How to Extend an Outbound IDoc

Applies to:

Developing and configuring SAP Intermediate Documents (IDocs) for data transfer.

Related till version ECC 6.0.

For more information, visit the Idoc homepage and the ABAP homepage.

Summary

This document provides details on why we need an extended Idoc and how to create it. We will also see a step by step procedure of creating an extended Idoc. The Steps involve creating a segment of required structure, Extend basic Idoc and assign new segment to extended Idoc. Also involves Maintaining message type, partner profile and implementing the User exit. Further the Idoc is transferred to the destination. A whole overview of Extended Outbound Idoc can be learned with this article. Though some related information already available in SCN, to make it more comprehensive, I plan to explain it using screen shots along with a real world example making it easier.

Author: Lakshmi Narayana Neeli

Company: Intelligroup Asia Pvt Limited.

Created on: 14 Aug 2009

Author Bio

Lakshmi Narayana Neeli works with Intelligroup Asia Pvt Limited since 2008. He has good knowledge in SAP ABAP and CRM.

Table of Contents

Introduction

SAP's presence in the IT world is propelled by its unique integration of all its sub systems to a central system. More than a linking together of applications, implementing SAP redirects the flow of information through a company and its partners to enhances the potential of its business functions. This flow of information is enabled by a core element - Intermediate Document, or IDoc. Technically, the IDoc is an example of Electronic Data Interchange (EDI).The IDoc concept borrows the best features of EDI and combines them with the best features of conventional transaction file formats. SAP R/3 systems send out data through Idoc (Intermediate Document), which in internally has segments and fields containing the data.

Why We Need an Extended Idoc?

SAP send out data through Idoc with many predefined segments, Message types and fields. But sometimes, these fields are not sufficient for a specific end-to-end business scenario as far as data transfer is concerned. So in such scenario, we can add new segments with completely new structure to the standard Idoc as an extension called as extended Idoc. Here we create a new structure and insert it into existing Idoc structure creating a whole new Idoc satisfying the requirement. Following blog gives out step-by-step approach for creation of the same.

Business Scenario – Invoice

For instance let us take a scenario in billing, where we already have a predefined Idoc type 'INVOIC02'. But the requirement is to transfer additional structure containing VBRK-KTGRD (Account assignment group for this customer) and VBRK-MANSP (Dunning block).

To get the requirement fulfilled we wish to create a segment structure add segment with two additional fields as an extension to the existing Idoc Type 'INVOIC02'. The basic steps are explained here and going further a more detailed step by step illustration is done.

Basic Steps in Invoice Extension Outbound Idoc

- Create a segment with two VBRK-KTGRD (Account assignment group for this customer) and VBRK-MANSP (Dunning block)
- Extend basic Idoc with new segment
- Assign Extended Idoc to message type
- Maintain partner profile i.e., Update Extended Idoc in partner profile
- Find relevant user exit
- Implement User exit
- Update values into this your segment of extended Idoc
- Trigger output from VF02 with medium 6
- Check your Idoc in WE02

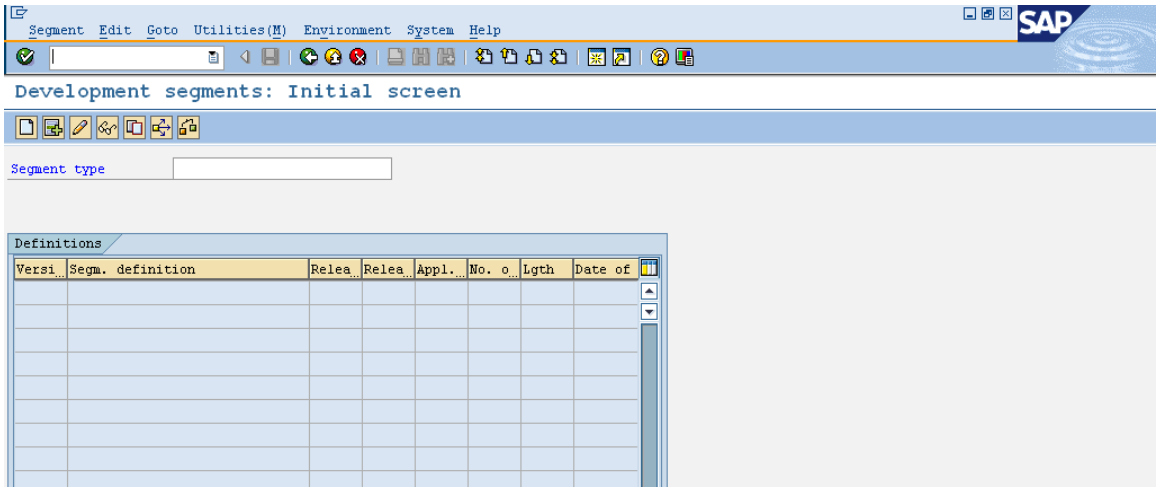
Now we will see a step by step creation of the extended Idoc.

