

Name: abap-dev-adt-create-class-structure

title: ABAP Development: Create a Global ABAP class and Data Dictionary Structure

description: You will learn how to create data retrieval logic based on a SELECT statement.

tags: [ tutorial>beginner, topic>abap-development ]

## Prerequisites

[Use the Data Preview and SQL Console] (link)

[Create and run an ABAP program] (link)

## Next Steps

[Create a CDS view ](link).

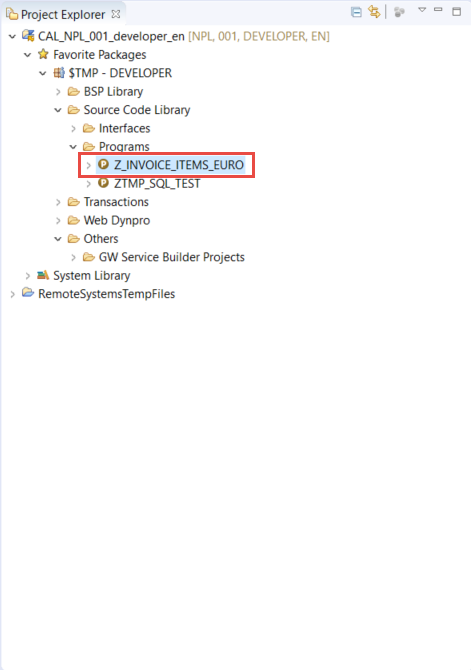
## Details

In the following exercise you will learn how to create data retrieval logic, by replacing the WRITE statement in your report with the SELECT statement you created previously in the SQL Console. In order to separate concerns properly and to show you a wider toolset the data retrieval logic will be encapsulated in a global ABAP class

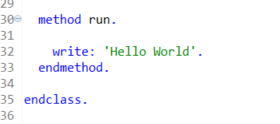
### Time to Complete

\*\*1 hour\*\*.

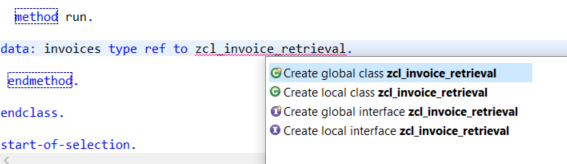
1. First, open your ABAP program, which you created in \_\_\_:



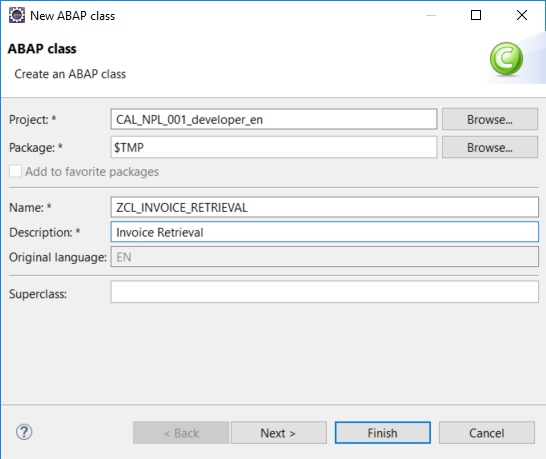
1. Remove the WRITE statement: Delete the whole line, by placing the cursor somewhere in the WRITE statement and choosing **Ctrl+D**.



1. Now create the global class for the data retrieval:
2. In the **run** method, create a local variable of the type **type ref to** **zcl\_invoice\_retrieval**:
3. data: invoices type ref to zcl\_invoice\_retrieval.
4. Since this class does not yet exist, you will get a syntax error. To create the class, place the cursor on the class name and press **Ctrl+1** to get a Quick Assist. In the Quick Fix menu double-click on **Create global class zcl\_invoice\_retrieval**:

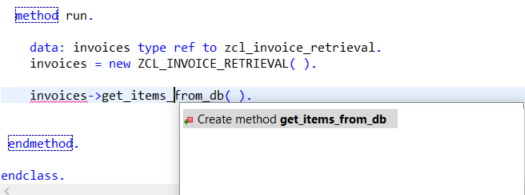


1. A wizard will appear to create a new ABAP class. Add a meaningful description and choose **Finish**.

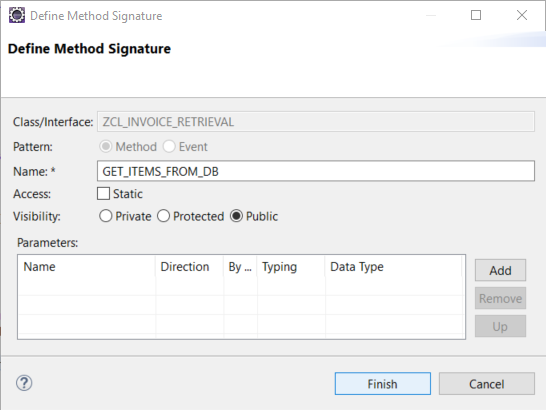


> A new editor will be opened showing the class you have created.

