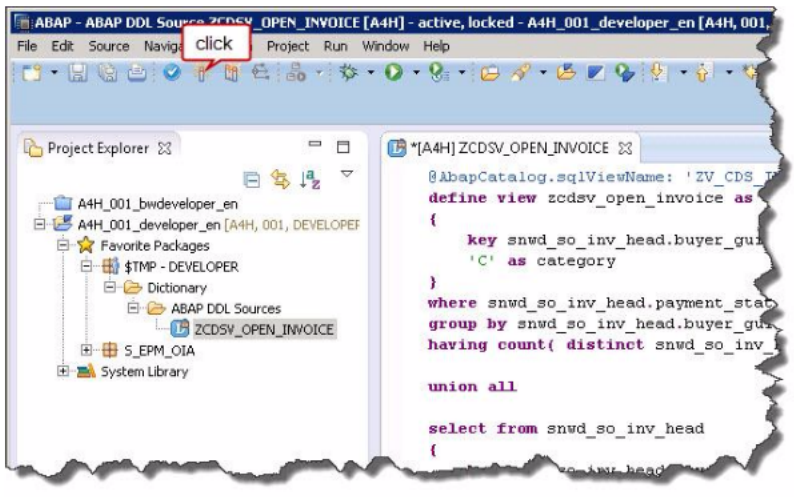
- Category C: `No_of_Inv` <= 2.000
- Category D: 2.000 < `No_of_Inv` <= 4.000
- Category S: 4.000 < `No_of_Inv`

Remark: You may ask whether the usage of the HAVING clause and therefore the usage of the UNION ALL functionality in the CDS view is necessary. The answer is yes, since CDS currently does not feature a searched CASE functionality like

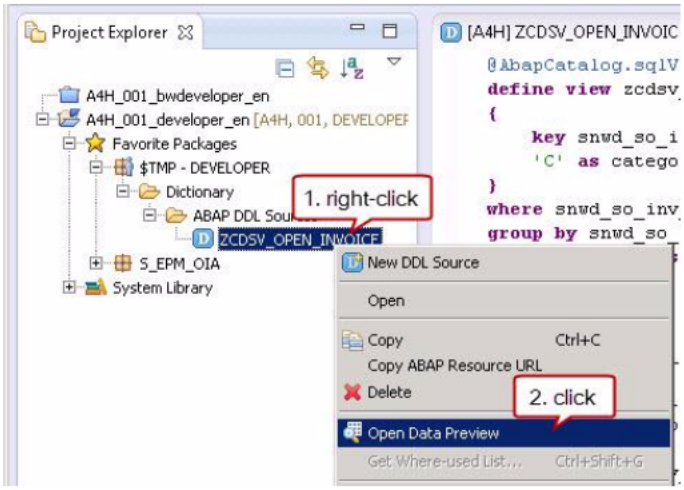
`CASE … when expression; then …`",

which would simplify the given task. Let me just say: Stay tuned ☺!

6. Activate the view (short cut `Ctrl+F3`).

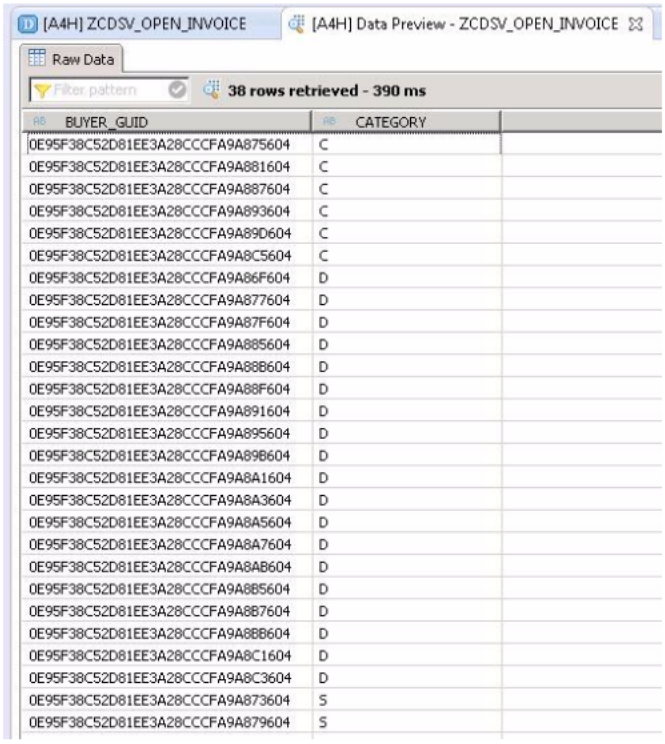


7. You can have a look at the output of the view. Right-click on the DDL source in the project explorer view to open the context menu and select *Open Data Preview*.

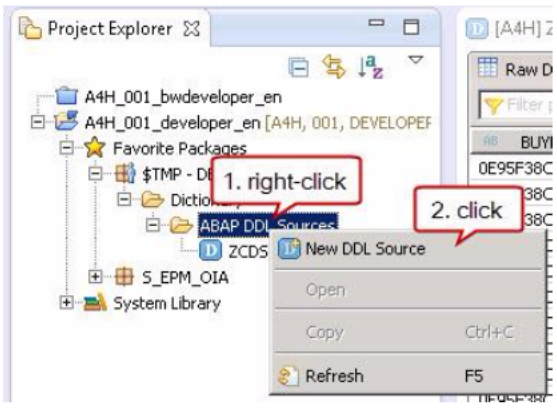


8. The data preview displays a list of customer unique IDs (`BUYER_GUID`) and the classification category.

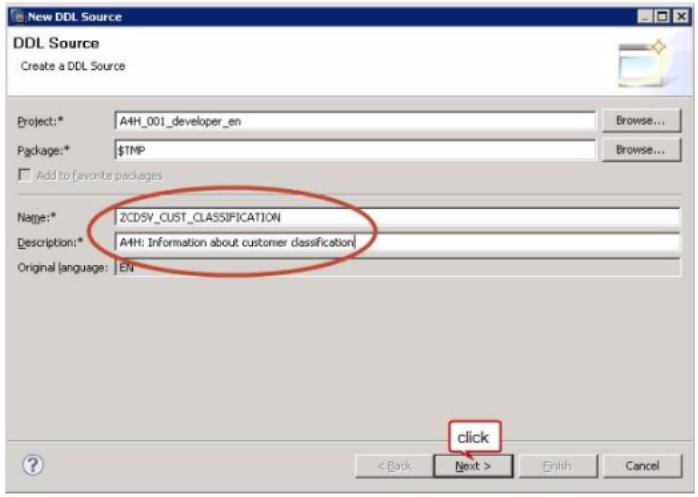
   Note: If the data preview shows an empty result set,
   a. you either need to double-check the modelling of your CDS views or
   b. you're lacking data in the system, e.g. if you're not using a virtual appliance and didn't generate data (see Generate Example Data). You may check the number of entries in transaction `SE11` (on table `SNWD_BPA`)



9. Ready for the second view! Right-click on the ABAP DDL Sources folder in the project explorer and select *New DDL Source* from the context menu



10. Maintain the necessary information:
    a. Name: `ZCDSV_CUST_CLASSIFICATION`
    b. Description: A4H: Information about customer classification

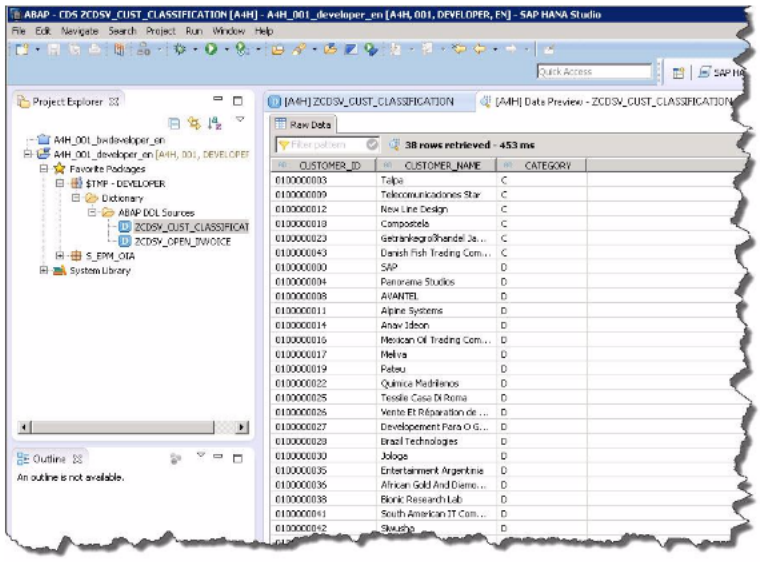


Code snippet: ZCDSV_CUST_CLASSIFICATION

```
@AbapCatalog.sqlViewName: 'ZV_CDS_CUST'
define view zcdsv_cust_classification as select from snwd_bpa as bpa
join zcdsv_open_invoice as opn_inv on bpa.node_key = opn_inv.buyer_guid
{
    key bpa.bp_id         as customer_id,
    key bpa.company_name as customer_name,
    opn_inv.category
 }
```

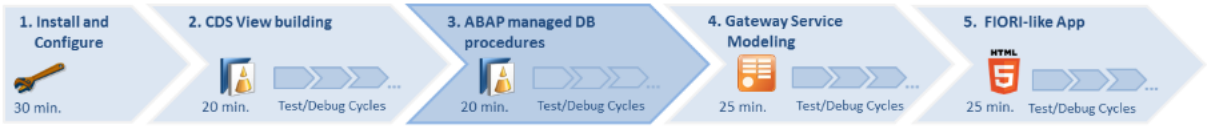
In the CDS view, the business partner table `SNWD_BPA` is joined to the previously created CDS view `ZCDSV_OPEN_INVOICE` on the customer unique ID, and semantic aliases are given to the business partner information.

11. Activate the DDL source (`Ctrl+F3`) and check the output of the view via the *Open Data Preview* feature in the context menu of the DDL source in the project explorer (see step 7).



Okay, you completed the first (or second if you count the configuration) step in this end-to-end development guide; you used the advanced view building capabilities of the ABAP 7.4 SP5 to create a CDS view.

6 ABAP managed Database Procedures (AMDP)



In the previous chapter you saw the usage of CDS views as one technique – of course Open SQL would have been another – for push-down of application logic to the database layer. Now, you'll get to know another techniques for code pushdown in the ABAP 7.4 SP5 namely ABAP managed Database procedures (AMDPs).

From a technical point of view, an ABAP managed database procedure is "just" a simple ABAP class methods containing database-specific procedure coding. Thus, an ABAP developer can write database procedure coding directly in the ABAP and the call of the database procedure is just a call of the class method.

But – you might guess – AMDPs can do even more; if you are developing an SQLScript-based AMDP on HANA, the AMDP runtime provides you with a complete syntax check on the SQLScript coding. And concerning lifecycle, AMDPs as well as the underlying database procedure are entirely managed by the ABAP server. In particular, only the ABAP class method has to be transported (using the well-known CTS) while the creation of the database procedure on the database layer happens at first execution of the class method. This technique allows – as was the case for CDS views – to transport AMDPs in heterogeneous system landscapes.

More information on AMDPs can be found in http://scn.sap.com/docs/DOC-51612 and references therein.

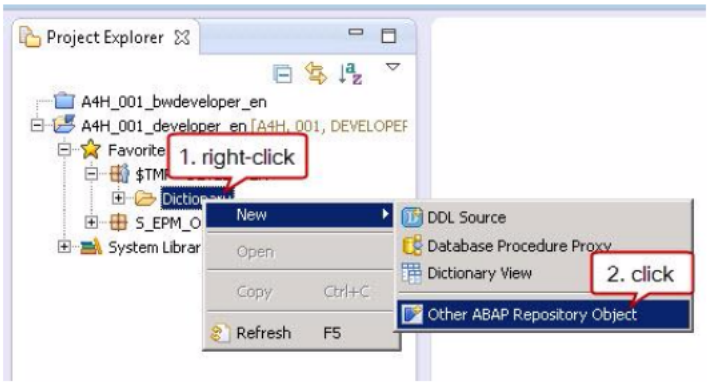
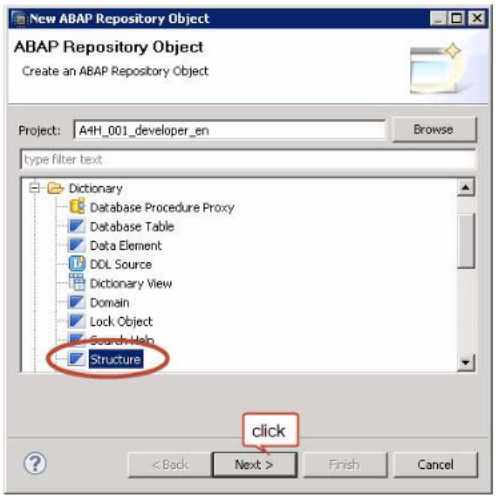
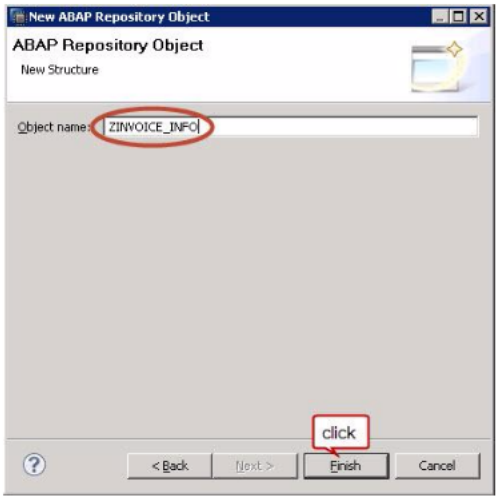
Let's create the AMDP

As mentioned in chapter What's inside this guide?, we'll create three ABAP managed database procedures:

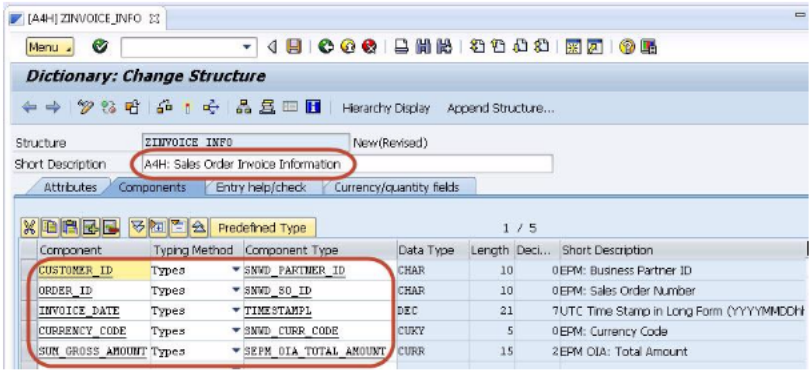
- `get_customer_info`, to retrieve customer information for a given customer including the summed gross amount currency-converted to USD for all open sales order invoices
- `get_invoice_info`, to retrieve the list of open sales order invoices for a given customer including the gross amount currency-converted to USD
- `get_curr_conv_relevant_items`, to retrieve the list of relevant sales order invoice items – consumed in the two previous AMDPs

ABAP managed database procedures can only be created and maintained via the ABAP Development Tools in Eclipse, so we go back (well technically we never left ☺) to the ABAP perspective of our SAP HANA studio.

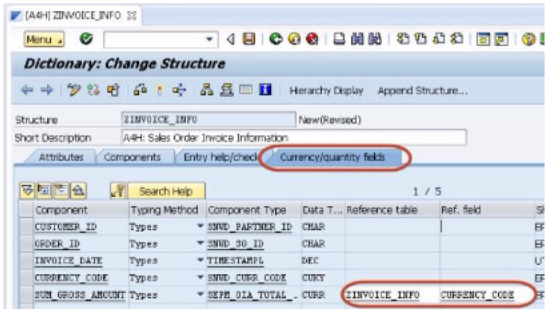
Before we start with the implementation of the AMDP, we'll create structured data types in the ABAP dictionary (transaction `SE11`). This will allow a simplified definition of the Gateway services and the consumption in the Fiori-like application.

Description	Screen Shot
1. In the ABAP perspective of the SAP HANA studio, right-click on the Dictionary folder of the `$TMP` package and select *New > Other ABAP Repository Object* from the context menu.	
2. In the New ABAP Repository Object dialog window, select *Dictonary > Structure*	
3. Provide the object name `ZINVOICE_INFO`	

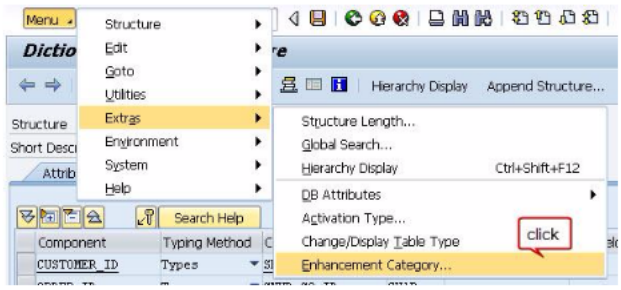
4. An embedded SAPGui opens (transaction `SE11`). Provide as short description “A4H: Sales Order Invoice Information” and the field list:
   - `CUSTOMER_ID Types SNWD_PARTNER_ID`
   - `ORDER_ID Types SNWD_SO_ID`
   - `INVOICE_DATE Types TIMESTAMPL`
   - `CURRENCY_CODE Types SNWD_CURR_CODE`
   - `SUM_GROSS_AMOUNT Types SEPM_OIA_TOTAL_AMOUNT`



5. Navigate to the Currency/quantity fields tab and maintain the currency code for field `SUM_GROSS_AMOUNT` as reference table `ZINVOICE_INFO` and reference field `CURRENCY_CODE`



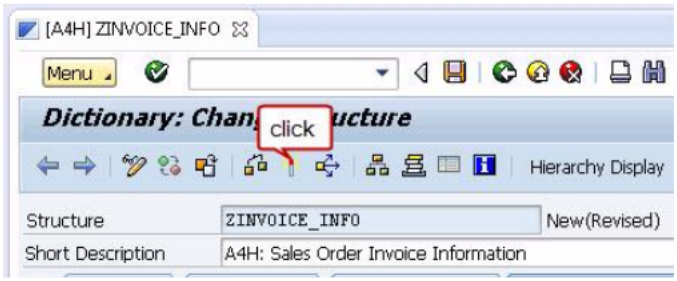
6. Maintain the enhancement category via *Menu > Extras > Enhancement Category*.



7. Confirm the information that you haven’t select an enhancement category yet… you actually just wanted to do so ☺…

8. Purely for simplicity reasons, choose *"Cannot Be Enhanced"* as Enhancement category.

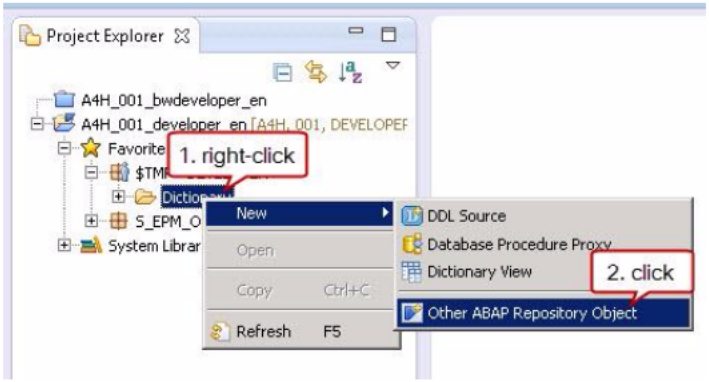
9. And with these eight easy steps, you're ready to activate the structure type via the well-known matchstick icon.