Create a Segment WE31

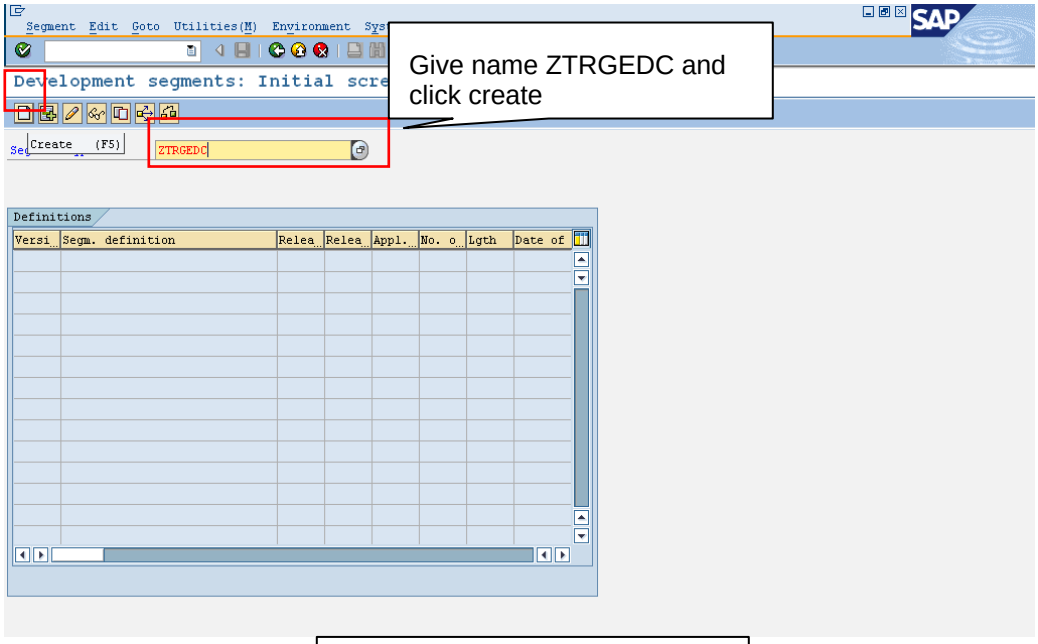
To create a segment with two VBRK-KTGRD (Account assignment group for this customer) and VBRK-MANSP (Dunning block)