1. Go back to your report and trigger the syntax check using the keyboard shortcut Ctrl+F2. The syntax error should no longer occur.
2. Create an instance of class zcl\_invoice\_retrieval using the new operator.
3. To read the records from the database, you need to call the method **get\_items\_from\_db**.  
   This method does not yet exist; we will create it with a Quick Assist.
4. Position the cursor on the name of the missing method and press Ctrl+1. In the Quick Fix menu double-click on Create method **get\_items\_from\_db** :



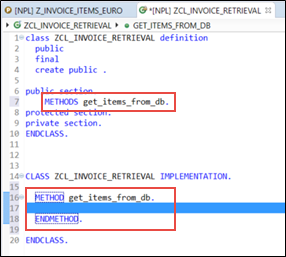
b. In the wizard that appears, create a **public** method without parameters, by simply choosing **Finish**:



> The Quick Fix creates:

- a method definition

- an empty method implementation in the class **ZCL\_INVOICE\_RETRIEVAL** :

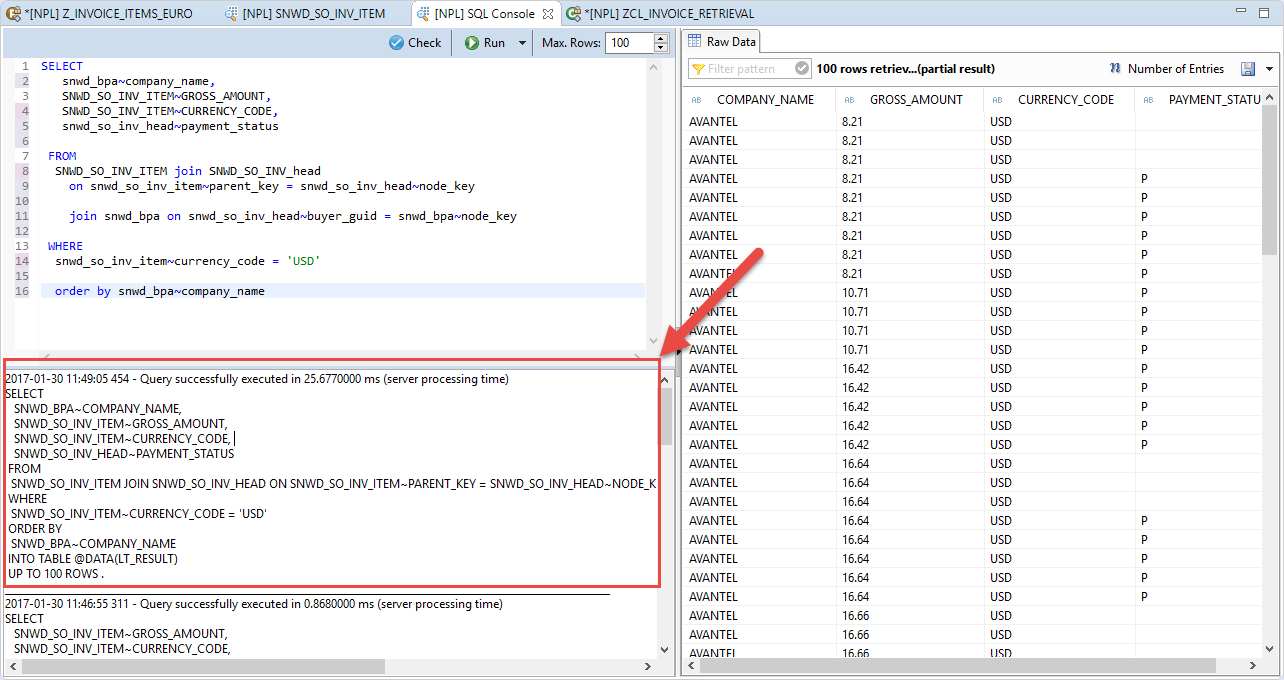


1. To improve readability, add an ABAP Doc comment to the method definition, eg:

“! Method reading invoice items from database

> ABAP Doc comments can be used to document APIs and are displayed in the Element Info. ABAP Doc comments begin with **"!** .