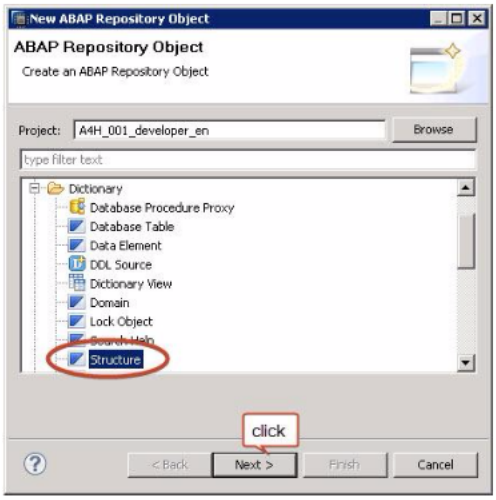
10. Check that the create object directory entry popup shows the correct packages (`$TMP` in our case) and click on the save icon.

11. Now, we'll repeat steps 1-10 to create the structure type `ZCUSTOMER_INFO`.
Don't scroll back up… I'll repeat them for you ☺

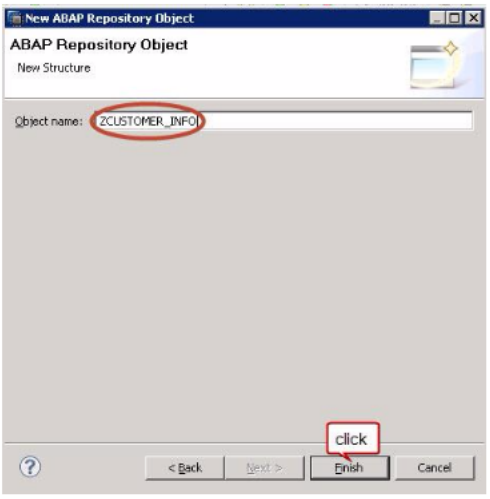
In the ABAP perspective of the SAP HANA studio, right-click on the Dictionary folder of the `$TMP` package and select *New > Other ABAP Repository Object* from the context menu.



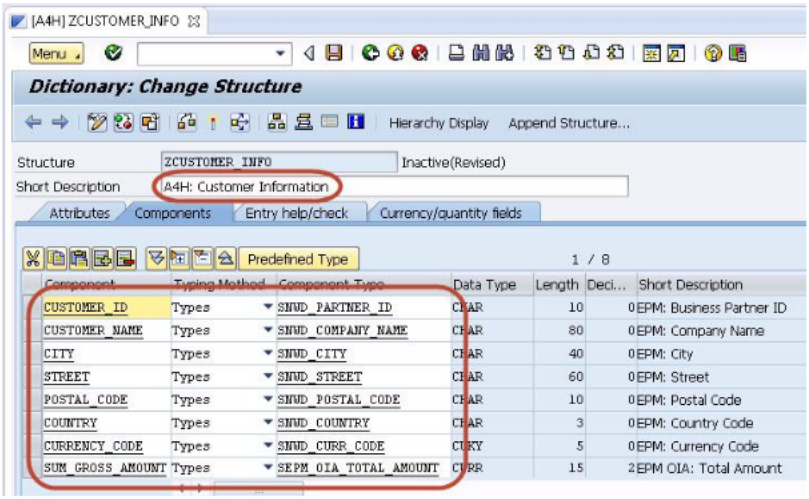
12. In the New ABAP Repository Object dialog window, select *Dictonary > Structure*



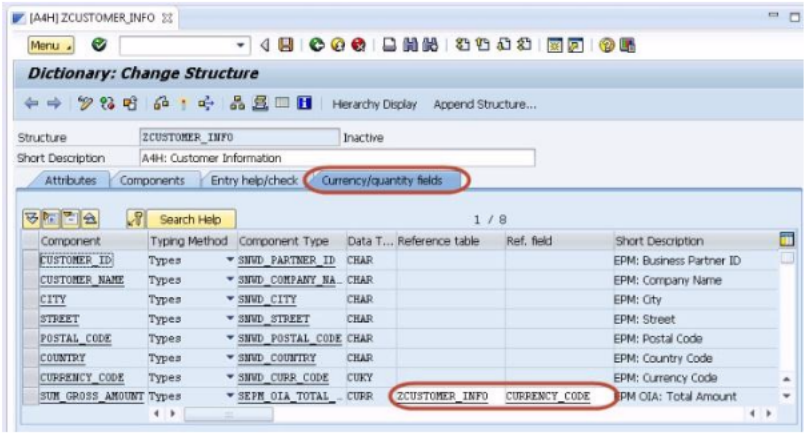
13. Provide the object name `ZCUSTOMER_INFO`



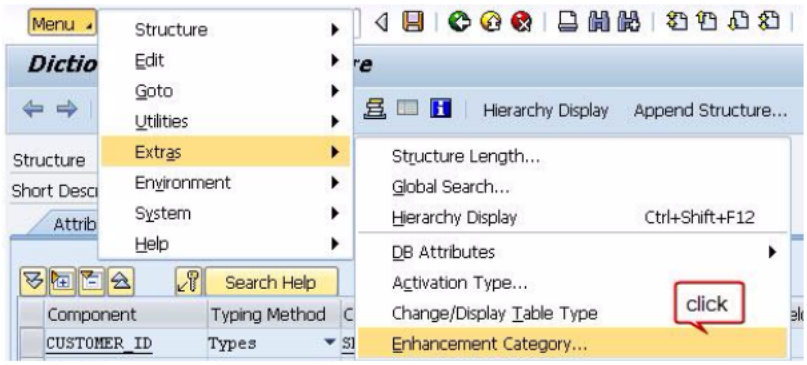
14. An embedded SAPGui opens (transaction `SE11`). Provide a short description "A4H: Customer Information" and the field list:
    - `CUSTOMER_ID Types SNWD_PARTNER_ID`
    - `CUSTOMER_NAME Types SNWD_COMPANY_NAME`
    - `CITY Types SNWD_CITY`
    - `STREET Types SNWD_STREET`
    - `POSTAL_CODE Types SNWD_POSTAL_CODE`
    - `COUNTRY Types SNWD_COUNTRY`
    - `CURRENCY_CODE Types SNWD_CURR_CODE`
    - `SUM_GROSS_AMOUNT Types SEPM_OIA_TOTAL_AMOUNT`



15. Navigate to the Currency/quantity fields tab and maintain the currency code for field `SUM_GROSS_AMOUNT` as reference table `ZCUSTOMER_INFO` and reference field `CURRENCY_CODE`



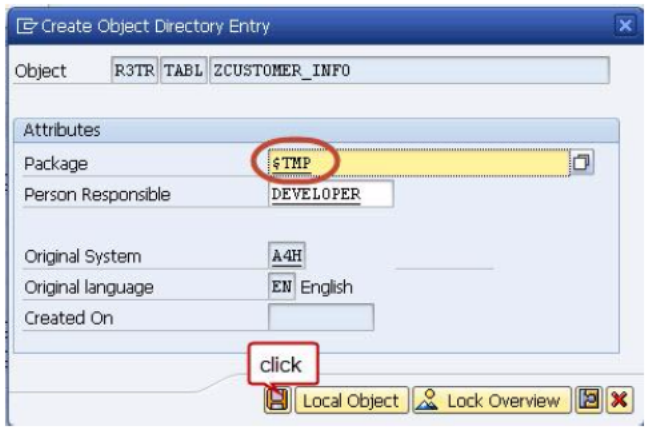
16. Maintain the enhancement category via *Menu > Extras > Enhancement Category*.



17. Confirm the information that you haven't select an enhancement category yet… you actually just wanted to do so ☺…

18. Purely for simplicity reasons, choose *"Cannot Be Enhanced"* as Enhancement category.

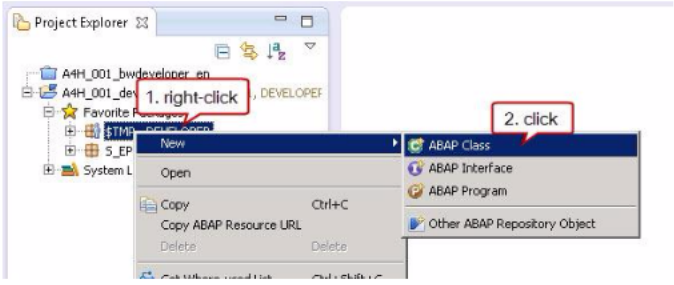
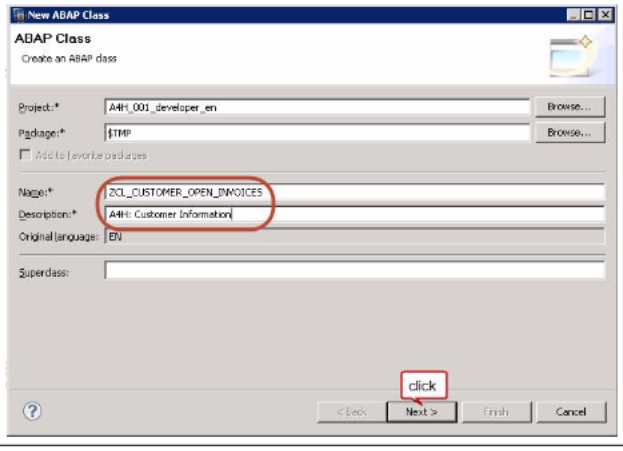
19. Activate the structure type via the well-known matchstick icon.

20. Check that the create object directory entry popup shows the correct packages (`$TMP` in our case) and click on save.	

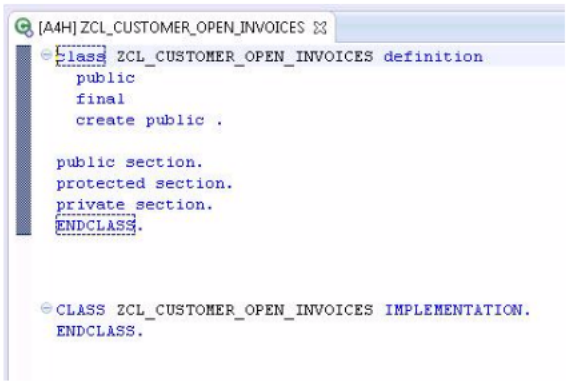
As mentioned before, we created these structure types since this simplifies the definition of the Gateway OData service as you will see in chapter GW Service. In general, ABAP managed database procedures do not require these steps. In principle, the structure types could have as well been defined in an ABAP interface or in an ABAP class.

Let's create the AMDP… now we really start!

You'll now implement the three ABAP managed database procedures `get_customer_info`, `get_invoice_info`, and the supporting `get_curr_conv_relevant_items`. All three methods will be implemented as class methods of the ABAP class `ZCL_CUSTOMER_OPEN_INVOICES`.

Description	Screen Shot
1. In the ABAP perspective of the SAP HANA studio, right-click on the `$TMP` package and select *New > ABAP Class* from the context menu.	
2. In the New ABAP Class dialog window provide: a. Name: `ZCL_CUSTOMER_OPEN_INVOICES` b. Description: A4H: Customer Information Continue via *Next* and confirm the following "no transport request is necessary" dialog.	

3. The ABAP Class editor window opens. The class definition and implementation skeletons are shown which you'll implement in the following steps.



4. Prepare the public section of the class definition:
   a. Prepare the ABAP class for AMDP usage via the marker interface
   b. Define the table types `TT_CUST_INFO` and `TT_INV_INFO` based on the structure types `ZCUSTOMER_INFO` and `ZINVOICE_INFO`
   c. Define the methods `get_customer_info` and `get_invoice_info` with AMDP-conform parameter interfaces, e.g. parameters are passed by value.

Code snippet

```
PUBLIC SECTION.
    INTERFACES: if_amdp_marker_hdb.
    TYPES:
      tt_cust_info TYPE STANDARD TABLE OF
         zcustomer_info WITH KEY customer_id,
      tt_inv_info  TYPE STANDARD TABLE OF
         zinvoice_info  WITH KEY customer_id.

    METHODS:
      get_customer_info
        IMPORTING
          VALUE(iv_client) TYPE symandt
          VALUE(iv_bupaid)
            TYPE zcustomer_info-customer_id
        EXPORTING
          VALUE(et_bpinfo) TYPE tt_cust_info,
      get_invoice_info
        IMPORTING
          VALUE(iv_client) TYPE symandt
          VALUE(iv_bupaid)
            TYPE zcustomer_info-customer_id
        EXPORTING
          VALUE(et_invinfo) TYPE tt_inv_info.
```

5. Prepare the private section of the class definition:
   a. Define the structure type `ty_rel_items` as well as the table type `tt_rel_items`
   b. Define the method `get_curr_conv_relevant_items` with an AMDP-conform parameter interface

   Note: The methods you just defined use the client as input parameter. If you are familiar with the ABAP

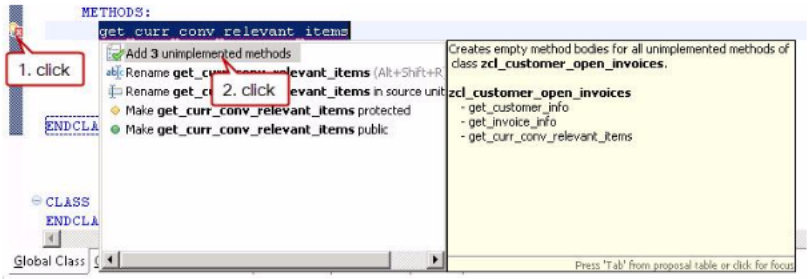
Code snippet

```
PRIVATE SECTION.
  TYPES:
    BEGIN OF ty_rel_items,
      client       TYPE snwd_so_inv_item-client,
      inv_i_guid   TYPE snwd_so_inv_item-node_key,
      inv_guid     TYPE snwd_so_inv_head-node_key,
      buyer_guid   TYPE snwd_bpa-node_key,
      customer_id  TYPE snwd_bpa-bp_id,
      invoice_date TYPE snwd_so_inv_head-created_at,
      gross_amount
        TYPE snwd_so_inv_item-gross_amount,
      currency_code_conv
        TYPE snwd_so_inv_item-currency_code,
    END OF ty_rel_items,

    tt_rel_items TYPE
       STANDARD TABLE OF ty_rel_items.
```

client handling, you know that OpenSQL statements per default add client filter (via a WHERE clause enhancement); except if a CLIENT SPECIFIED clause is added. In native SQL or in database procedures, you have to take care about client handling yourself.

```
METHODS:
  get_curr_conv_relevant_items
    IMPORTING
      VALUE(iv_client) TYPE symandt
      VALUE(iv_bupaid)
        TYPE zcustomer_info-customer_id
    EXPORTING
      VALUE(et_conv_items) TYPE tt_rel_items.
```

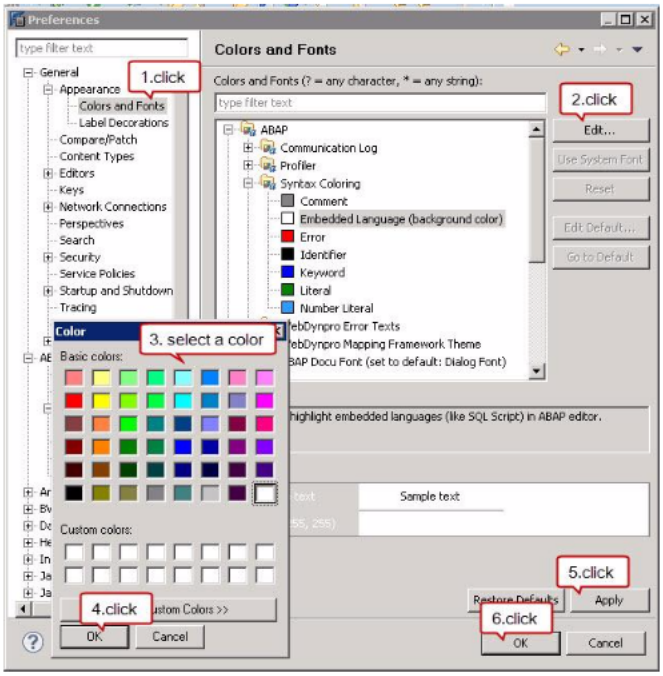
6. You defined three unimplemented methods. The light-bulb icon indicates there is a quick fix for this issue. Click on the icon and select the quick-fix to implement the methods.



7. The three method implementations are added. Let's start with the implementation of the `get_current_conv_relevent_items`.

```
CLASS zcl_customer_open_invoices IMPLEMENTATION.

  METHOD get_curr_conv_relevant_items.

  ENDMETHOD.


  METHOD get_invoice_info.

  ENDMETHOD.


  METHOD get_customer_info.

  ENDMETHOD.

ENDCLASS.
```

8. Before we start the implementation, configure the background coloring for embedded languages. Open *Windows > Preferences* (not shown on the screenshot) select *General > Appearance > Colors and Fonts*, Select *ABAP > Syntax Coloring > Embedded Language*. Via the Edit button you get some basic colors you can choos from or – my personal favourite – you can choose Custom Colors.



9. Back to the ABAP Class editor, implement the `get_curr_conv_relevant_items` as read-only AMDP. This is achieved using the language statement "BY DATABASE PROCEDURE" and we'll do an AMDP for the HANA database (HDB) using SQLScript as language. The using clause includes the ABAP dictionary types that will be used to query from.

   The method body only contains SQLScript coding. First the actual date is retrieved, which will be used as date for the currency conversion later.
   Then, the sales order invoice items for a given business partner (provided via the importing parameter `iv_bupaid`) are selected if the `payment_status` of the sales order is empty (not-yet paid). And

Code snippet: `get_curr_conv_relevant_items`

```
METHOD get_curr_conv_relevant_items BY DATABASE PROCEDURE
         FOR HDB
         LANGUAGE SQLSCRIPT
         OPTIONS READ-ONLY
         USING snwd_bpa snwd_so_inv_head snwd_so_inv_item.

    -- declare a local variable
    declare lv_today date;
    -- get current date for conversion
    select current_date into lv_today from dummy;

    -- select relevant invoice items
    lt_relevant_items =
      select
         i.client      as client,
         i.node_key    as inv_i_guid,
         h.node_key    as inv_guid,
         bpa.node_key  as buyer_guid,
         bpa.bp_id     as customer_id,
         h.created_at  as invoice_date,
         i.gross_amount,
         i.currency_code
      from snwd_so_inv_item as i
      join snwd_so_inv_head as h
        on i.client     = h.client
       and i.parent_key = h.node_key
      join snwd_bpa          as bpa
        on h.client     = bpa.client
       and h.buyer_guid = bpa.node_key
     where h.client     = :iv_client
       and bpa.bp_id    = :iv_bupaid
       and h.payment_status = '';

    --convert gross amount of items to currency 'USD'
    et_conv_items =
```

additional information on the business partner and the sales order header is selected into the temporary table `lt_relevant_items`. All these relevant sales order items in `lt_relevant_items` are subject to a currency conversion routine (using the CE-function `ce_conversion`) with target currency USD and providing the resultset for the exporting parameter `et_conv_items`.

```
      ce_conversion( :lt_relevant_items,
                     [ family = 'currency',
                       method = 'ERP',
                       steps  =
'shift,convert,shift_back',
                       source_unit_column =
"CURRENCY_CODE"     ,
                       output_unit_column =
"CURRENCY_CODE_CONV",
                       target_unit        = 'USD',
                       reference_date     = :lv_today,
                       client             = :iv_client
],
                       [gross_amount] ) ;

  ENDMETHOD.
```

10. Implement the method `get_invoice_info` as AMDP and provide the ABAP dictionary tables `SNWD_BPA` and `SNWD_AD`, as well as the AMDP `ZCL_CUSTOMER_OPEN_INVICES=>get_curr_conv_relevant_items` in the using clause.

    In the method body, the AMDP is called and the result is retrieved into the temporary table `lt_converted_items`.

    Then the gross amounts are aggregated per sales order invoice and the information on the sales order ID, the invoice date are added.

    The result set of the query is return via the exporting table `et_invinfo`.