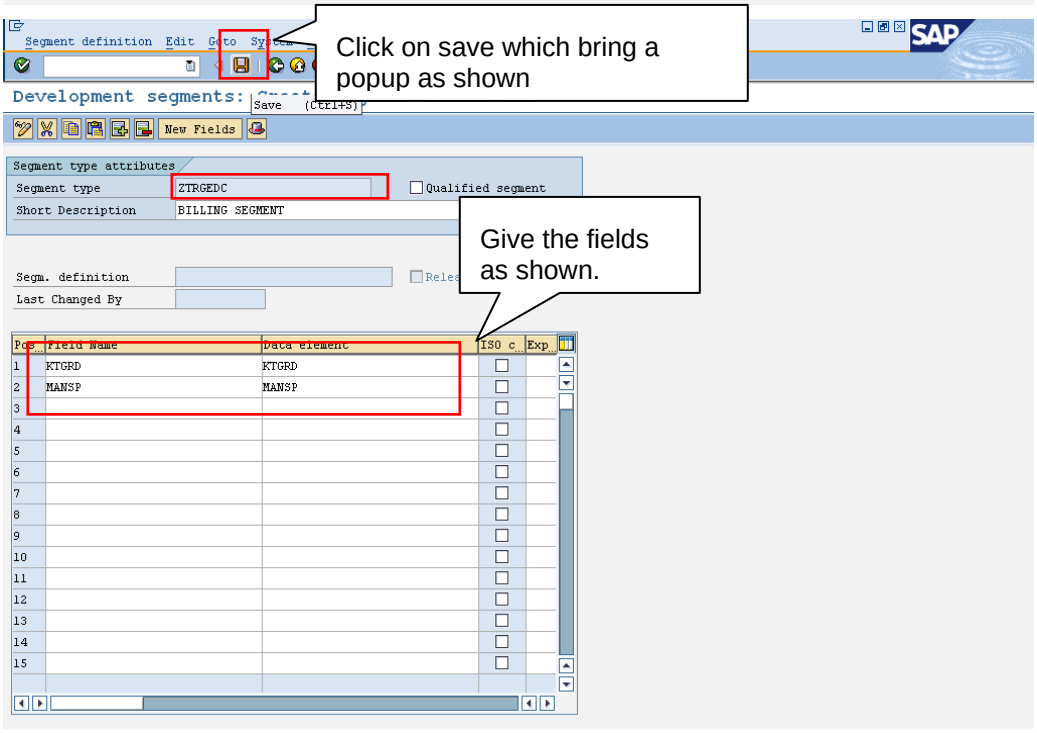
WE31- Create Segment Type

In this transaction we create a segment type. This segment type has two fields KTGRD and MANSP as specified from VBRK table. This segment will be used in extended Idoc as follows. In transaction WE31 we get the following screen.