1. Now you will complete the method implementation of **get\_items\_from\_db**, using the SQL Console:
2. Open the **SQL Console**.
3. On the bottom left hand side of the SQL console is the query section. It contains the last executed SELECT statements as Open SQL statements ready to be transferred into your ABAP code:



Your code should look like this. Note that the SQL Console query section automatically adds the INTO clause INTO TABLE @DATA(LT\_RESULT):

SELECT

SNWD\_BPA~COMPANY\_NAME,

SNWD\_SO\_INV\_ITEM~GROSS\_AMOUNT,

SNWD\_SO\_INV\_ITEM~CURRENCY\_CODE,

SNWD\_SO\_INV\_HEAD~PAYMENT\_STATUS

FROM

SNWD\_SO\_INV\_ITEM JOIN SNWD\_SO\_INV\_HEAD ON SNWD\_SO\_INV\_ITEM~PARENT\_KEY = SNWD\_SO\_INV\_HEAD~NODE\_KEY JOIN SNWD\_BPA ON SNWD\_SO\_INV\_HEAD~BUYER\_GUID = SNWD\_BPA~NODE\_KEY

WHERE

SNWD\_SO\_INV\_ITEM~CURRENCY\_CODE = 'USD'

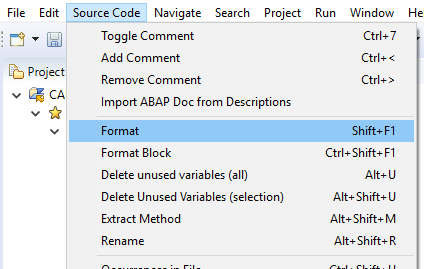
ORDER BY

SNWD\_BPA~COMPANY\_NAME

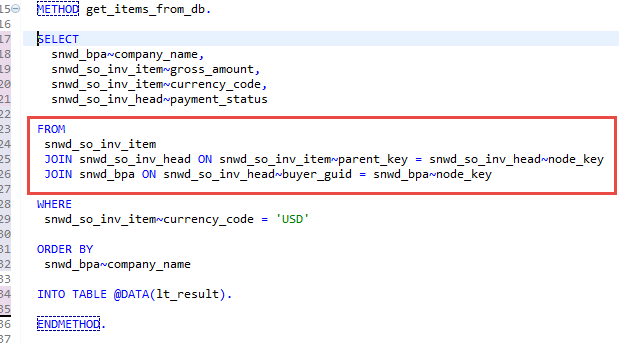
INTO TABLE @DATA(LT\_RESULT).

UP TO 100 ROWS.

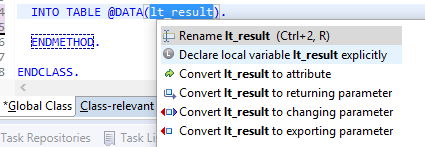
1. Resize the query section and copy the Open SQL statement using the shortcut Ctrl+C.
2. Now paste the statement into the method implementation of **get\_items\_from\_db**.
3. Now you can format (ie “pretty-print”) the source code: Open the Source menu and Click **Format**. (Alternatively you can use the shortcut **Shift+F1**.).  
   > If you want to specify your formatting settings, you can do this in the project's properties. Right-click on the Project in the Project Explorer and choose **Properties**.



1. Delete the addition UP TO 100 ROWS.
2. To make the SELECT statement more readable, add some line breaks in the JOIN conditions:



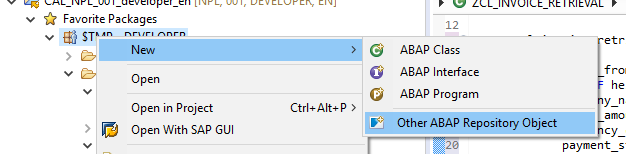
1. Now you will define your own Data Dictionary (“DDIC”) structure as a result type of the method: First, you will declare a local variable explicitly by using a quick fix. Afterwards we will create the Dictionary structure:
2. Declare the local variable using a Quick Assist, by positioning the cursor on the declared inline variable `LT\_RESULT` and choosing `Ctrl+1`:



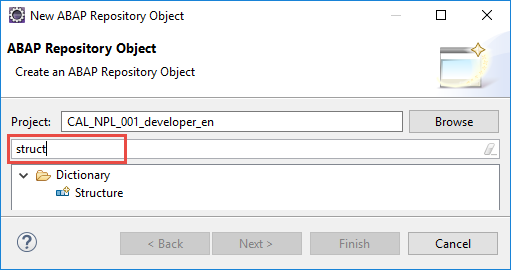
b. Select **Declare local variable `lt\_result` explicitly** by double clicking in the Quick Assist menu.  
 > Adds DATA: lt\_result TYPE STANDARD TABLE OF helper\_type.

In the next steps you will replace the local type with a global Data Dictionary structure.

1. First, create the global Data Dictionary structure:
2. In the Project Explorer, select the folder $TMP-DEVELOPER [or $TMP-<USERNAME>]:

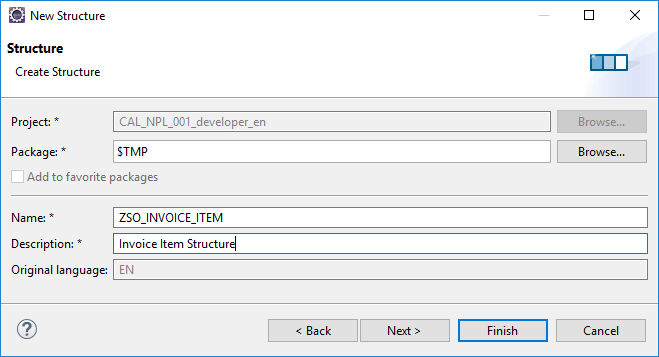


1. In the wizard that appears, filter the list of ABAP repository object types by entering **struct**.



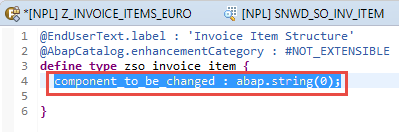
1. Then enter the following and choose **Finish.**

* Name = ZSO\_INVOICE\_ITEM
* Description = eg Invoice item structure



> A new text editor is opened showing the content of the newly created Data Dictionary structure.

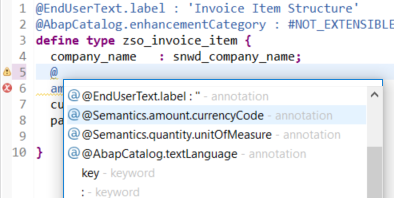
1. Remove the generated example component**component\_to\_be\_changed** from the structure:



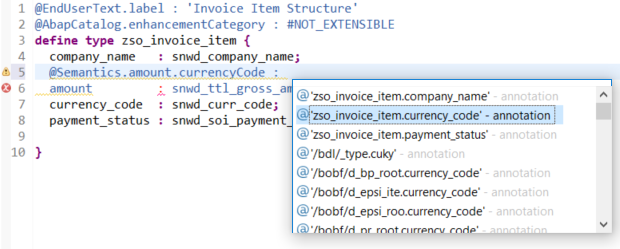
1. In the item structure, define the fields **company\_name**, **amount**, **currency\_code**, and **payment\_status.**

> The editor shows a syntax error because the amount has not yet been bound to the currency code.

* 1. To bind the amount to the currency code:
* Add a new line in front of the field **amount**, add **@** and open code completion, by entering **Ctrl+1**.  
  > A list of all possible annotations is shown.
* Select the annotation @Semantics.amount.currencyCode:



* Trigger the code completion again by entering **: ‘** and then entering **Ctrl+1**:



1. Choose the annotation zso\_invoice\_item.currency\_code
2. Finally choose Save (Ctrl+S) and Activate (Ctrl+F3).
   * 1. We added the link between the amount field and the currency code. The editor should not show any syntax errors any more.