Code snippet: `get_invoice_info`

```
METHOD get_invoice_info BY DATABASE PROCEDURE
       FOR HDB
       LANGUAGE SQLSCRIPT
       OPTIONS READ-ONLY
       USING snwd_so snwd_so_inv_head
zcl_customer_open_invoices=>get_curr_conv_relevant_items.


    call
"ZCL_CUSTOMER_OPEN_INVOICES=>GET_CURR_CONV_RELEVANT_ITEMS"
(
             iv_client     => :iv_client,
             iv_bupaid     => :iv_bupaid,
             et_conv_items => :lt_converted_items );

    --aggregated gross amounts per sales order invoice
    et_invinfo =
         select
           customer_id,
           so_id as order_id,
           invoice_date,
           currency_code_conv as currency_code,
           sum( conv_items.gross_amount )
             as sum_gross_amount
         from :lt_converted_items as conv_items
         join snwd_so_inv_head as h
           on h.client     = conv_items.client
          and h.node_key   = conv_items.inv_guid
        join snwd_so as so
           on so.client    = h.client
          and so.node_key  = h.so_guid
         group by customer_id, so_id, invoice_date,
                  currency_code_conv
         order by so_id asc;

  ENDMETHOD.
```

11. Implement the method `get_customer_info` as AMDP and provide the ABAP dictionary tables `SNWD_BPA` and `SNWD_AD`, as well as the AMDP `ZCL_CUSTOMER_OPEN_INVOICES=>get_curr_conv_relevant_items` in the USING clause.

    In the method body, the AMDP is called and the result is retrieved into the temporary table `lt_converted_items`.

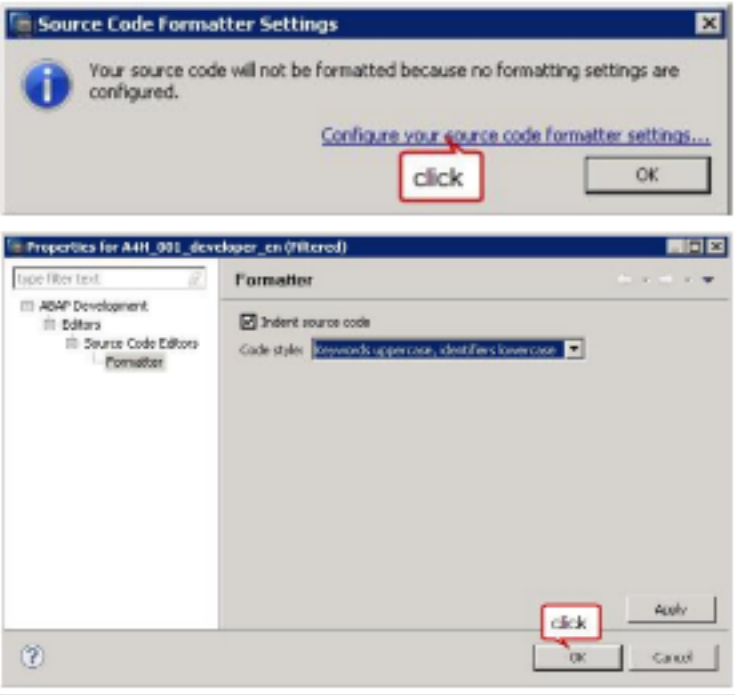
    Then all gross amounts for the given customer are aggregated and additional information on the customer address and name is added.

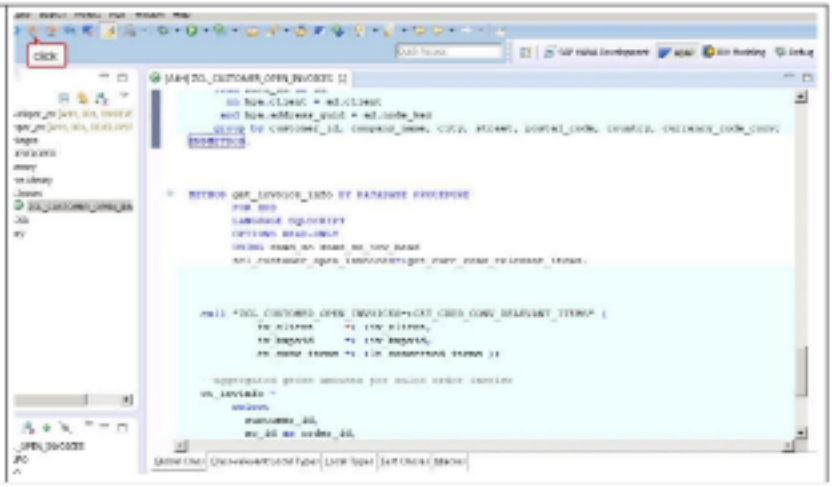
    The result set of the query is return via the exporting table `et_bpinfo`.

Code snippet: `get_customer_info`

```
METHOD get_customer_info BY DATABASE PROCEDURE
          FOR HDB
          LANGUAGE SQLSCRIPT
          OPTIONS READ-ONLY
          USING snwd_bpa snwd_ad
          zcl_customer_open_invoices=>get_curr_conv_relevant_items.

    call
"ZCL_CUSTOMER_OPEN_INVOICES=>GET_CURR_CONV_RELEVANT_ITEMS" (
            iv_client      => :iv_client,
            iv_bupaid      => :iv_bupaid,
            et_conv_items => :lt_converted_items );

    --aggregated gross amounts per customer
    et_bpinfo =
      select
         customer_id,
         bpa.company_name as customer_name,
         ad.city,
         ad.street,
         ad.postal_code,
         ad.country,
         conv_items.currency_code_conv as currency_code,
         sum( conv_items.gross_amount ) as sum_gross_amount
       from :lt_converted_items as conv_items
       join snwd_bpa as bpa
         on conv_items.client = bpa.client
        and buyer_guid        = bpa.node_key
       join snwd_ad as ad
         on bpa.client = ad.client
        and bpa.address_guid = ad.node_key
       group by customer_id, company_name, city, street,
                postal_code, country, currency_code_conv;
  ENDMETHOD.
```

12. Before we activate the class, I think it's a good time to run the pretty printer (short cut `Shift+F1`). As in the good-old SAPGui you have to define the project specific pretty printer settings – or in the ADT language the formatter settings. Navigate to configure the settings and configure the settings of your choice.



13. Activate the class (shortcut `Shift+F3`) or via the matchstick icon.	

Congrats, you just finished the second major step of this end-to-end development guide; you defined and implemented ABAP managed database procedures! Now get yourself ready for the consumption of the CDS views and AMDPs in a SAP NetWeaver Gateway OData service… our next step towards the Fiori-like application.

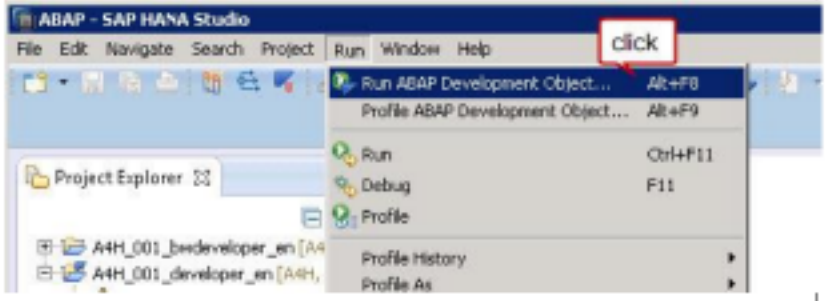
7 Gateway OData Service

SAP NetWeaver Gateway will play the role of OData service provider in this end-to-end guide. OData is a standardized protocol, which is used for the communication between the AS ABAP backend system and the Fiori-like application running in the browser of your PC or Mobile device.

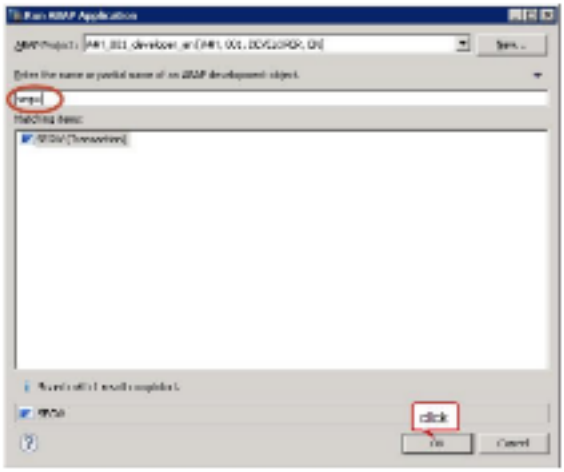
As of SAP NetWeaver release 7.4 SP2, you don't need an additional plugin to use the Gateway functionality but we have a fully functional Gateway "on board" ☺.

In the following, you'll develop the OData service `ZE2E_CUST_INFO` featuring three entity sets:

- CustomerClassifications, consuming the CDS view `ZCDSV_CUST_CLASSIFICATION`
- CustomerInfos, consuming the AMDP `get_customer_info` of class `ZCL_CUSTOMER_OPEN_INVOICES`
- OpenInvoices, consuming the AMDP `get_invoice_info` of class `ZCL_CUSTOMER_OPEN_INVOICES`

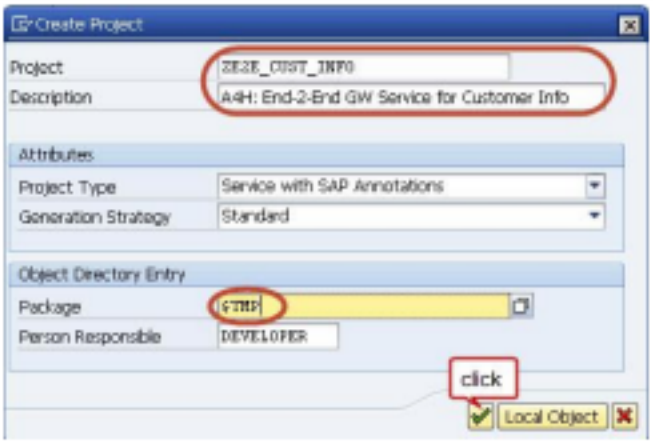
Description	Screen Shot
1. The OData service is developed in the Gateway service builder transaction (transaction code `SEGW`). In the ABAP perspective of the SAP HANA studio, navigate to *Run > Run ABAP Development Object* (short cut `Alt+F8`)	

2. In the Run ABAP Application dialog, filter for "SEGW" and continue.

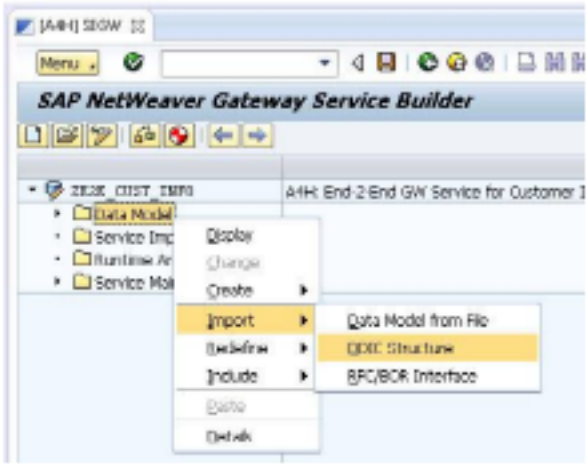


3. An embedded SAPGui opens. Create a new project.

4. In the Create Project dialog, provide
   - Name: `ZE2E_CUST_INFO`
   - Description: A4H: End-2-End GW Service for Customer Info
   - Package: `$TMP`



5. Right-click on the Data Model and select *Import > DDIC Structure* from the context menu.

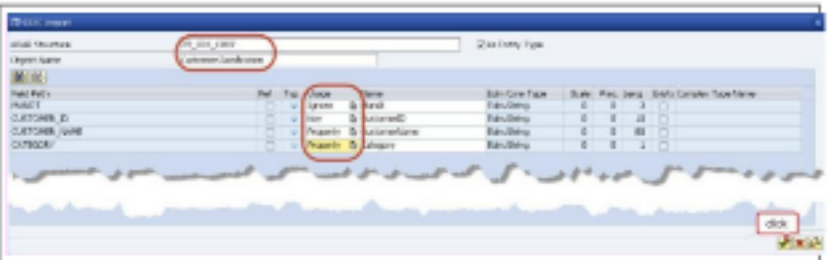


6. In the DDIC Import dialog, choose `ZV_CDS_CUST` as ABAP Structure and hit return.

   Change the Object Name to **CustomerClassification**.

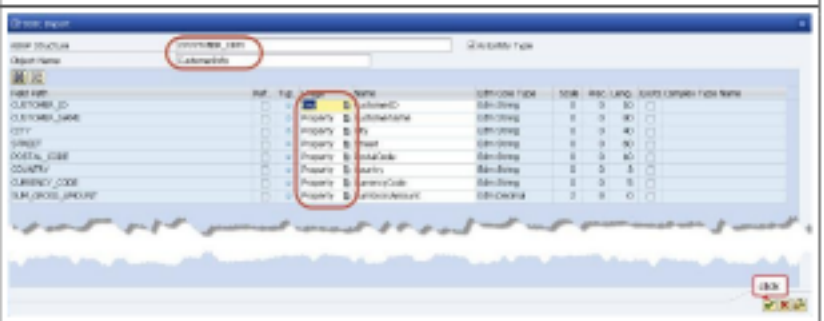
   You see a list of fields and a proposed Usage tyes, which should be adjusted like
   - `MANDT` as Ignore
   - `CUSTOMER_ID` as Key
   - `CUSTOMER_NAME` and `CATEGORY` as Property

   Note: The DDIC view `ZV_CDS_CUST` is the SQL view corresponding to the CDS view



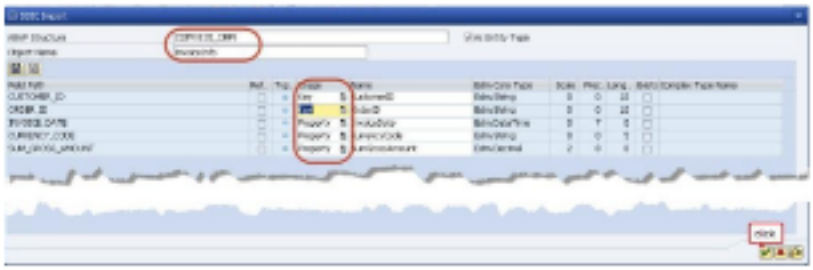
7. Repeat step 5 (*Data Model > Import > DDIC Structure*). In the DDIC Import dialog, choose `ZCUSTOMER_INFO` as ABAP Structure and hit return.

   Change the Object Name to **CustomerInfo** and verify the usage of all fields is set to Property, except for `CUSTOMER_ID`, which is used as Key.

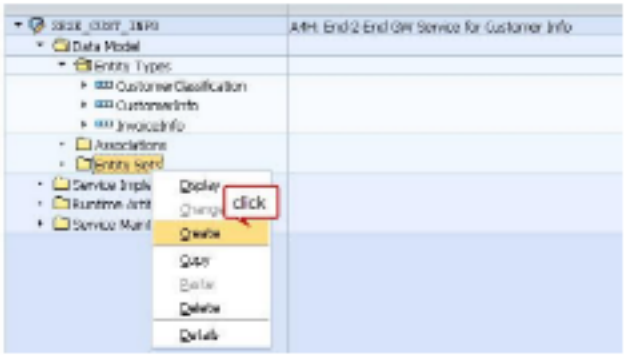


8. Repeat step 5 (*Data Model > Import > DDIC Structure*). In the DDIC Import dialog, choose `ZINVOICE_INFO` as ABAP Structure and hit return.

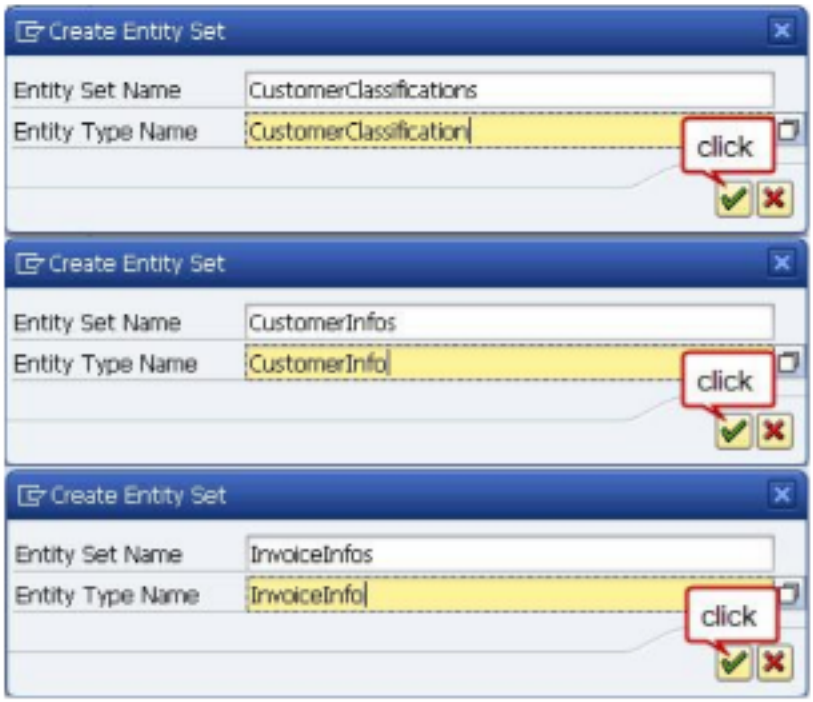
   Change the Object Name to **InvoiceInfo** and verify the usage of all fields `CUSTOMER_ID` and `ORDER_ID` are set Key and the Property is set for all other fields.



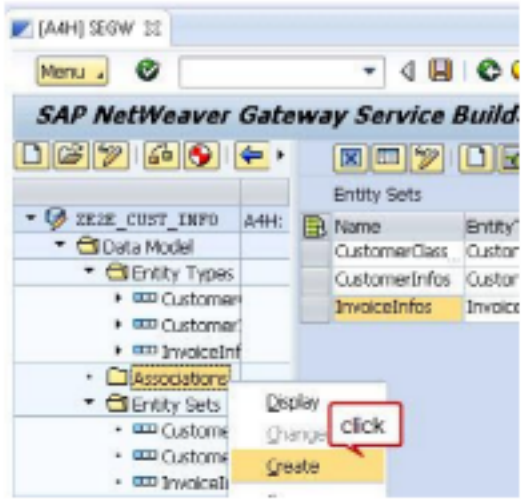
9. Via the context menu of *Data Model* > *Entity Sets*, create the entity sets corresponding to the entity types defined in the previous steps.



10. Create the Entity Sets:
    - CustomerClassifications based on CustomerClassification
    - CustomerInfos base on CustomerInfo
    - InvoiceInfos based on InvoiceInfo

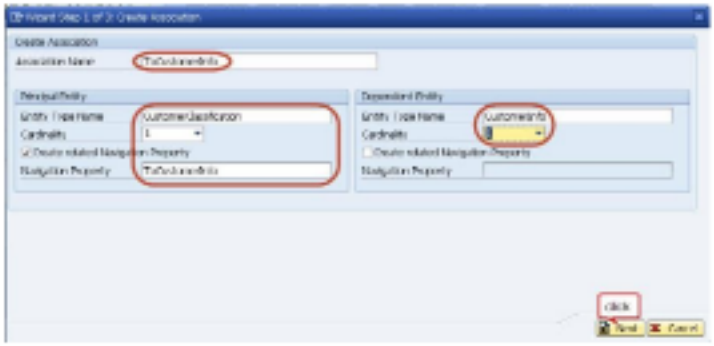


11. The relations between entity are defined in associations. Select Create from the context of *Data Model* > *Associations*.