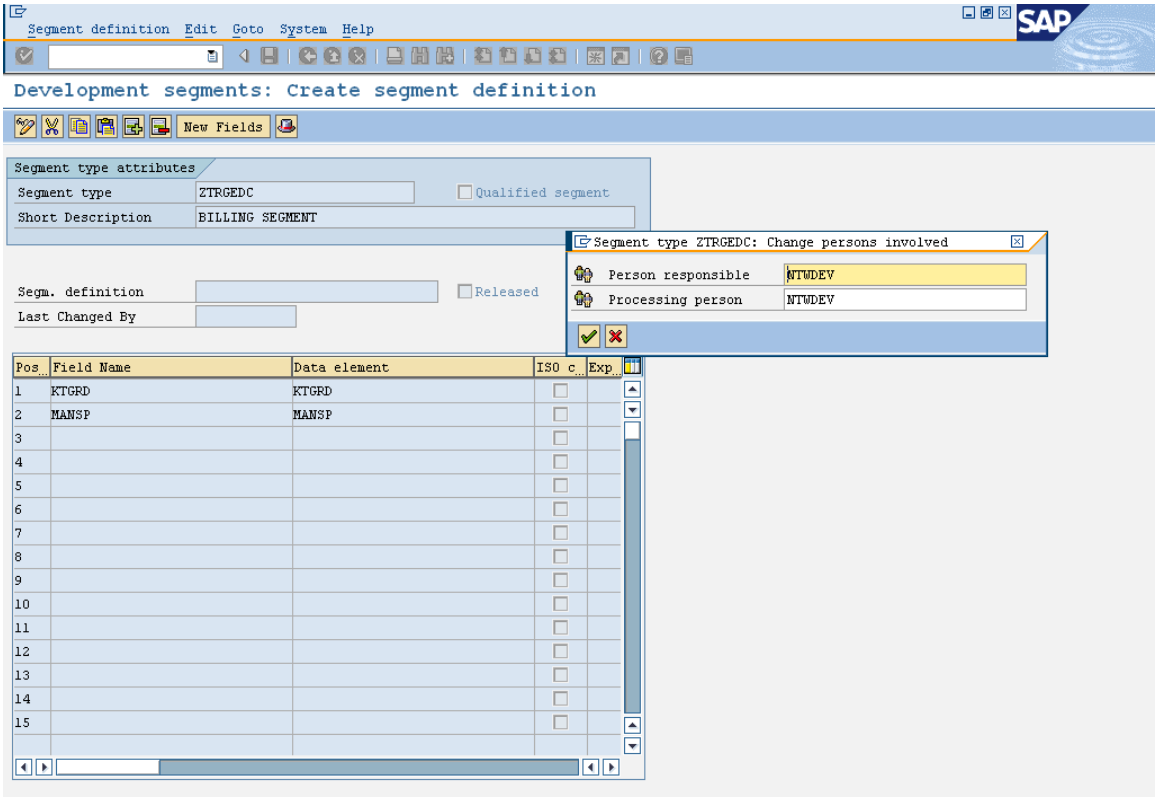


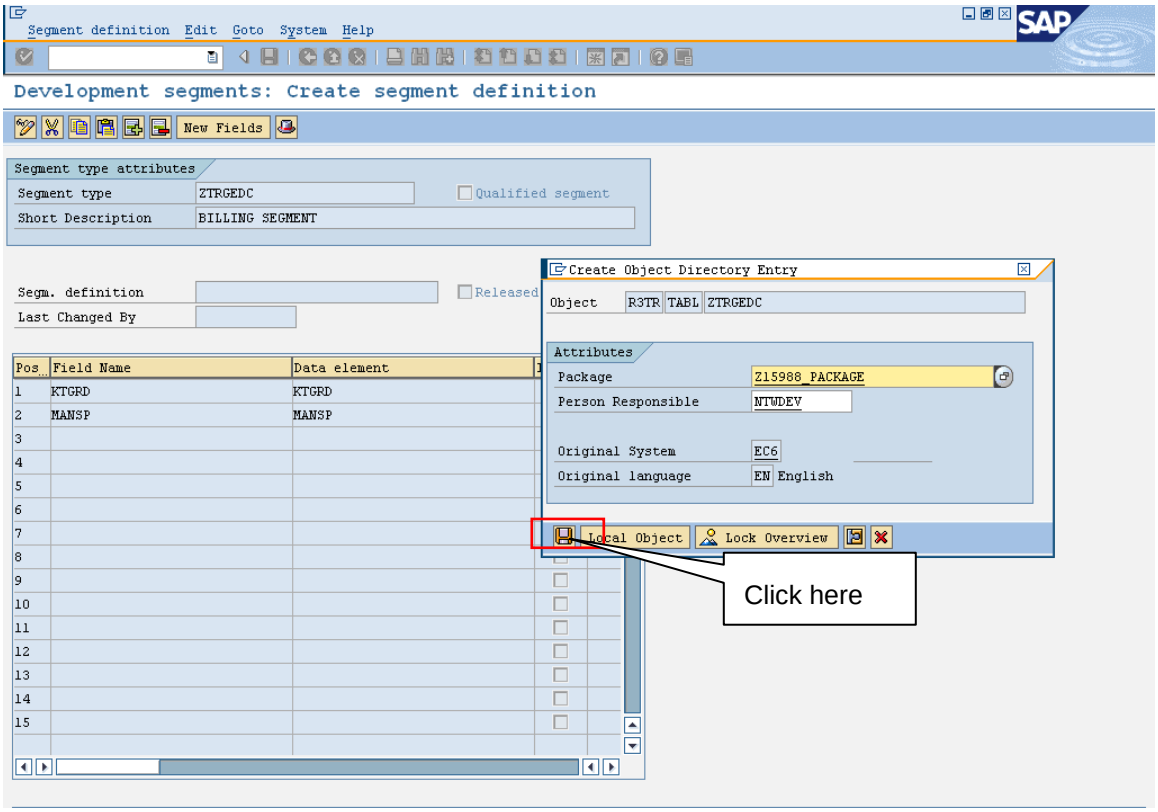
Create a segment “ZTRGEDC” as shown.