@EndUserText.label : 'Invoice Item Structure'

@AbapCatalog.enhancementCategory : #NOT\_EXTENSIBLE

**define** **type** zso\_invoice\_item **{**

company\_name **:** snwd\_company\_name**;**

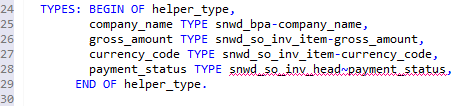
@Semantics.amount.currencyCode : 'zso\_invoice\_item.currency\_code'

amount **:** snwd\_ttl\_gross\_amount**;**

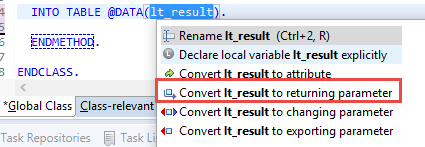
currency\_code **:** snwd\_curr\_code**;**

payment\_status **:** snwd\_soi\_payment\_status\_code**;**

**}**

1. Now, switch back to the editor of your invoice retrieval class **ZCL\_INVOICE\_RETRIEVAL**.   
   > In the next steps, you will replace the local type **helper\_type** with the Data Dictionary structure that you created.
   1. In the method **get\_items\_from\_db**, change the type of the variable **lt\_result** to a standard table of **zso\_invoice\_item**:  
      
   2. Remove the local type **helper\_type**:  
      

Your method still does not return any data. Therefore, you will use another Quick Assist to declare your local variable as a returning parameter. You want to convert the local variable to a returning parameter of your method so that you can access the result from your program.

* 1. To do so place the cursor on the variable **lt\_result** and enter **Ctrl+1**.
  2. Choose Convert lt\_result to returning parameter:  
     

> Note that the returning parameter was added to the method and an additional table type based on the structure was generated.

1. You can also use ABAP Doc to document method parameters with a Quick Assist. Place the cursor inside of the ABAP Doc comment. Then enter **Ctrl+1** to open the Quick Assist menu and double-click on **Add missing parameters to documentation**.  
   > The ABAP Doc comment is extended by a **@parameter ... |** . You can now use this to document the method parameters: To do so, just enter the documentation after the pipe symbol ( **|** ).
2. We have documented our method using ABAP Doc. However, we also want to see the same short texts in the description fields of the form-based Class Builder in SAP GUI.  
   To do this, we need to tag the required text in the ABAP Doc as "synchronized" to ensure that it is synchronized with the Class Builder.
   1. Mark the shorttext for your method in ABAP Doc as "synchronized" by surrounding it with the tag <p class="shorttext synchronized">...</p>.
   2. Do the same for the shorttext of your parameter **lt\_result**.
3. **Activate** the class by clicking the activation iconfieldicon in the toolbar.  
     
   The code for your class should look like this:

CLASS zcl\_invoice\_retrieval DEFINITION

PUBLIC

FINAL

CREATE PUBLIC .

PUBLIC SECTION.

TYPES: ty\_table\_of\_zso\_invoice\_item TYPE STANDARD TABLE OF zso\_invoice\_item WITH DEFAULT KEY.

"! method reading invoice items from db

METHODS get\_items\_from\_db

RETURNING

VALUE(lt\_result) type ty\_table\_of\_zso\_invoice\_item.

PROTECTED SECTION.

PRIVATE SECTION.

ENDCLASS.

CLASS zcl\_invoice\_retrieval IMPLEMENTATION.

METHOD get\_items\_from\_db.

SELECT

snwd\_bpa~company\_name,

snwd\_so\_inv\_item~gross\_amount,

snwd\_so\_inv\_item~currency\_code,

snwd\_so\_inv\_head~payment\_status

FROM

snwd\_so\_inv\_item

JOIN snwd\_so\_inv\_head ON snwd\_so\_inv\_item~parent\_key = snwd\_so\_inv\_head~node\_key

JOIN snwd\_bpa ON snwd\_so\_inv\_head~buyer\_guid = snwd\_bpa~node\_key

INTO TABLE @lt\_result

WHERE

snwd\_so\_inv\_item~currency\_code = 'USD'

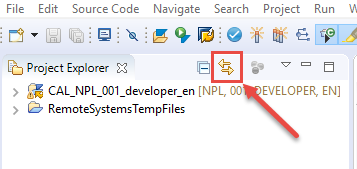
ORDER BY

snwd\_bpa~company\_name.

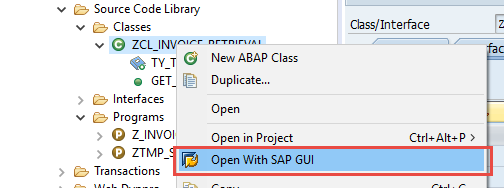
ENDMETHOD.

ENDCLASS.