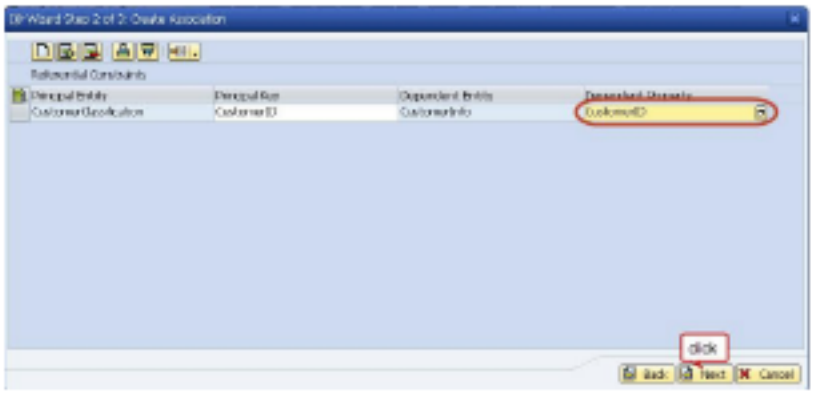


12. The CustomerClassification is related with the CustomerInfo with cardinality 1:1.

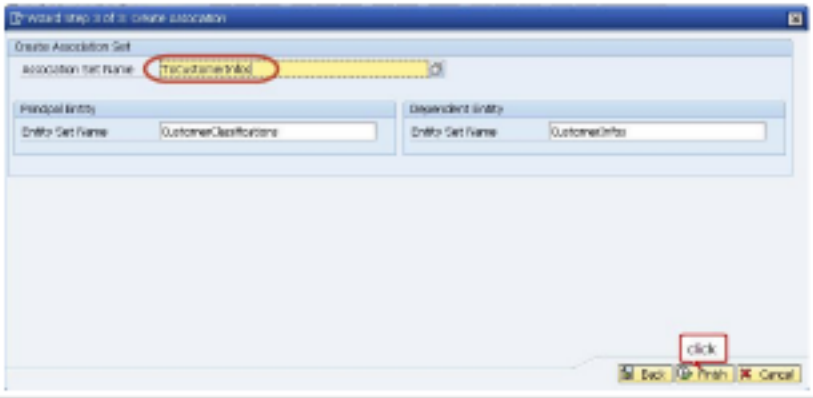
    Create the association ToCustomerInfo via the Create Association wizard. The principal Entity is CustomerClassification with cardinality 1, which should additionally obtain the navigation property ToCustomerInfo. The dependent entity is CustomerInfo with cardinality 1 as well.



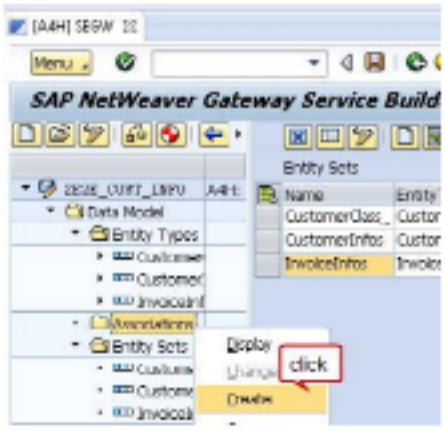
13. In the second step, specify the dependent property `CustomerID` of entity CustomerInfo



14. Finally, set the Association Set name to ToCustomerInfos and click *Finish*.

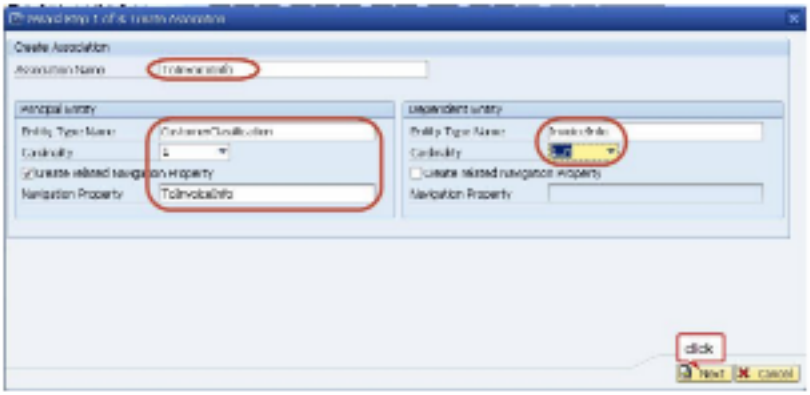


15. Repeat step 11 to create a second association (Select *Create* from the context of *Data Model > Associations*)

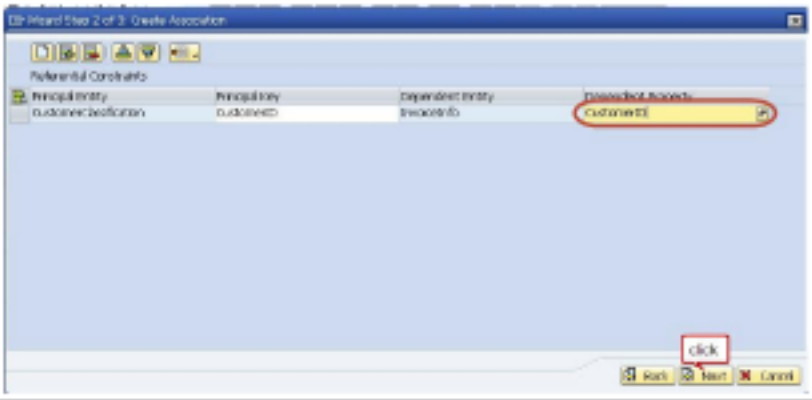


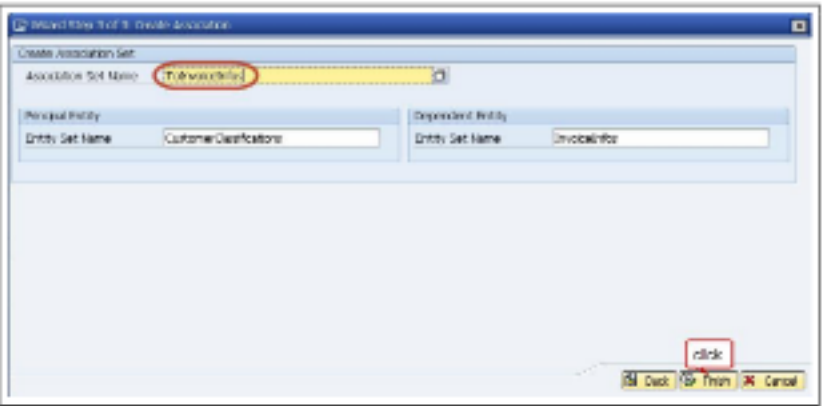
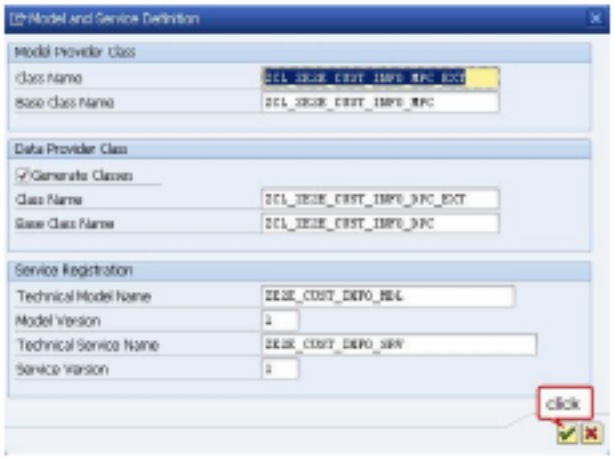
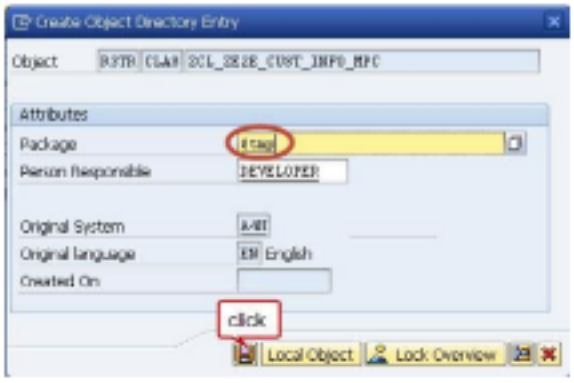
16. We'll need a 1:N relation between CustomerClassification and the InvoiceInfo.

    Create the association ToInvoiceInfo via the Create Association wizard. The principal Entity is CustomerClassification with cardinality 1, which should additionally obtain the navigation property ToInvoiceInfo. The dependent entity is InvoiceInfo with cardinality 1…n.

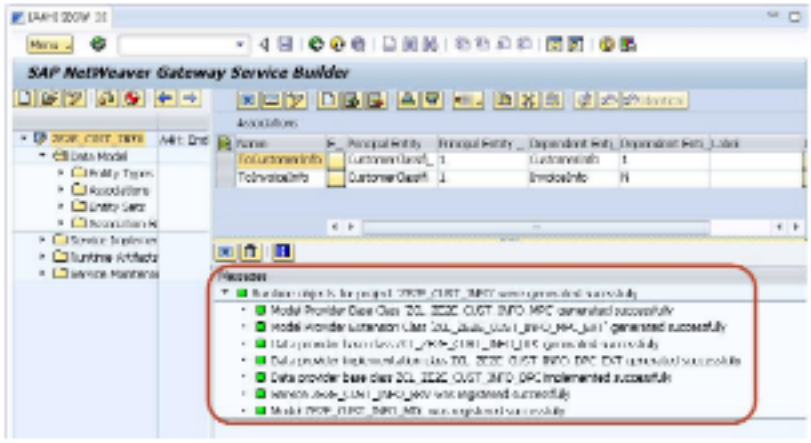


17. In the second step, specify the dependent property `CustomerID` of entity InvoiceInfo.



18. Finally, set the Association Set name to ToInvoiceInfos and click *Finish*.	
19. Save the project and generate the runtime objects via the generate button.	
20. You receive a list of model and service definitions, just continue with the check-mark button.	
21. In the Create Object Directory Entry specify package `STMP` and save.	

22. If the generation has been successful, you see the messages as shown in the screen shot.



23. Now you're finally ready for the implementation of the OData services.

Good news here is, that we only need to do this manually for two of the three entity sets, while we'll use SADL to create the necessary services for the CustomerClassification based on the CDS view.

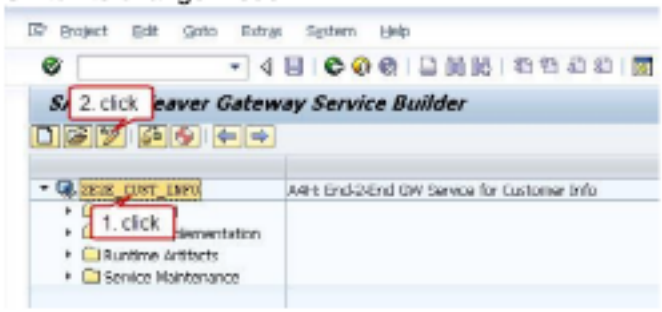
Bad news is, that Windows does not like too many windows inside windows (what a contradiction ☹)… therefore we'll doing a trick! Please switch the gateway project to display mode (pencil icon), open another internal mode (via `/oSEGW` in the `OKCode` field), select your project in this other internal mode and switch to change mode (pencil icon).

Procedure corresponding to the "bad news" part ☹:

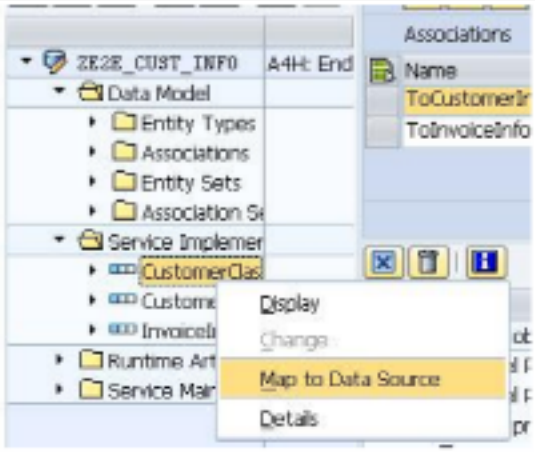
Switch to display mode:

Open a second internal mode in a "detached" SAP GUI window:

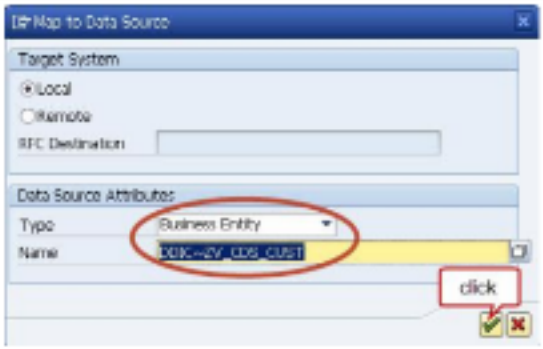
Switch to change mode:



24. Select *Map to Data Source* from the context menu of the *Service Implementation > CustomerClassification.*

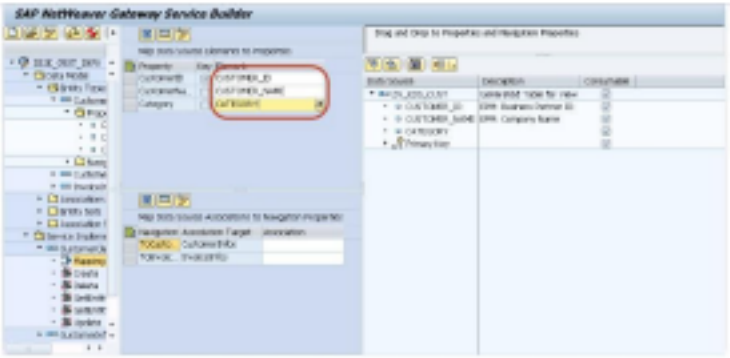


25. Select *Business Entity* as Type and `DDIC~ZV_CDS_CUST` as Name.



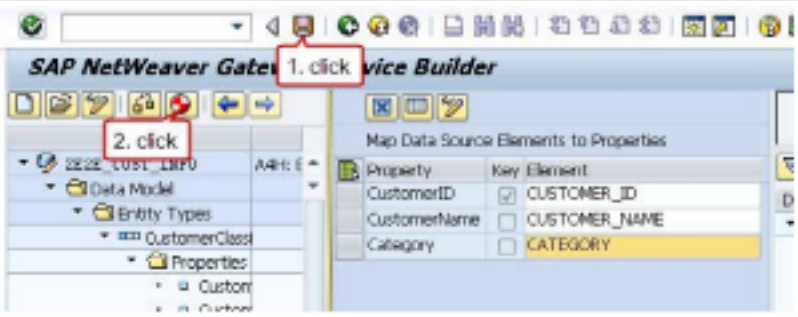
26. Provide the Elements mapping to the properties.

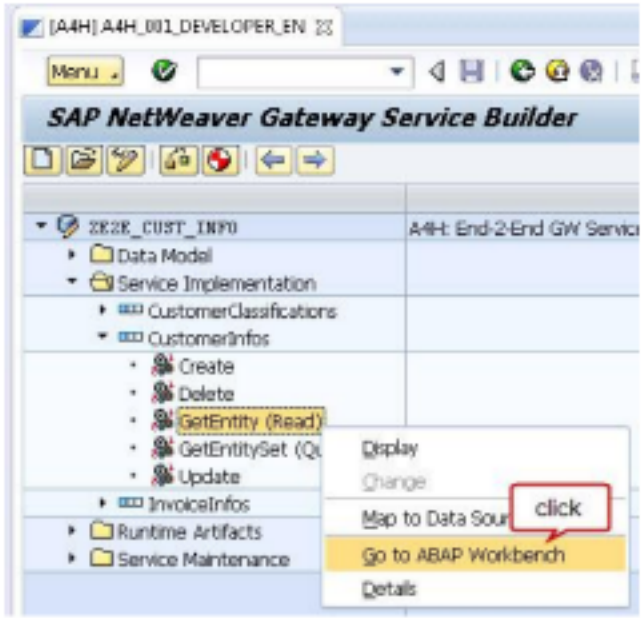
Note: This information is case sensitive, so please use capital letters or just use the `F4` help to select the elements from the given list.



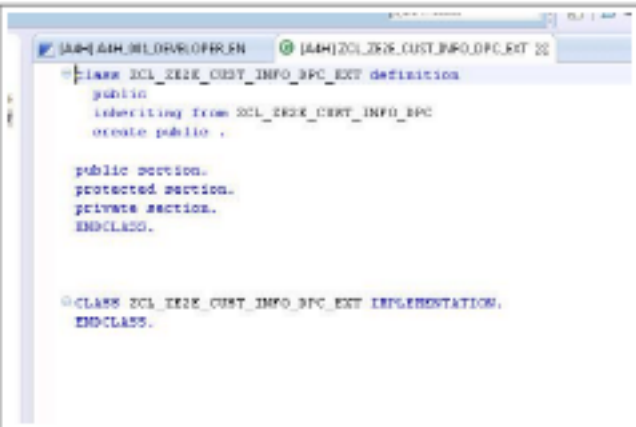
27. Save the project and (re-)generate the runtime objects.

This generates the necessary getter method. If you're interested on the implementation details, have a look at the methods `CUSTOMERCLASSIFI_GET_ENTITY` and `CUSTOMERCLASSIFI_GET_ENTITYSET` of class `ZCL_ZE2E_CUST_INFO_DPC` (use shortcut `Ctrl+Shift+A`



to open the class or have a look at the classes in the `$TMP` package) **Note:** You could now savely switch back to the gateway project tab in your HANA Studio (no more need for the additional internal mode created in step 23), but you can as well continue with the "detachted" SAPGui window.	
28. We need two more implementations, i.e. the `GetEntity` method for CustomerInfos and the `GetEntitySet` method for InvoiceInfos. Navigate to the implementations, via select *Go to ABAP Workbench* from the context menu of Service *Implementation > CustomerInfos > GetEntitySet (Query)*	
29. Confirm the information that you didn't implement the method yet.	

30. The ABAP class editor opens including the class definition and implementation skeletons for class `ZCL_ZE2E_CUST_INFO_DPC_EXT`, which you'll enhance with the two missing implementations in the following steps.

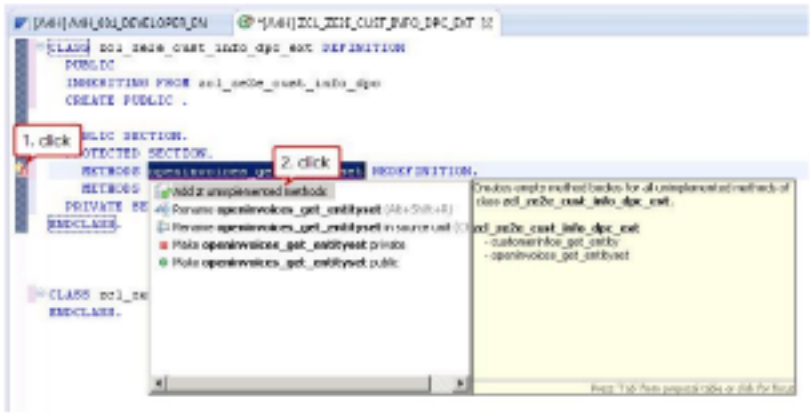


31. In the protected section of the class, define the two methods as redefinitions.

Code snippet

```
PROTECTED SECTION.
   METHODS invoiceinfos_get_entityset REDEFINITION.
   METHODS customerinfos_get_entity   REDEFINITION.
```

32. Use the quick-fix to add the unimplemented methods, i.e. click on the light-bulb icon and choose *Add 2 unimplemented methods*.