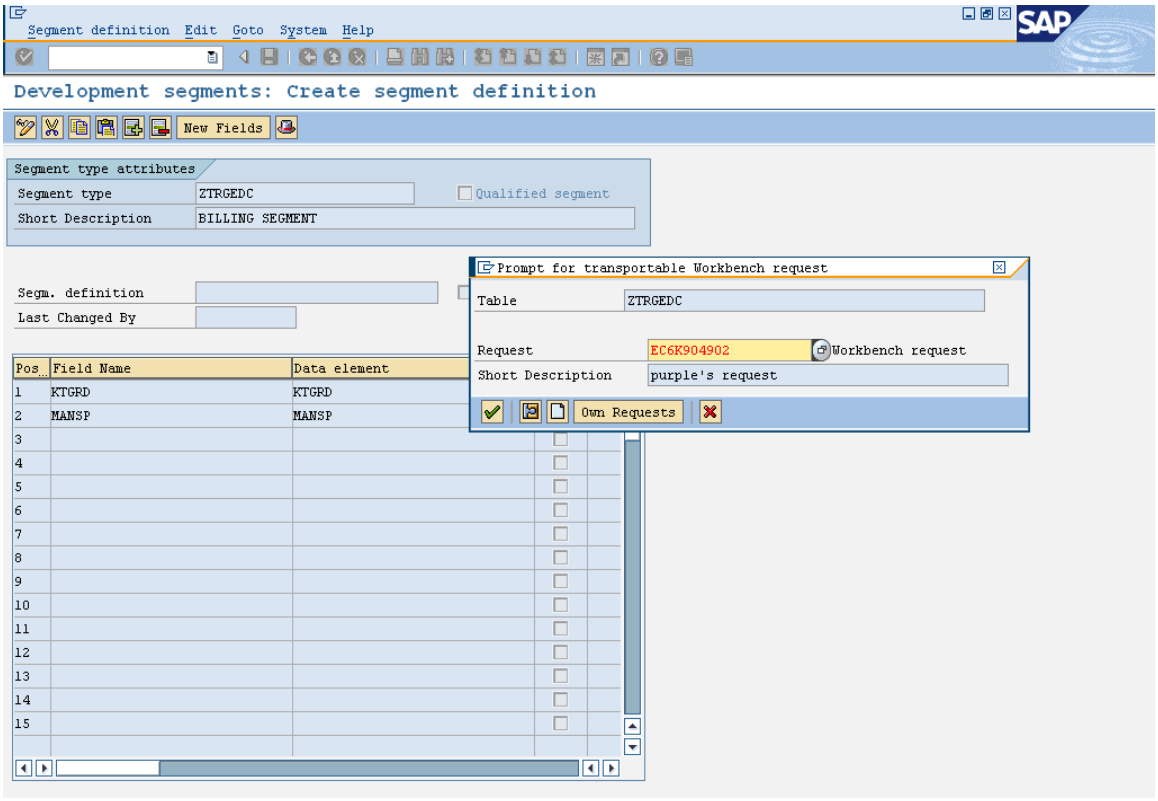


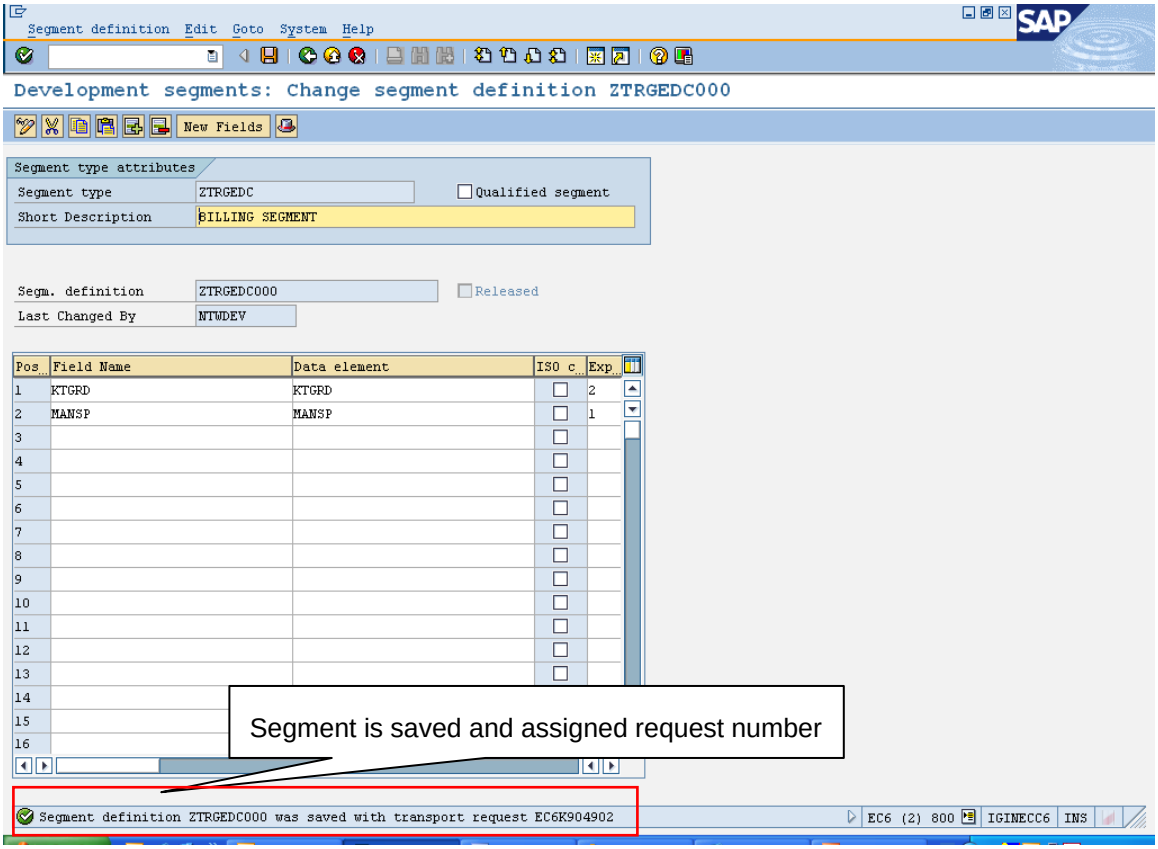