1. Finally in this section, you will check that the synchronized short texts are also shown in the Class Builder. First we have to open the class in SAP GUI.   
   To easily find the class in the Project Explorer, choose **Link with Editor**

:

1. In the Project Explorer, select the class **ZCL\_INVOICE\_RETRIEVAL** and choose **Open with SAP GUI** from the context menu:

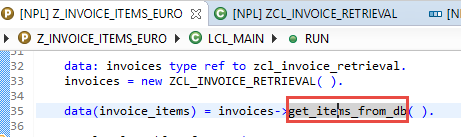


1. The method description shows the text we entered in the ABAP Doc comment in the synchronized tag.  
     
   Choose **Parameters.** You will see that the description of the parameter has also been synchronized.
2. Close the class in SAP GUI and return to your ABAP program by choosing **Close**:

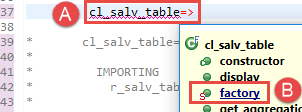


1. Back in the ABAP Program, position the cursor on the method call **GET\_ITEMS\_FROM\_DB** and display the Element Info of the method by choosing Element Info (**F2)**.  
   In addition to the method signature it also shows the ABAP Doc you wrote before.  
   Close the Element Info by choosing **ESC**.
2. Now you use a new inline declared variable, **data(invoice\_items)** , to receive the result of the returning parameter **invoices->get\_items\_from\_db( )** :

**data(invoice\_items) = invoices->get\_items\_from\_db( ).**



1. Now you can display the invoice items in an ALV Grid, using the class CL\_ALV\_TABLE :
2. Enter **cl\_salv\_table=>** and get the code completion proposals by entering **Ctrl+SPACE** :
3. Select the static method **factory** and …
4. Insert the full signature of the method call by pressing **Shift+Enter :**

  
  
> If you prefer to insert the full signature by default, you can change the behavior of the code completion in the Preferences. Select Window in the menu and click on Preferences. In the Preferences Dialog enter code completion in the filter field or open the following path ABAP Development > Editors > Source Code Editors > Code Completion. In the Code Completion settings, you can activate a checkbox to Always insert full signature on completion.

1. In the generated method call:
2. Remove the commented exporting parameters **list\_display**, **r\_container**, and **container\_name** using the shortcut **Ctrl+D**
3. Uncomment the importing parameter **r\_salv\_table** using the shortcut **Ctrl+7** and assign it to an inline variable **alv\_table**
4. Assign the variable **invoice\_items** to the changing parameter **t\_table**
5. Then call the display method of ALV\_TABLE :  
    alv\_table->display( ).
6. Activate your report by clicking the activation icon fieldicon in the toolbar or using the keyboard shortcut **Ctrl+F3**.

Your method code should look like this:  
 method run.

data: invoices type ref to zcl\_invoice\_retrieval.

invoices = new ZCL\_INVOICE\_RETRIEVAL( ).

data(invoice\_items) = invoices->get\_items\_from\_db( ).

cl\_salv\_table=>factory(

IMPORTING

r\_salv\_table = data(alv\_table)

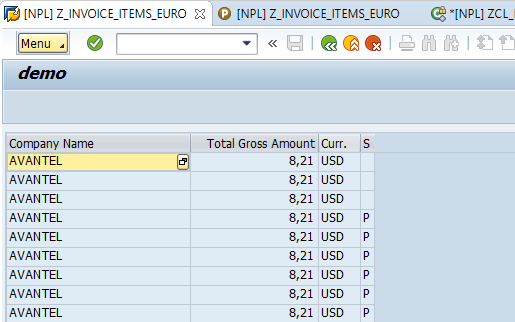
CHANGING

t\_table = invoice\_items ).

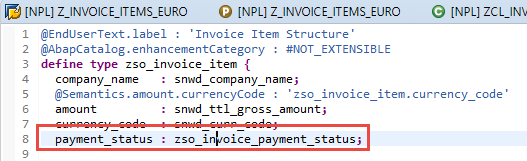
alv\_table->display( ).

endmethod.