33. Implement the `customerinfos_get_entity` method:

Code snippet: `customerinfos_get_entity`

```
METHOD customerinfos_get_entity.

    "Entity can only be accessed via the navigation property ToCustomerInfo
    "of entity type CustomerClassification and if the key CustomerID is provided
    IF iv_source_name <> zcl_ze2e_cust_info_mpc=>gc_customerclassification.
      RAISE EXCEPTION TYPE /iwbep/cx_mgw_tech_exception.
    ENDIF.
    READ TABLE it_key_tab INTO DATA(ls_key) WITH KEY name = 'CustomerID' ##NO_TEXT.
    IF sy-subrc <> 0.
      RAISE EXCEPTION TYPE /iwbep/cx_mgw_busi_exception
        EXPORTING
          textid = /iwbep/cx_mgw_busi_exception=>filter_not_supported.
    ENDIF.

    DATA lv_customer_id_filter TYPE snwd_partner_id.
    "Apply the conversion of snwd_partner_id (domain D_EPM_ID)
    CALL FUNCTION 'CONVERSION_EXIT_ALPHA_INPUT'
      EXPORTING
        input  = ls_key-value
```

```
      IMPORTING
        output = lv_customer_id_filter.

    "create an instance of class zcl_customer_open_invoices
    DATA(lo_cust_info) = NEW zcl_customer_open_invoices( ).

    "call the ABAP managed DB procedure respectively the
    "ABAP class method get_customer_info
    lo_cust_info->get_customer_info(
      EXPORTING
        iv_client   = sy-mandt
        iv_bupaid   = lv_customer_id_filter
      IMPORTING
        et_bpinfo =   DATA(lt_entity)
    ).

    "only one line is expected to be retrieved
    IF lines( lt_entity ) <> 1.
      RAISE EXCEPTION TYPE /iwbep/cx_mgw_busi_exception
        EXPORTING
          message_unlimited = | Problem in Customer Info; { lines( lt_entity ) } <> 1 |
          textid            = /iwbep/cx_mgw_busi_exception=>business_error_unlimited.
    ENDIF.

    "map the first row of lt_entity to the output
    er_entity = lt_entity[ 1 ].
  ENDMETHOD.
```

34.Implement the invoiceinfos_get_entityset method:

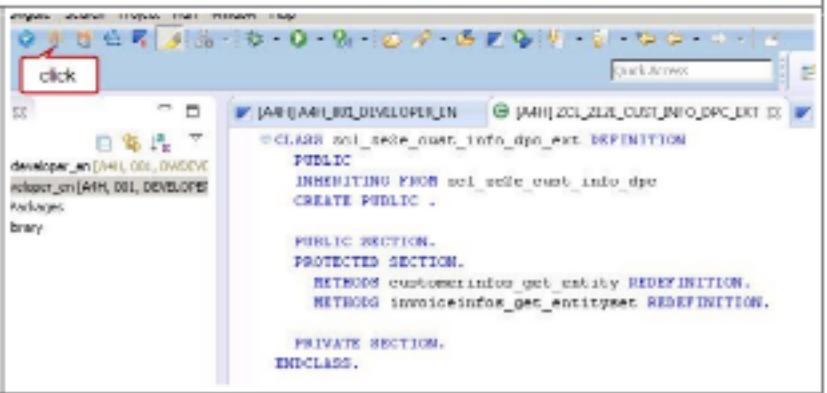
Code snippet: invoiceinfos_get_entityset

```
  METHOD invoiceinfos_get_entityset.

    "Entityset can only be accessed via the navigation property ToInvoiceInfo
    "of entity type CustomerClassification and if the key CustomerID is provided
    IF iv_source_name <> zcl_ze2e_cust_info_mpc=>gc_customerclassification.
      RAISE EXCEPTION TYPE /iwbep/cx_mgw_tech_exception.
    ENDIF.
    READ TABLE it_key_tab INTO DATA(ls_key) WITH KEY name = 'CustomerID' ##NO_TEXT.
    IF sy-subrc <> 0.
      RAISE EXCEPTION TYPE /iwbep/cx_mgw_busi_exception
        EXPORTING
          textid = /iwbep/cx_mgw_busi_exception=>filter_not_supported.
    ENDIF.

    DATA lv_customer_id_filter TYPE snwd_partner_id.
    "Apply the conversion of snwd_partner_id (domain D_EPM_ID)
    CALL FUNCTION 'CONVERSION_EXIT_ALPHA_INPUT'
      EXPORTING
        input  = ls_key-value
      IMPORTING
        output = lv_customer_id_filter.

    "create an instance of class zcl_customer_open_invoices
    DATA(lo_cust_info) = NEW zcl_customer_open_invoices( ).

    "call the ABAP managed DB procedure respectively the
    "ABAP class method get_invoice_info
    lo_cust_info->get_invoice_info(
      EXPORTING
```

```
        iv_client   = sy-mandt
        iv_bupaid   = lv_customer_id_filter
      IMPORTING
        et_invinfo =   DATA(lt_entityset)
    ).

    "provide the resultset of the AMDP to the tabular
    "tabular output parameter
    et_entityset = lt_entityset.
  ENDMETHOD.
```

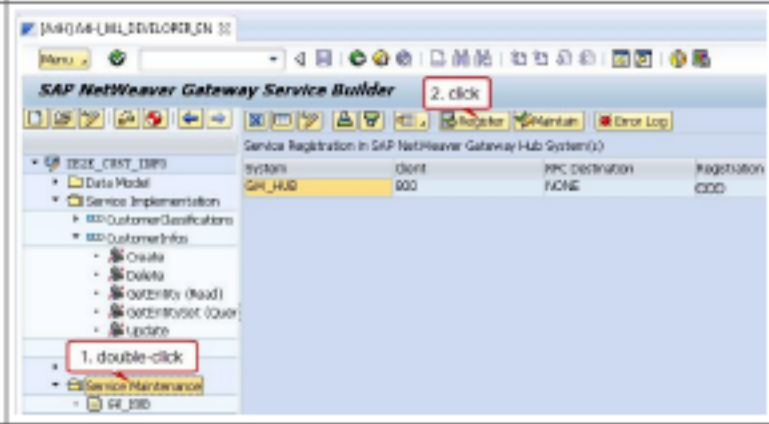
35. Save and activate the class (shortcuts `Ctrl+S`, `Ctrl+F3`).

    If you'd like to format / apply the pretty printer remember shortcut `Shift+F1` 😉.

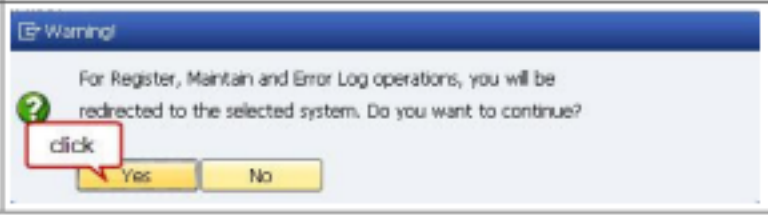


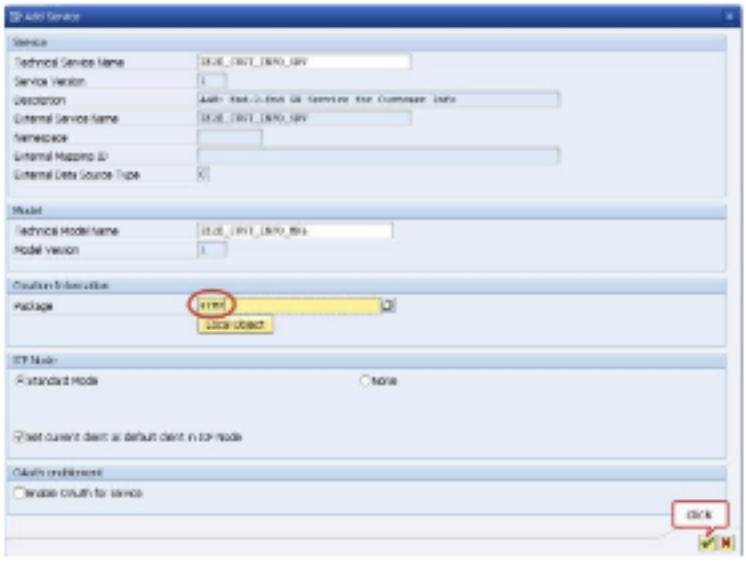
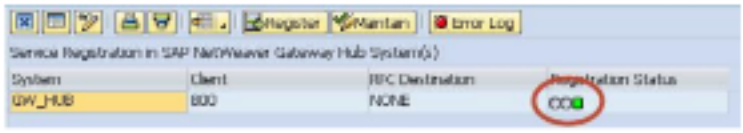
36. Last step is to register the OData service. Go back to transaction `SEGW` (you might still have the tab open or use `Alt+F8` to open it again).

    Double-click on *Service Maintainance* and then on *Register*.



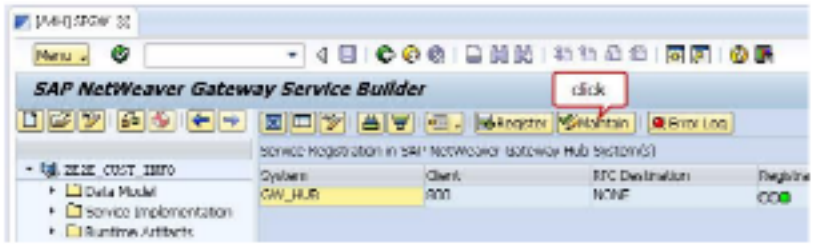
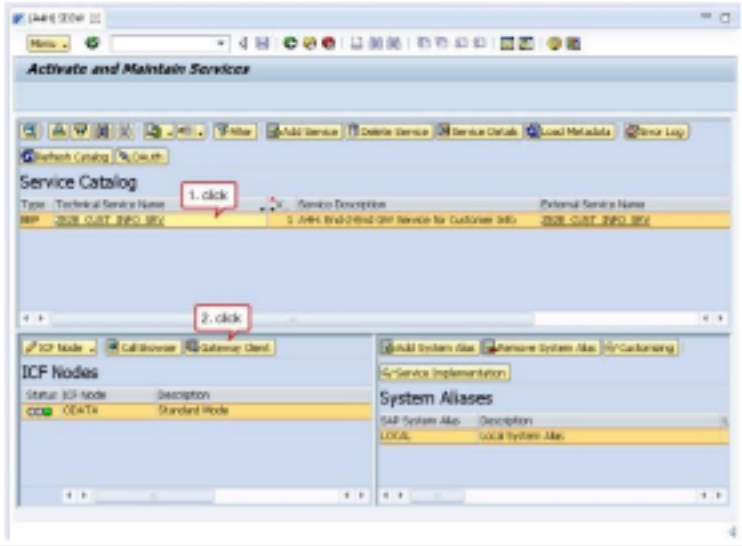
37. Confirm the Warning.



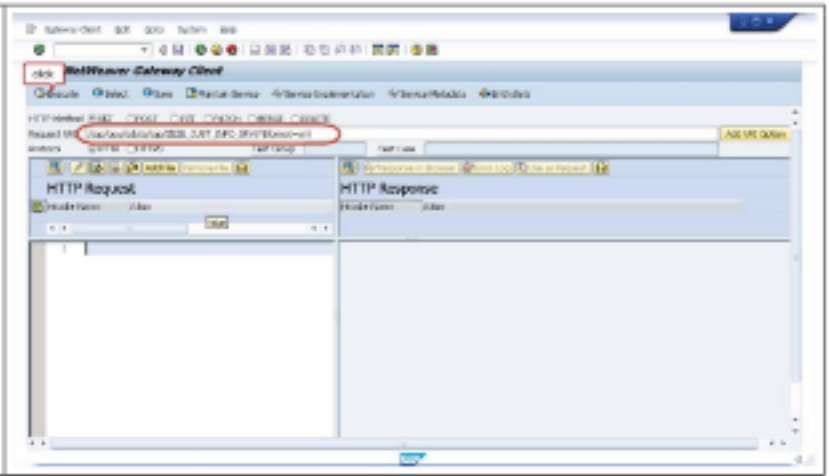
38. Provide the package `$TMP` to which the OData service will be added.	
39. If everything went fine you can see the registration status indicated by the green status indicator.	

Would you like to test the GW service?

If yes, this can be done in several ways. I'll tell you how to use the Gateway client (transaction `/IWFND/GW_CLIENT`).

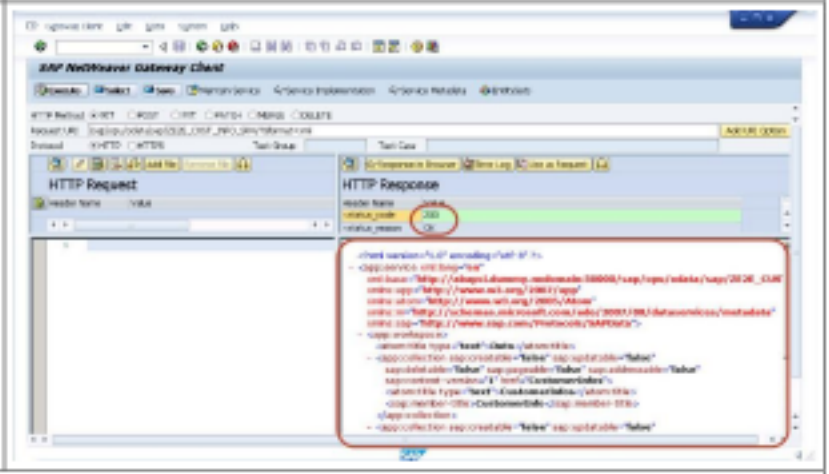
Description	Screen Shot
1. One option how you can navigate to the gateway client transaction is to click on the *Maintain* icon	
2. In the service maintainance display, select the service from the service catalog and click on the Gateway Client button in the lower right part of the view.	
3. Confirm the SAP GUI security information dialog (maybe even with the remember my decision functionality ☺).	

4. Check the Request URI (`/sap/opu/odata/ZE2E_CUST_INVO_SRV/?$format=xml`) and execute (`F8`)



5. Several more SAP GUI Security information windows appear. Read them carefully 😳 or just click allow (and now, please remember my decision!)

6. If everything went fine, you should see a successful HTTP Response (status code 200) and the xml result of the OData request.



7. Repeat the test for the Request URIs:
   - `/sap/opu/odata/sap/ZE2E_CUST_INFO_SRV/$metadata`
     Metadata of the OData service
   - `/sap/opu/odata/sap/ZE2E_CUST_INFO_SRV/CustomerClassifications`
     List of Customer Classifications, i.e. query on the CDS view (get entity set method of CustomerClassification)
   - `/sap/opu/odata/sap/ZE2E_CUST_INFO_SRV/CustomerClassifications('100000000')/ToCustomerInfo`
     Query the customer information for customer ID 100000000 (SAP AG), technically executing the customer information AMDP
   - `/sap/opu/odata/sap/ZE2E_CUST_INFO_SRV/CustomerClassifications('100000000')/ToInvoiceInfo`
     Query the list of sales order invoice information for customer ID 100000000 (SAP AG), technically executing the invoice information AMDP

Well done, you finished the implementation and testing of the SAP NetWeaver Gateway OData Service and hence the second to last step on our journey to the Fiori-like application.

If you would like to learn more about SAP NetWeaver Gateway and/or contact the experts, please visit http://scn.sap.com/community/netweaver-gateway.

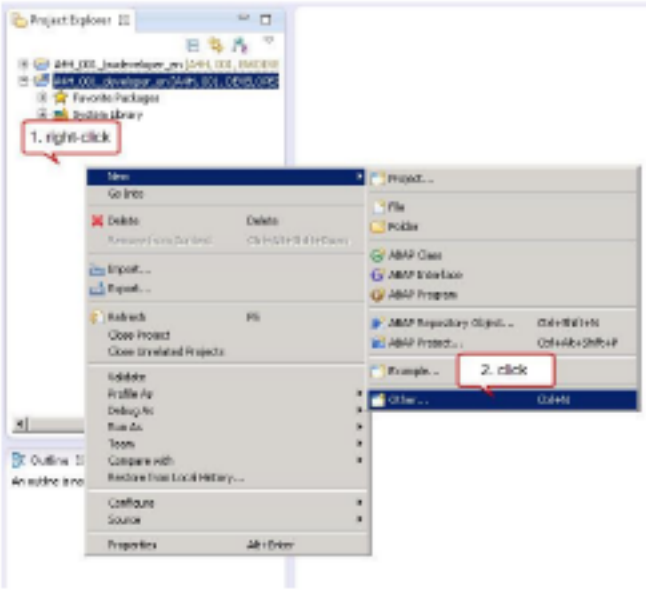
8 Fiori-like Application

Ready for the final step? You'll now create a Fiori-like application to display the list of Customer Classification Information and to do a drill-down on the detailed customer and open sales order information.

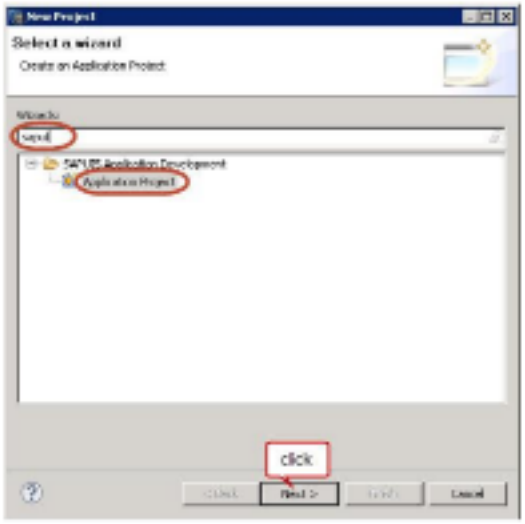
Technically, the Fiori-like application is a so-called split application (sap.m.SplitApp - see https://sapui5.netweaver.ondemand.com/sdk/#content/Overview.html for details on available Controls, API references, etc.) targeted for mobile devices.

In the following, you'll create and develop SAPUI5 application locally, where locally means you develop a SAPUI5 application in you Eclipse installation. The local SAPUI5 application will then be imported to the ABAP backend using the SAPUI5 repository team provider. Having provided the application to the ABAP backend, you're ready to run and test your Fiori-like application in the browser.

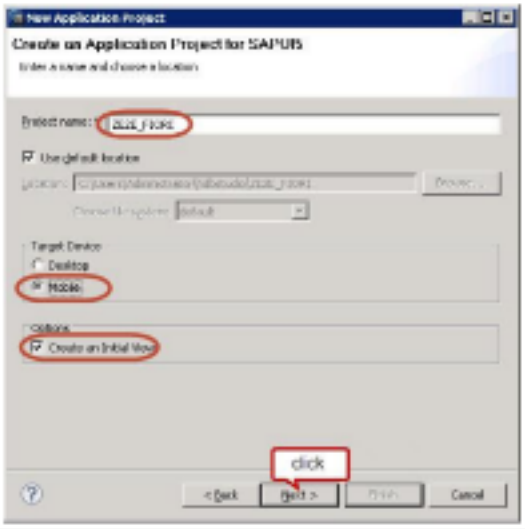
Local SAPUI5 Application Development

Description	Screen Shot
1. Right-click in the Project Explorer view and select *New > Other* from the context menu.	

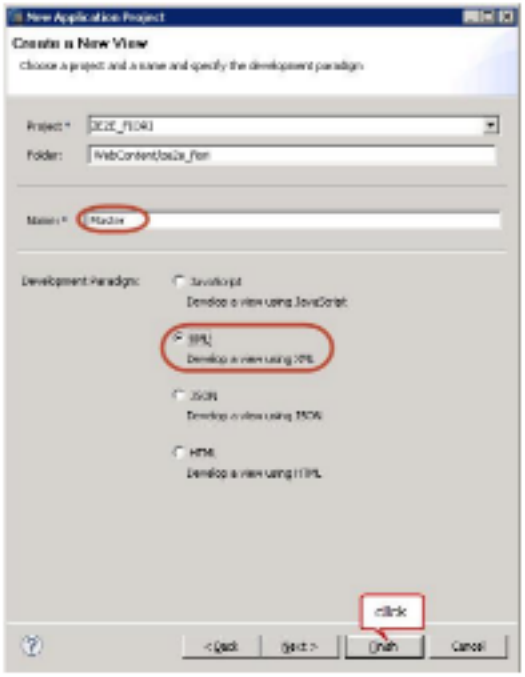
2. In the new project wizard, select SAPUI5 *Application Development > Application Project* (to filter for e.g. sapui eases the search for the corresponding entry).



3. Provide the necessary information for the new application project:
   - Project name: `ZE2E_FIORI`
   - Target Device: Mobile
   - Options: Create an Initial View (checkbox)

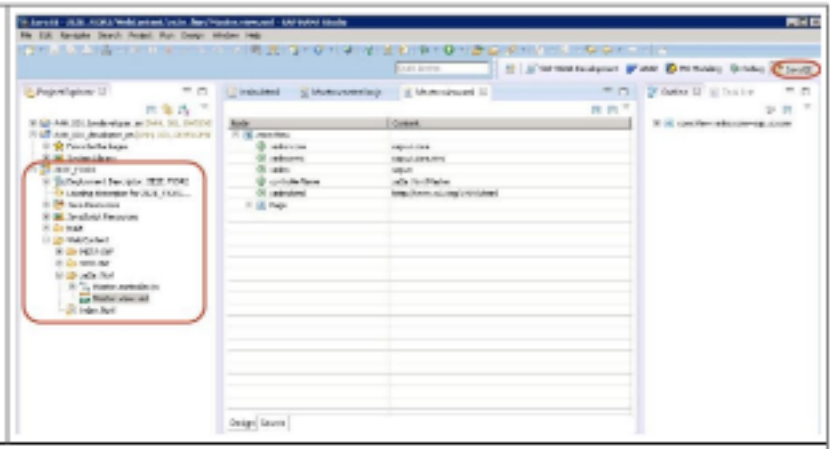


4. Provide the information for the initial view:
   - Name: Master
   - Development Paradigm: XML (radio-button)



5. The SAPUI5 develoment project is created.

   You're redirected to the Java EE perspective and the three development artefacts are opened, namely the `index.html`, the `Master.controller.js`, and the `Master.view.xml`



6. Implement the `index.html` file (best thing is to copy/paste everything 😉):

Code snippet: `index.html`

```
<!DOCTYPE HTML>
<html>
<head>
<meta http-equiv="X-UA-Compatible" content="IE=edge">

<script src="resources/sap-ui-core.js"
        id="sap-ui-bootstrap"
        data-sap-ui-libs="sap.m, sap.me, sap.ui.layout"
        data-sap-ui-xx-bindingSyntax="complex"
        data-sap-ui-theme="sap_bluecrystal">
</script>

<script>
       // Define path of the OData backend service
       var sODataServiceUrl =
         "/sap/opu/odata/sap/ze2e_cust_info_srv";
       var oODataModel =
         sap.ui.model.odata.ODataModel(sODataServiceUrl);
       oODataModel.attachRequestFailed(function() {
              alert("OData Request Failed!");
       });
       sap.ui.getCore().setModel(oODataModel);

       // Tell the Application where to find the
       // local resources
       sap.ui.localResources("ze2e_fiori");

       // Master & Detail pages
       var oMasterPage = sap.ui.view({
              id : "idMaster1",
              viewName : "ze2e_fiori.Master",
              type : sap.ui.core.mvc.ViewType.XML
       });
       var oDetailPage = sap.ui.view({
              id : "idDetail1",
              viewName : "ze2e_fiori.Detail",
              type : sap.ui.core.mvc.ViewType.XML
       });

       // Split Application
       var oSplitApp = new sap.m.SplitApp();
```

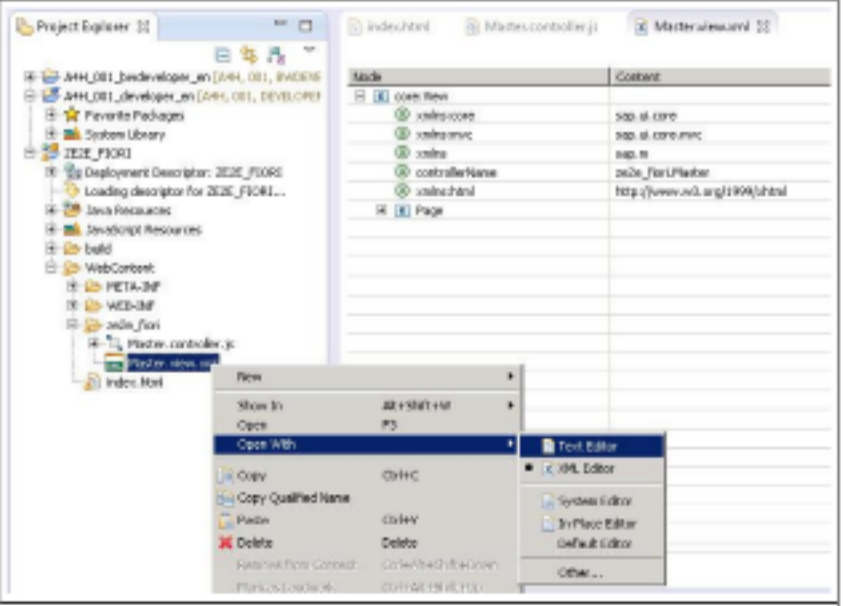
```
        oSplitApp.addMasterPage(oMasterPage);
        oSplitApp.addDetailPage(oDetailPage);
        oSplitApp.placeAt("content");
</script>

</head>
<body class="sapUiBody" role="application">
        <div id="content"></div>
</body>
</html>
```

7. Implement the `Master.controller.js`:

Code snippet: `Master.controller.js`

```
sap.ui.controller("ze2e_fiori.Master", {

        handlePress : function(oEvt) {

                var oContext = oEvt.getSource().getBindingContext();

                if (oContext) {
                        sap.ui.getCore().getEventBus().publish("e2e_app",
                                        "ReadyToFetchDetails", {
                                                oCtx : oContext
                                        });
                }
        }

});
```

8. Open the `Master.view.xml` with a text editor. Right-click on `Master.view.xml` and select *Open With > Text Editor* from the context menu



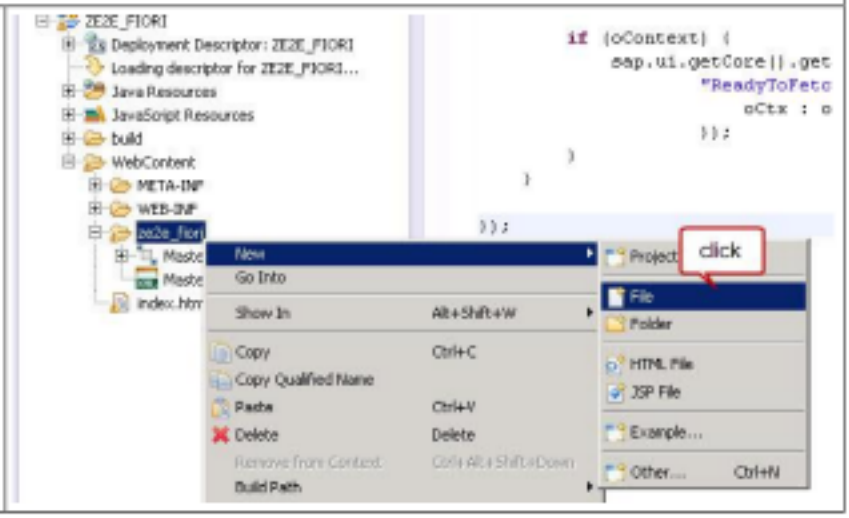
9. Implement the `Master.view.xml`:

Code snippet: `Master.view.xml`

```
<core:View xmlns:core="sap.ui.core"
           xmlns:mvc="sap.ui.core.mvc"
```

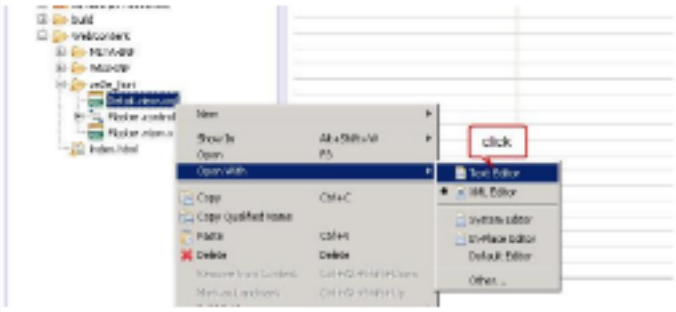
```
          xmlns:m="sap.m"
                controllerName="ze2e_fiori.Master"
                xmlns:html="http://www.w3.org/1999/xhtml">
      <m:Page id="page_Master"
              class="sapUiFioriObjectPage"
              title="Customer List">
    <m:content>

    <m:List
        id="list_CustClassification"
        headerText="Customers"
        growing="true"
        growingThreshold="5"
        items="{/CustomerClassifications}">
        <m:ObjectListItem
          title="{CustomerName}"
          number="{Category}"
          numberUnit="Category"
          press="handlePress"
          type="Active">

          <m:attributes>
            <m:ObjectAttribute text="{CustomerID}" />
          </m:attributes>
        </m:ObjectListItem>

        </m:List>
    </m:content>
  </m:Page>
</core:View>
```

10. In addition to the Master page, which has been created during creation of the SAPUI5 application, a Detail page is needed for the SplitApp. Create a new file via the context menu of the `ze2e_fiori` folder by selecting *New > File*



11. In the New File wizard, provide the file name `Detail.view.xml`

12. Open the `Detail.view.xml` file with the Text Editor via the context menu *Open With > Text Editor*



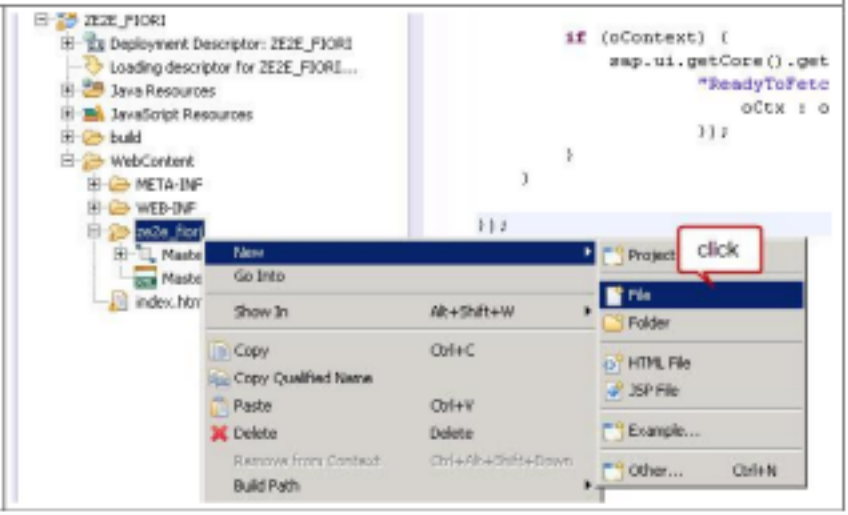
13. Implement the `Detail.view.xml`:

Code snippet: `Detail.view.xml`

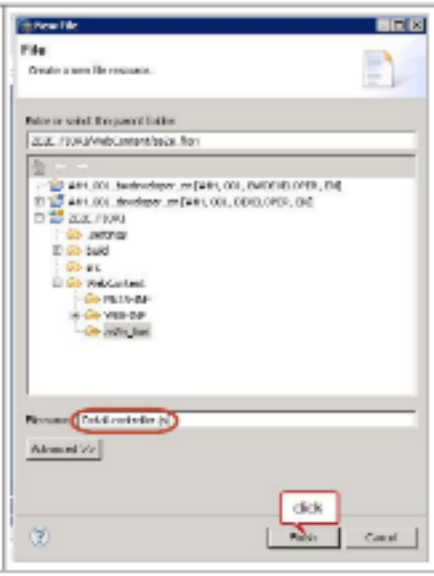
```
<core:View xmlns:core="sap.ui.core"
           xmlns:mvc="sap.ui.core.mvc"
           xmlns="sap.m"
                 controllerName="ze2e_fiori.Detail"
                 xmlns:html="http://www.w3.org/1999/xhtml">
      <Page id="page_Detail"
            class="sapUiFioriObjectPage"
            title="Detailed Customer Information" >
   <content>

   <ObjectHeader
       id="head_Invoices"
      title="{CustomerName}"
      number="{SumGrossAmount}"
       numberUnit="{CurrencyCode}" >
       <attributes>
         <ObjectAttribute text="{CustomerID}" />
         <ObjectAttribute text="{City}" />
         <ObjectAttribute text="{PostalCode}" />
         <ObjectAttribute text="{Street}" />
         <ObjectAttribute text="{Country}" />
       </attributes>
   </ObjectHeader>
```

```
    <!--ToInvoiceInfo in path is navigation property in OData -->
    <Table
        id="tbl_Invoices"
        items="{ path : 'ToInvoiceInfo'}"
        noDataText="No data available - Please select a customer"
        growing="true"
        growingThreshold="20">
        <columns>
          <Column>
            <header><Label text="Order ID" /></header>
          </Column>
          <Column hAlign="Center" >
            <header><Label text="Creation Date" /></header>
          </Column>
          <Column hAlign="Right" >
            <header><Label text="Gross Amount" /></header>
          </Column>
        </columns>
        <items>
          <ColumnListItem>
            <cells>
              <ObjectIdentifier title="{OrderID}"/>
              <Text text="{InvoiceDate}"/>
              <ObjectNumber number="{SumGrossAmount}" numberUnit="{CurrencyCode}" />
            </cells>
          </ColumnListItem>
        </items>
      </Table>
    </content>
  </Page>
</core:View>
```

14. Additionally, a file for the Detail controller is required. Create a new file via the context menu of the `ze2e_fiori` folder by selecting *New > File*



15. In the New File wizard, provide the file name `Detail.controller.xml`



16. Implement the `Detail.controller.js`:

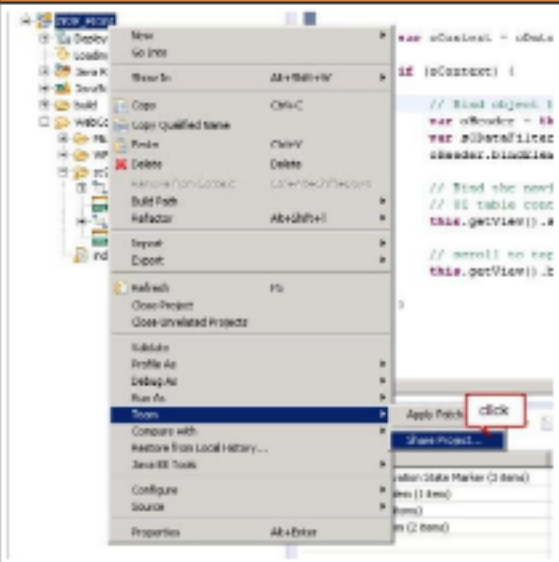
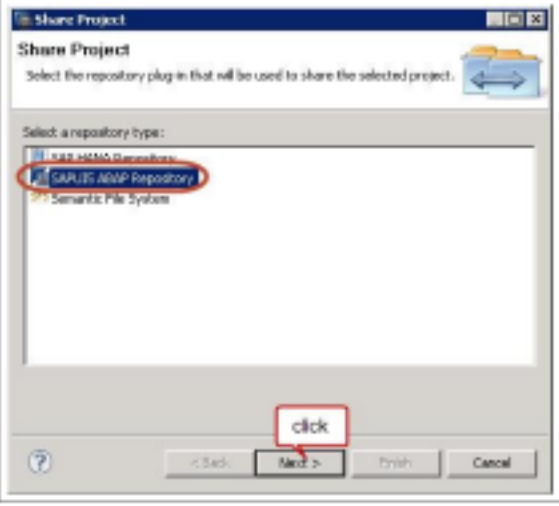
Code snippet: `Detail.controller.js`

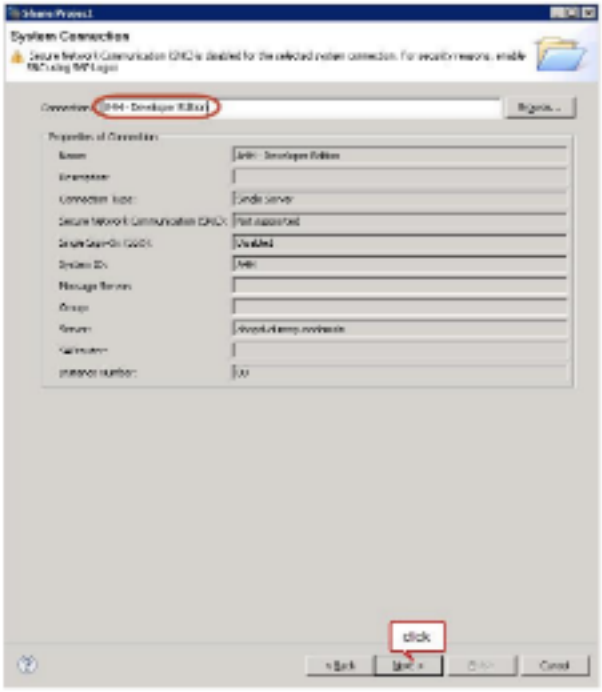
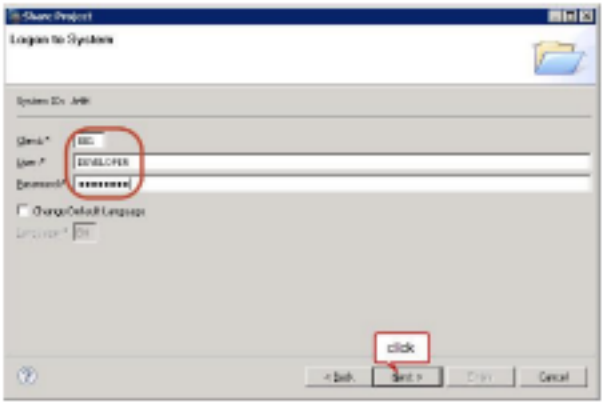
```
sap.ui.controller("ze2e_fiori.Detail", {

	/**
	 * Called when a controller is instantiated and its View controls (if
	 * available) are already created. Can be used to modify the View before it
	 * is displayed, to bind event handlers and do other one-time initialization.
	 * @memberOf ze2e_fiori.Detail
	 */
	onInit : function() {
		// handle data loaded events
		var oBus = new sap.ui.getCore().getEventBus();
		oBus.subscribe("e2e_app", "ReadyToFetchDetails",
				this.handleFetchDetails, this);
	},

	handleFetchDetails : function(sCannelID, sEvtId, oData) {

		var oContext = oData.oCtx;

		if (oContext) {
			// Bind object header to the OData model
			var oHeader = this.byId('head_Invoices');
			var sODataFilterPath = oContext.sPath + '/ToCustomerInfo';
			oHeader.bindElement(sODataFilterPath);

			// Bind the navigation properties of the OData Service to the
			// UI table control
			this.getView().setBindingContext(oContext);

			// scroll to top of page
			this.getView().byId("page_Detail").scrollTo(0);

		}

	},

});
```

17.Save all files	

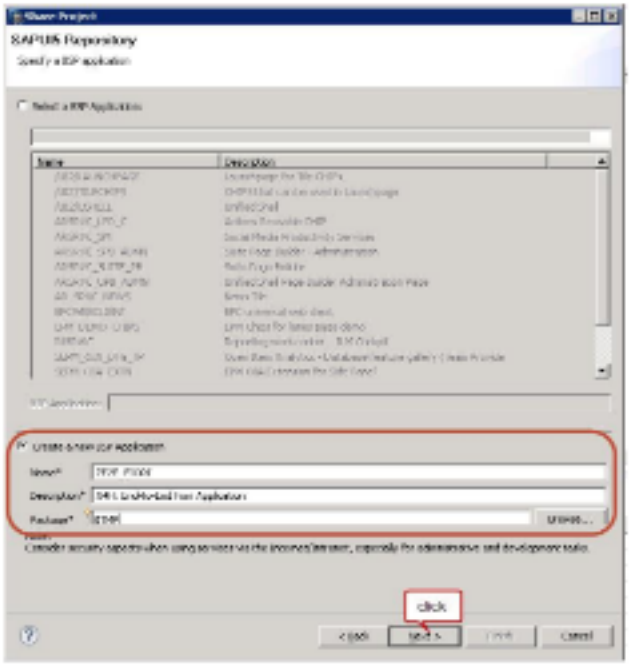
Import of the SAPUI5 Application to the ABAP backend

After local development of the SAPUI5 application, you'll now import the application into the SAPUI5 ABAP Repository (technically a BSP repository) using the SAPUI5 Repository Team Provider.