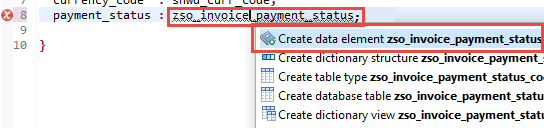
1. Run your report by choosing Execute (**F8)**.  
   The invoice items are displayed in the SAP List Viewer (or “ALV grid”):



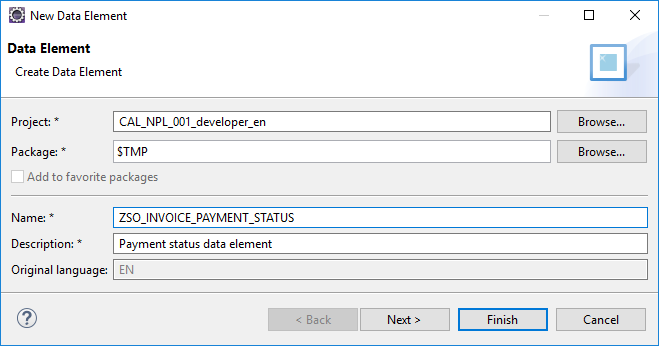
1. Notice that the payment status contains the internal codes such as P. You will now change this to a boolean field that just returns X (true) if the invoice is paid or SPACE (false) if it has not been paid. You will implement this logic in a new data element.
2. Go back to your Data Dictionary structure and change the type of field **payment\_status** to **zso\_invoice\_payment\_status :**



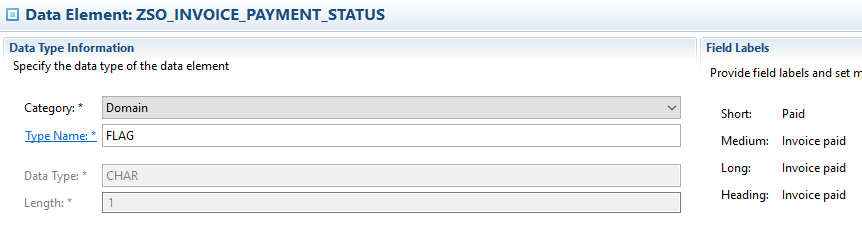
1. Since the data element **zso\_invoice\_payment\_status** does not exist, you get a syntax error. You will now resolve this using Quick Assist:
2. Select **zso\_invoice\_payment\_status** , open Quick Assist by choosing **Ctrl+1** , and create a new data element by choosing **Create data element zso\_invoice\_payment\_status** andchoosing **Enter** :



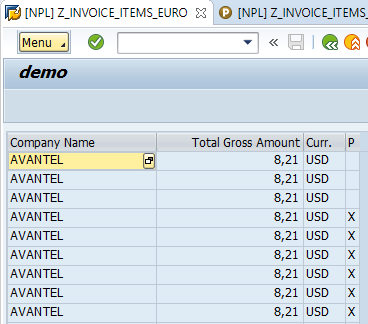
1. Enter a description for the new data element in the field Description and choose **Finish :**



1. Under “Field Labels” on the right, enter the following:
   * Short = **Paid**
   * Medium = Invoice paid
   * Long = Invoice paid
   * Title = Invoice paid



1. Choose Save (Ctrl+S) and Activate (Ctrl+F3) fieldicon.
2. Back in the structure **ZSO\_INVOICE\_ITEM**, choose **Check ABAP Development Object (Ctrl+F2)** fieldicon**.** Then choose **Activate (Ctrl+F3)** fieldicon.  
   > The Data Dictionary structure **ZSO\_INVOICE\_ITEM** is now activated.
3. Back in the retrieval class **Z\_INVOICE\_ITEMS\_EURO**, add a loop in the implementation part of the method **get\_items\_from\_db** which transforms the values of **payment\_status** into a flag that is **X** (true) when the invoice has been paid. Then activate the class by choosing **Activate (Ctrl+F3)** fieldicon.  
   > There is a preference which allows you to reuse SAP GUI for Windows when running applications. To enable this feature, select **Window** in the menu and choose on **Preferences**. In the Preferences dialog open **ABAP Development > SAP GUI Integration** and tick the check box **Reuse SAP GUI window when running applications**.
4. Go back to your report, , and execute it by choosing **F8**.  
   > In the SAP List Viewer, your ALV should look like this:



Summary

**You have completed the exercise!**

**You are now able to:**

* Create an ABAP Project which represents a logon to your development system
* Use the Data Preview to look at the content in database tables and views
* Use the SQL Console to create and fine tune more complex select statements
* Create ABAP Programs Classes, Dictionary structures and data elements
* Use Quick Assists via the shortcut for quick fixes **Ctrl+1**
* Document your source code using ABAP Doc