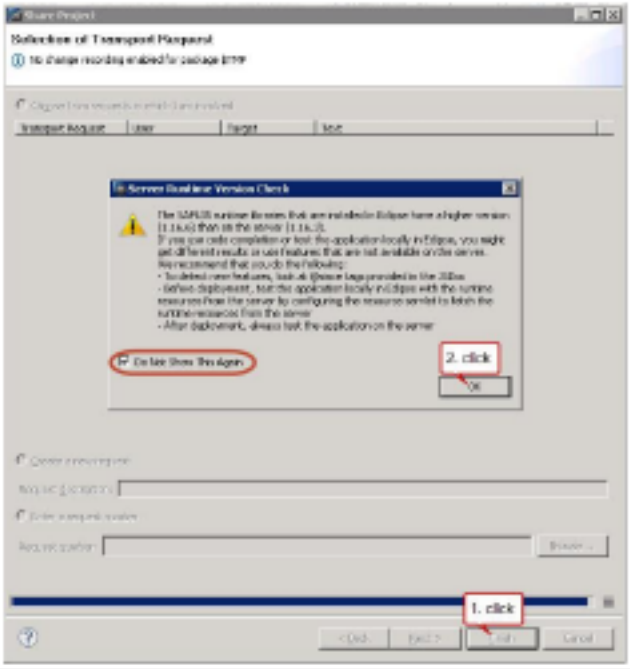
Description	Screen Shot
1. Select *Team > Share Project* from the context menu of the `ZE2E_FIORI` application project	
2. In the Share Project dialog, select *SAPUI5 ABAP Repository*	

3. Select the backend connection, e.g. on the virtual appliance you can choose *A4H – Developer Edition*, i.e. the ABAP backend connection provided in the SAP Logon application **Note:** In case you're not working with a virtual appliance, the connection name may differ.	
4. Provide the system logon credentials	

5. In the Share Project dialog, choose to create a new BSP application and provide:
   - Name: `ZE2E_FIORI`
   - Description: A4H: End-to_end Fiori Application
   - Package: `$TMP`

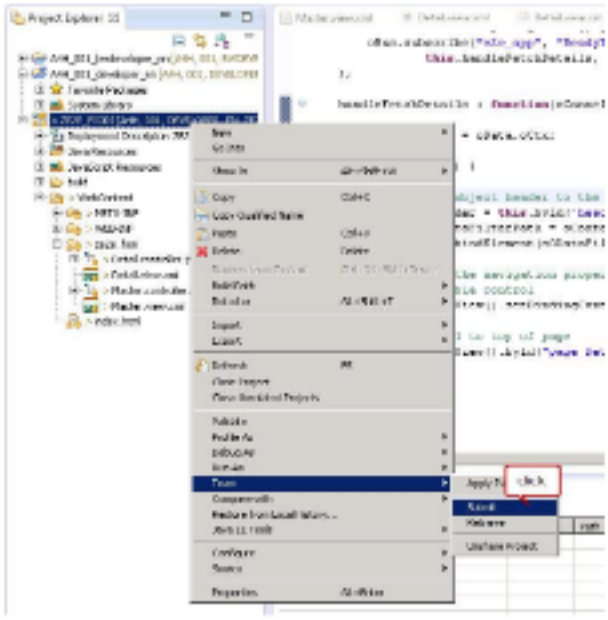


6. The "No transport request" information is shown. Continue with *Finish*, which results in a Server Runtime Version Check information which you can just confirm and continue.

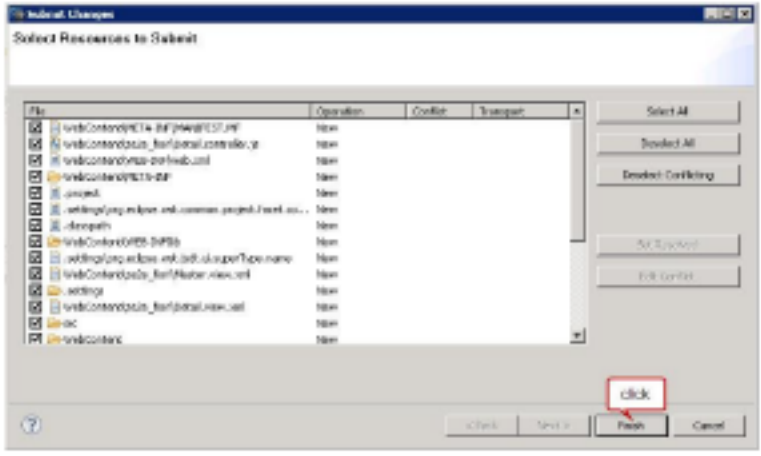


7. You have now prepared the ABAP backend respectively the SAPUI5 ABAP Repository and you can now submit the application to the repository.

   For this purpose, choose *Team* > *Submit* from the context menu of the `ZE2E_FIORI` application project.



8. The Submit Changes dialog shows the list of information that will be submitted to the repository, check the list and continue with Finish.

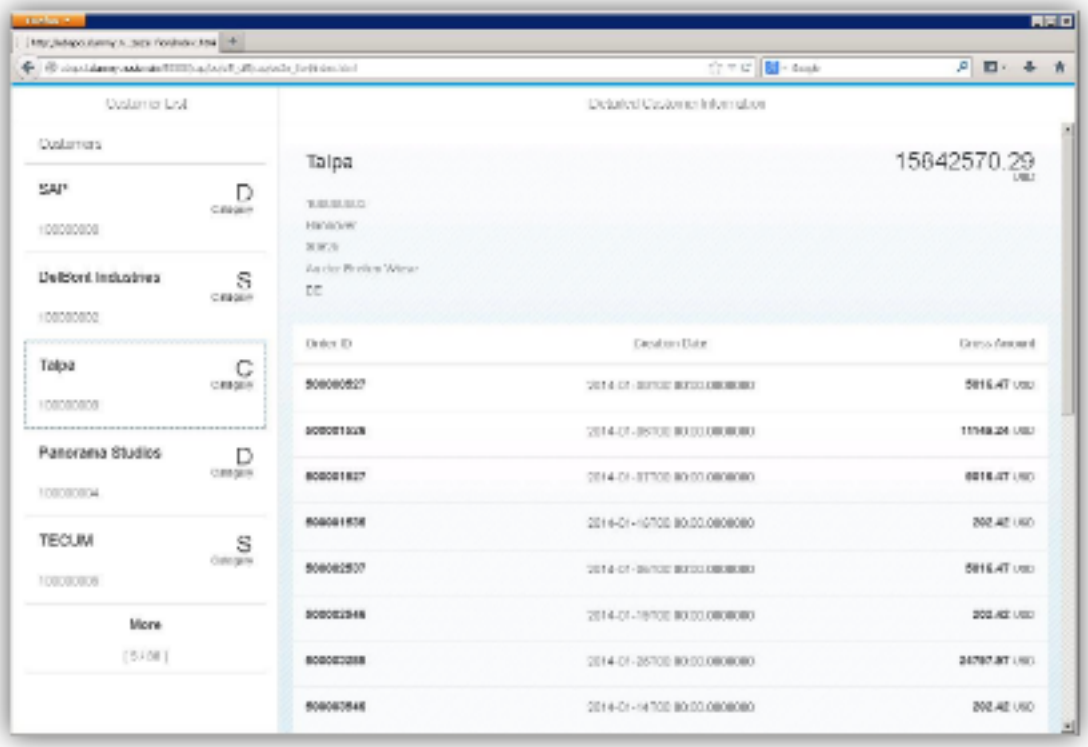


Jippee!

You have successfully created the Fiori-like application! But…

You would like to test the Fiori-like App, right?

If yes, open the browser of your choice (or the one you like best from the installed ones ☺) and open http://abapci.dummy.nodomain:50000/sap/bc/ui5_ui5/sap/ze2e_fiori/index.html. After providing the system logon credentials you should finally see the picture I promised you at the very beginning of the guide (…at least after you selected a customer from the list on the right-hand side of the application!):



Thanks for joining…

…this end-to-end development guide from HANA via ABAP to a SAP Fiori-like application. We hope you liked developing Core Data Services (CDS) Views and ABAP managed database procedures and encourage you to also have a look at the latest features in OpenSQL. With the SAP NetWeaver Gateway as OData provider you stepped towards the user interface and finally developed (okay with a lot of copy-and-paste) a very simple SAP Fiori-like application… feel free to enhance the application to include all the eye-candy and mobile-device-leveraging features!

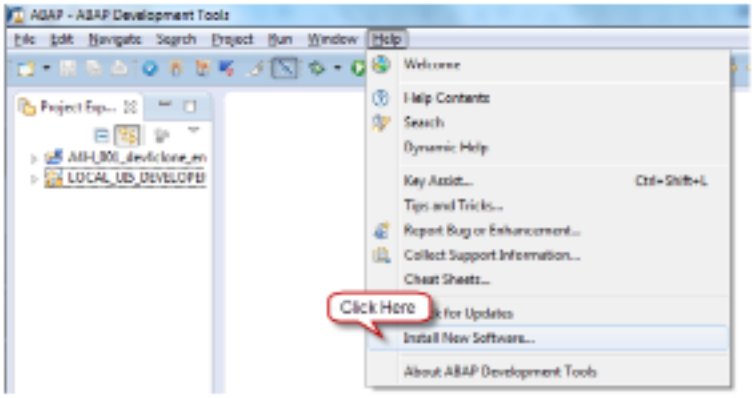
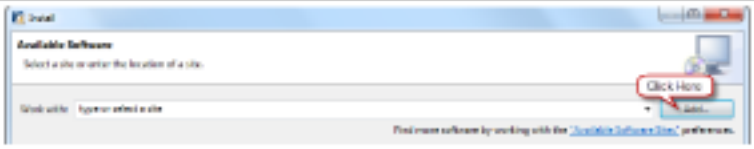
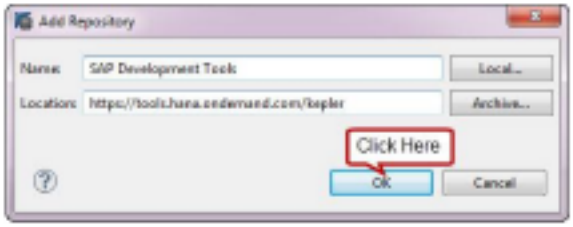
Appendix

Installation Guides

Install Eclipse

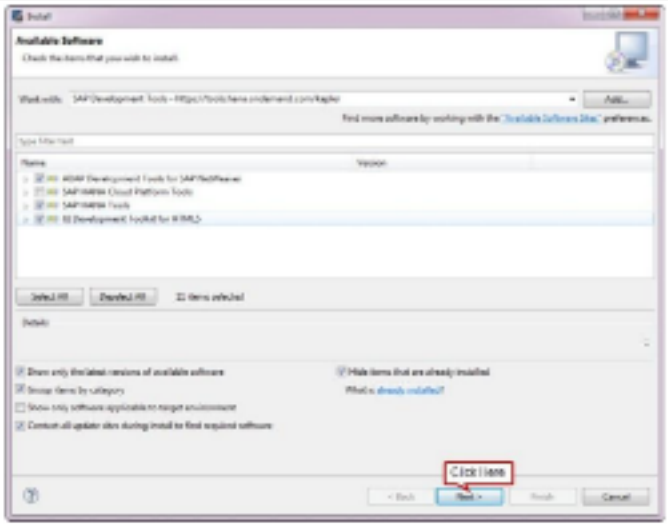
In the following I assume that you are working on Windows. In this case, download the 32-bit Version of Eclipse Kepler from (www.eclipse.org), extract the zip file to a folder of your choice, and run the eclipse.exe application.

Install needed Eclipse Plug-Ins

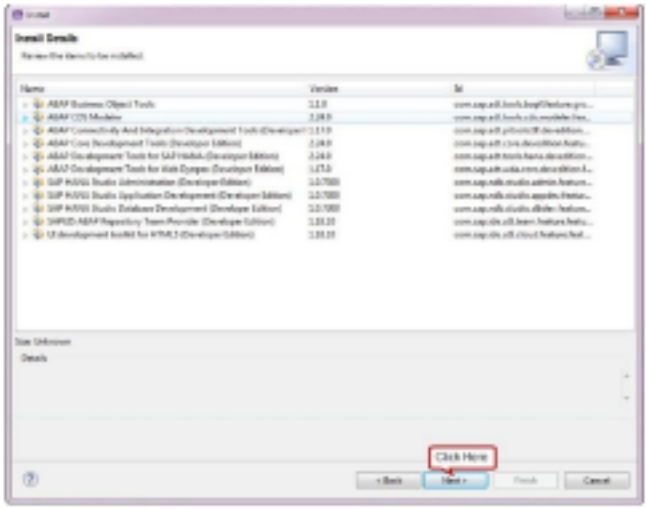
Description	Screen Shot
1. Open the *Help* menu and choose *Install new Software…*	
2. Click on *Add...* to add the needed update Sites for the Plug-Ins	
3. Add the update site https://tools.hana.ondemand.com/kepler	

Name	Location
SAP Development Tools	https://tools.hana.ondemand.com/kepler

4. Install the plugins:
   - ABAP Development Tools for SAP NetWeaver
   - SAP HANA Tools
   - UI Development Toolkit for HTML5

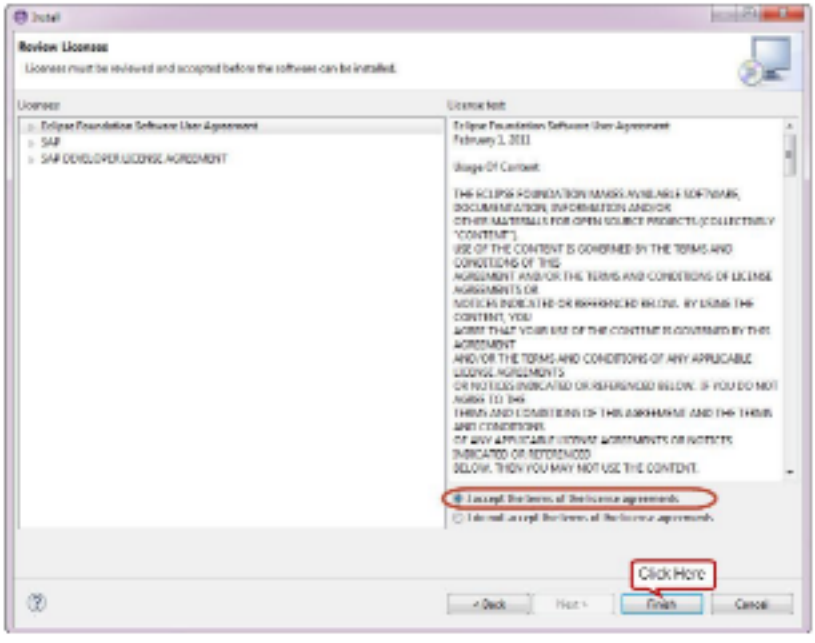


5. Check the installation details

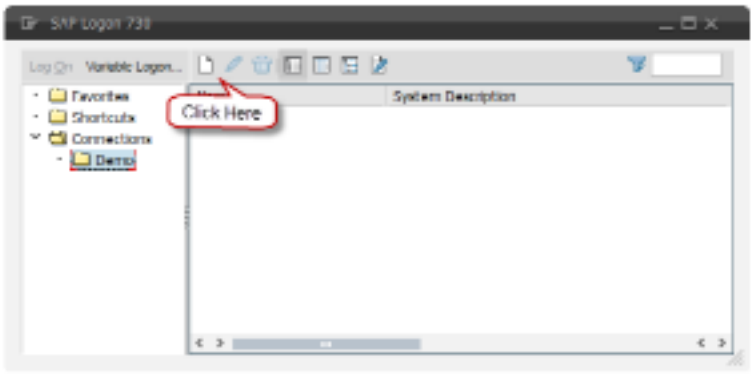
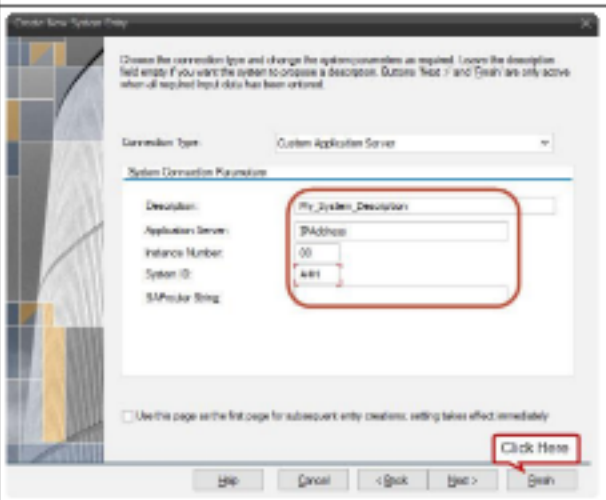
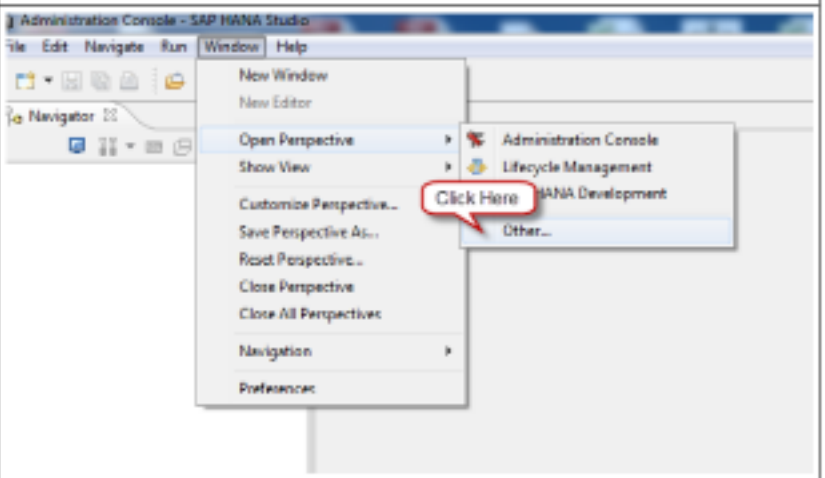


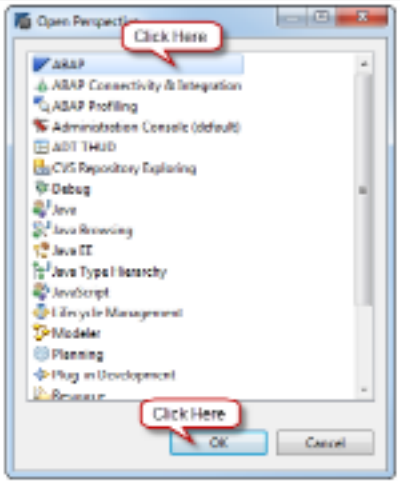
6. Accept the terms of the license agreements finalize the installation.

   Note: During the installation process you'll be asked to confirm security warnings and to restart the Eclipse. Please do the needy here ☺.

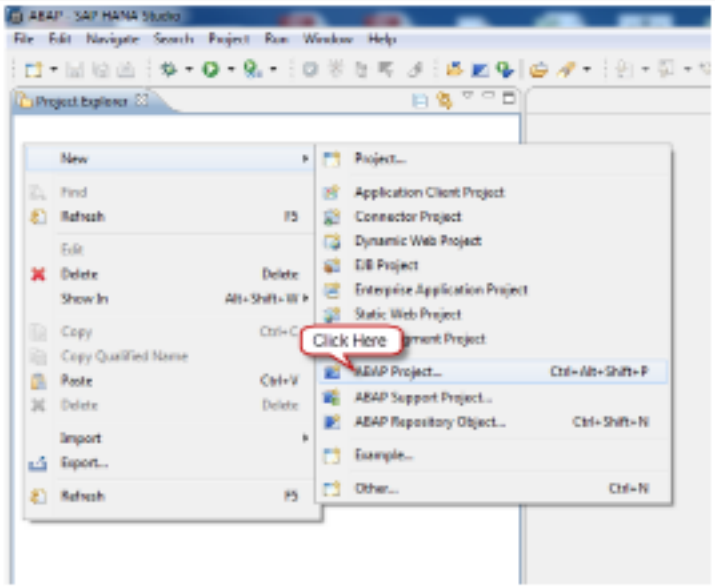


Add an ABAP Backend Connection (SAP Logon and ADT)

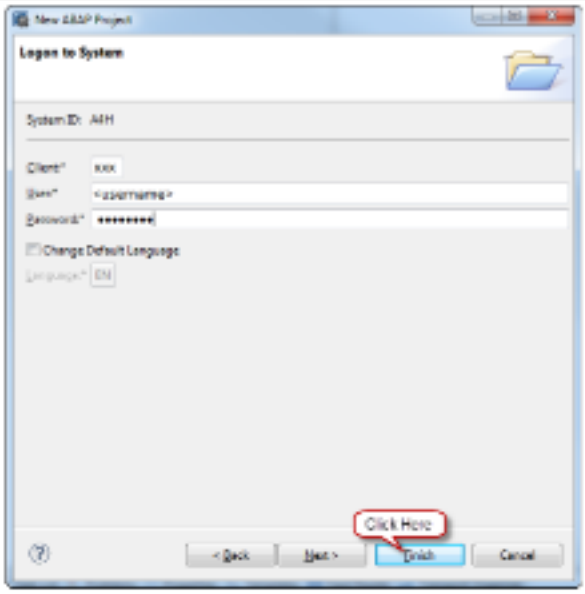
Description	Screen Shot
1. To add an ABAP System to the ABAP Development Tools you need to create the entry for your system in the SAP Logon first. So, please open the SAP Logon application and click on the New System icon.	
2. Add the system details, i.e. Application Server (IP Address or System alias if maintained in the `C:\Windows\System32\drivers\etc\hosts` file (as is the case for virtual appliances).	
3. Go to the ABAP perspective of your Eclipse installation.	



4. Right-click in the Project explorer view and select *New > ABAP Project* from the context menu

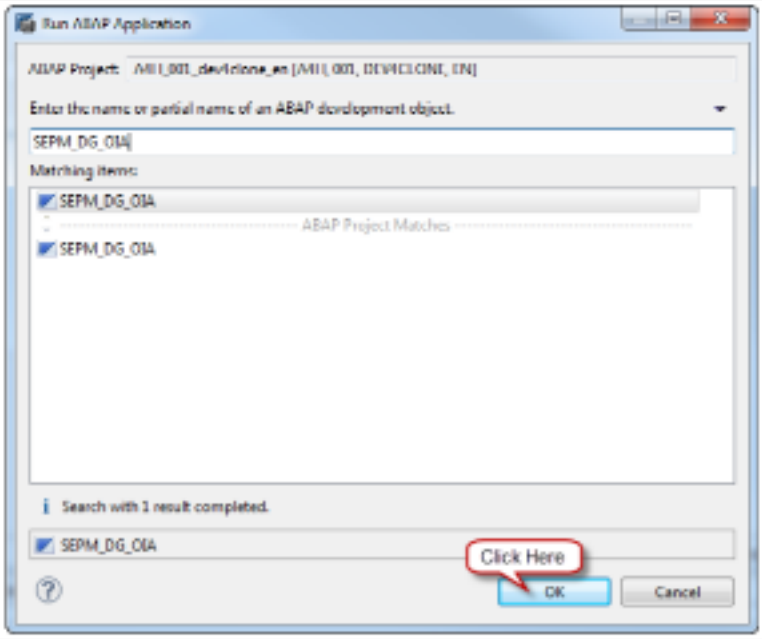
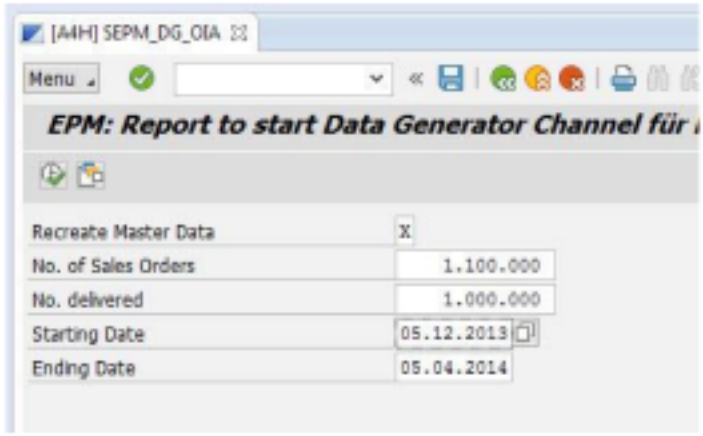


5. In the dialog window, enter the necessary system data:
   a. Client
   b. User
   c. Password



System Configuration & Example Data Generation

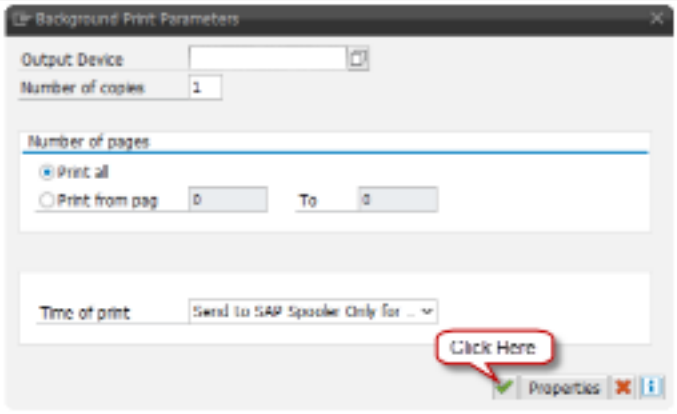
Generate Example Data

Description	Screen Shot
1. Press *Run ABAP Development Object* (short cut `Alt+F8`)	
2. Select transaction `SEPM_DG_OIA`	
3. Configure the data generator according to the screenshot and please read the explanation below.	

"Explanation below"
The transaction will generate the all data necessary to run the described scenario and show you the benefit of the SAP HANA. We recommend you to run the data generation as a background job because it takes a while to generate the data.
The values as shown on the screenshot will generate 1.000.000 Sales Orders with approximately 6.000.000 Sales Order Items, the according Invoices and the corresponding master data.

Press `F9` to execute the program as a background job or `F8` or the execute button to run the program directly (not recommend).

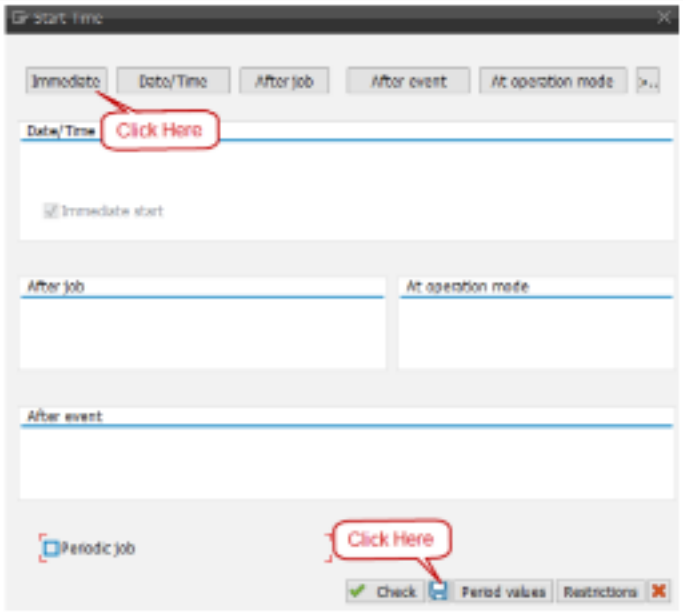
4. If you have chosen wisely to run it as a background job, just continue with the next dialog window.



5. Press the *Immediate* button and press the *save* button afterwards.

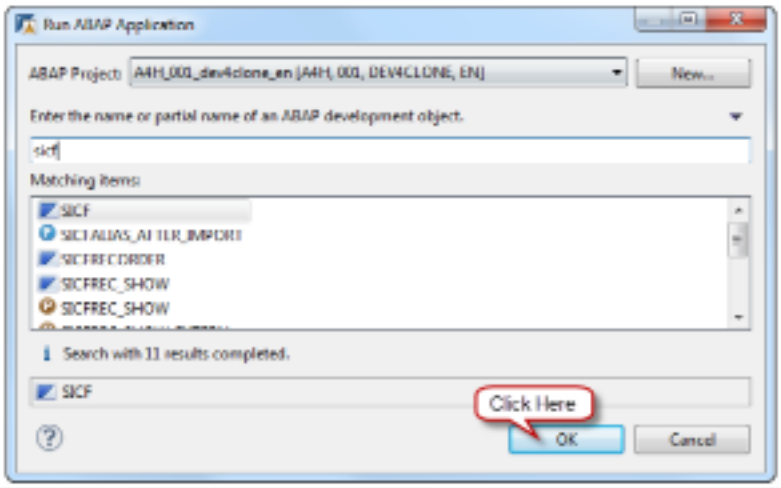
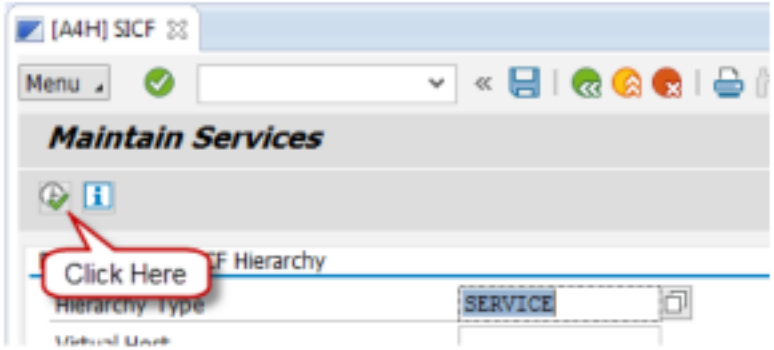
   This is now the time to take a break and fetch a coffee or continue with the configuration.

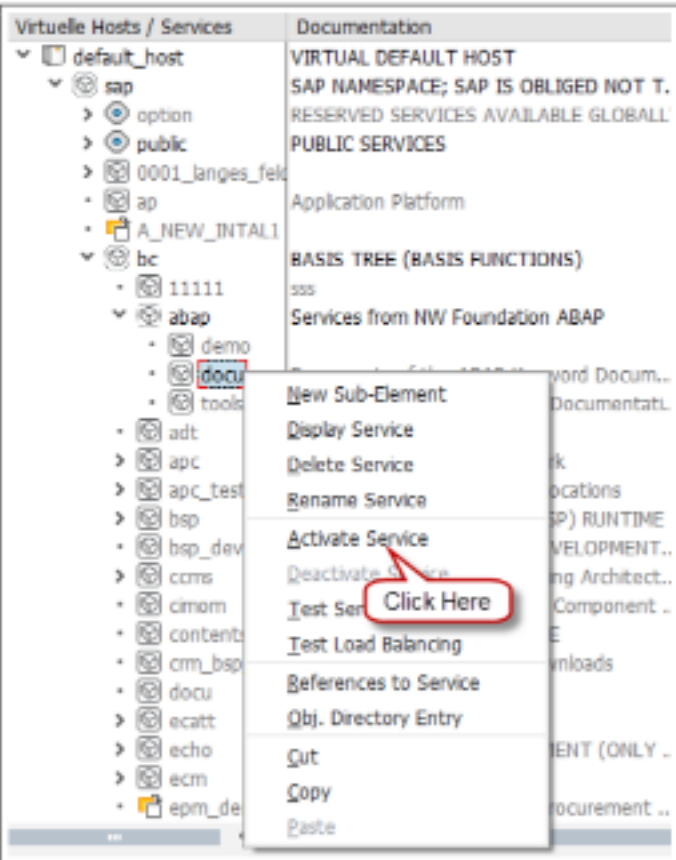
   To check whether the data generator finished, please have a look at transaction `SM37` and check on job `RS_EPM_DGC_HANA`.



ICF Configuration

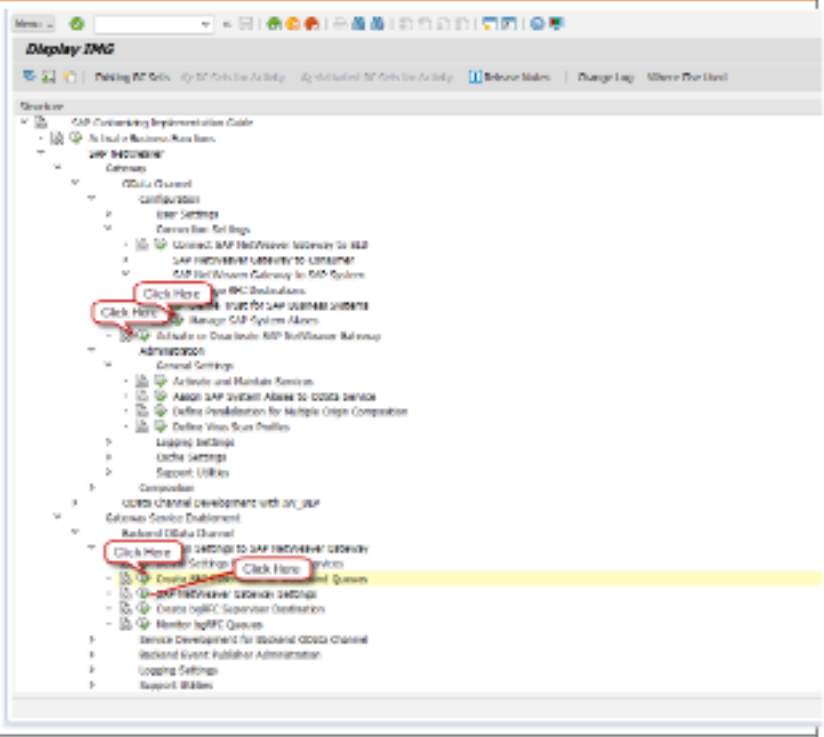
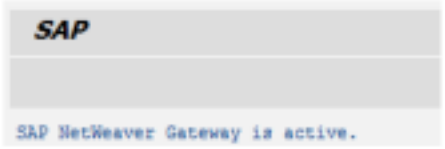
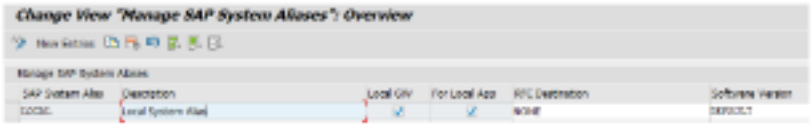
For SAP Gateway services, SAP UI5 and ABAP Docu you will need to activate special nodes on the ABAP ICF Server.

Description	Screen Shot
1. Run transaction `SICF` (short cut `Alt+F8` and select `SICF`)	
2. In the opened search screen just press execute	

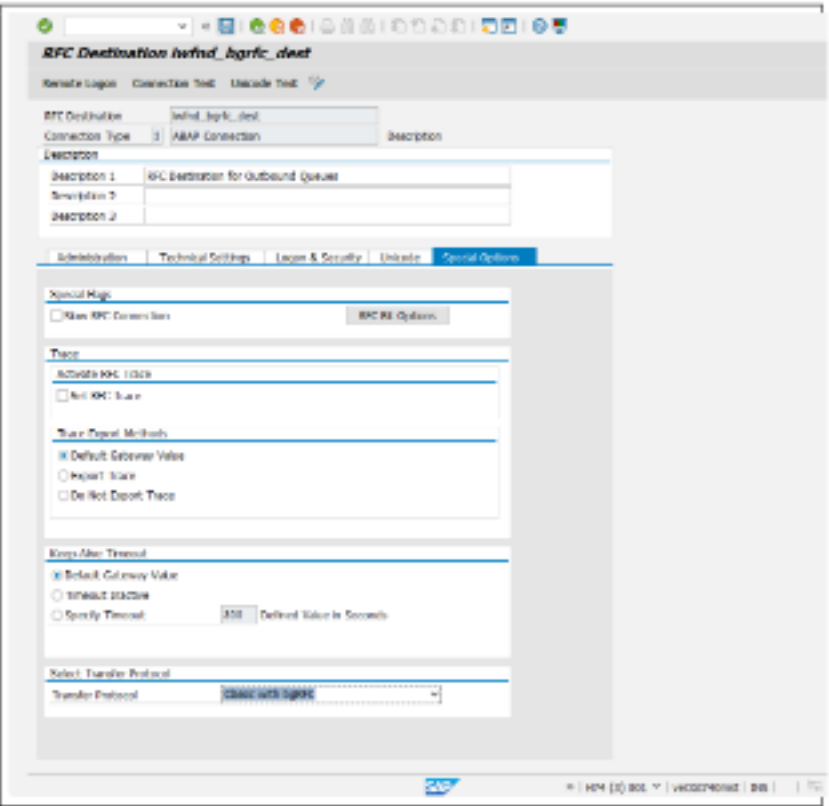
3. Activate the Service for ABAP Docu (`sap > bc > docu`)	
4. Repeat it for the UI5 Node (`sap > bc > ui5_ui5` and press *YES* (with sub nodes)	
5. Finally activate the Gateway node (*default_host > sap > opu*) also with sub nodes	

Customizing for UI5 & Gateway Services

Remark: You will be requested for a Customizing transport during the configuration steps.

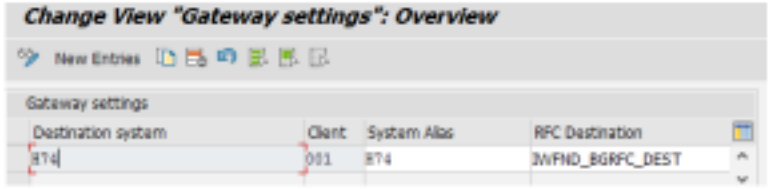
Description	Screen Shot
1. Run transaction SPRO (short cut Alt+F8 and select SPRO). 2. Click on the Button *SAP Reference IMG* and open the folders as shows on the screen shot	
3. Click on Activate or Deactivate SAP NetWeaver Gateway. In the dialog window press *Activate*, and click *back* in the menu or press F3.	
4. Click on *Manage SAP System Aliases*. In the new screen, press *New Entries* and insert the values as shown on the screenshot.	

5. Click *back* in the menu or press `F3`. Next step is to *Create RFC Destination for Outbound Queues*. In the configuration screen click the create icon and insert the values as shown on the screen shot.



6. Go back two times (press `F3` twice). Now you're ready to configure the *SAP NetWeaver Gateway Settings*. Click *New Entries* and enter the values.

   Note: H74 is the `SID` of the ABAP system resp. a system alias and might differ in your case. Same holds for the client, i.e. 001 in the screen shot.



Appendix: ADT Shortcuts

Edit

Ctrl+Shift+A Open development object
Ctrl+F2 Check development object
Ctrl+F3 Activate development object
Ctrl+Shift+F3 Activate all inactive objects
Ctrl+Space Code completion
Ctrl+1 Quick fix proposal
Ctrl+< Add comment
Ctrl+Shift+< Remove comment
Shift+F1 Format source aka pretty printer

Help

F1 ABAP keyword documentation
F2 Show code element information
Ctrl+3 Search for commands & views
Ctrl+Shift+L List all keyboard shortcuts

Navigate

F3 Open definition
Alt+Left Backward history
Alt+Right Forward history
Ctrl+T Quick hierarchy
F4 Open Type Hierarchy
Ctrl+O Quick outline
Ctrl+Shift+G Where-used list

Run, Debug

F8 Run current ABAP object
Alt+F8 Select & run ABAP application
Ctrl+Shift+B Toggle breakpoint
F5, F6, F7, F8 Step into, over, return, resume

Appendix: SAP HANA Development Guide

http://help.sap.com/hana/hana_dev_en.pdf

Appendix: SAP HANA SQL Script Reference

http://help.sap.com/hana/hana_dev_sqlscript_en.pdf

www.sap.com