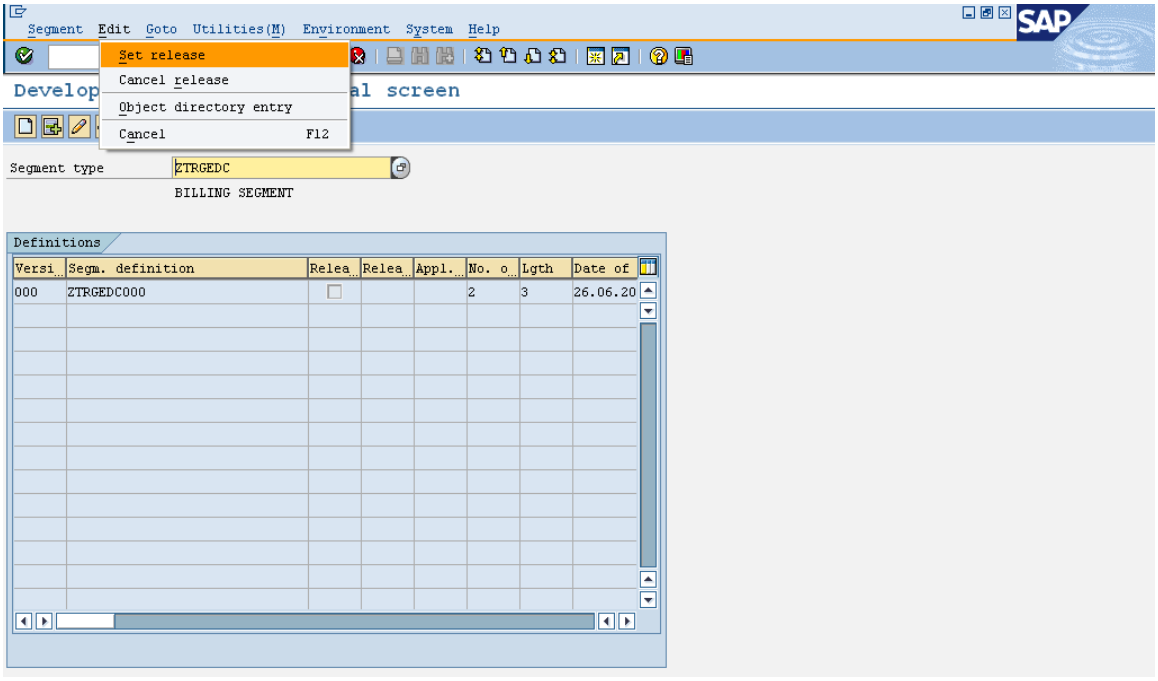
.

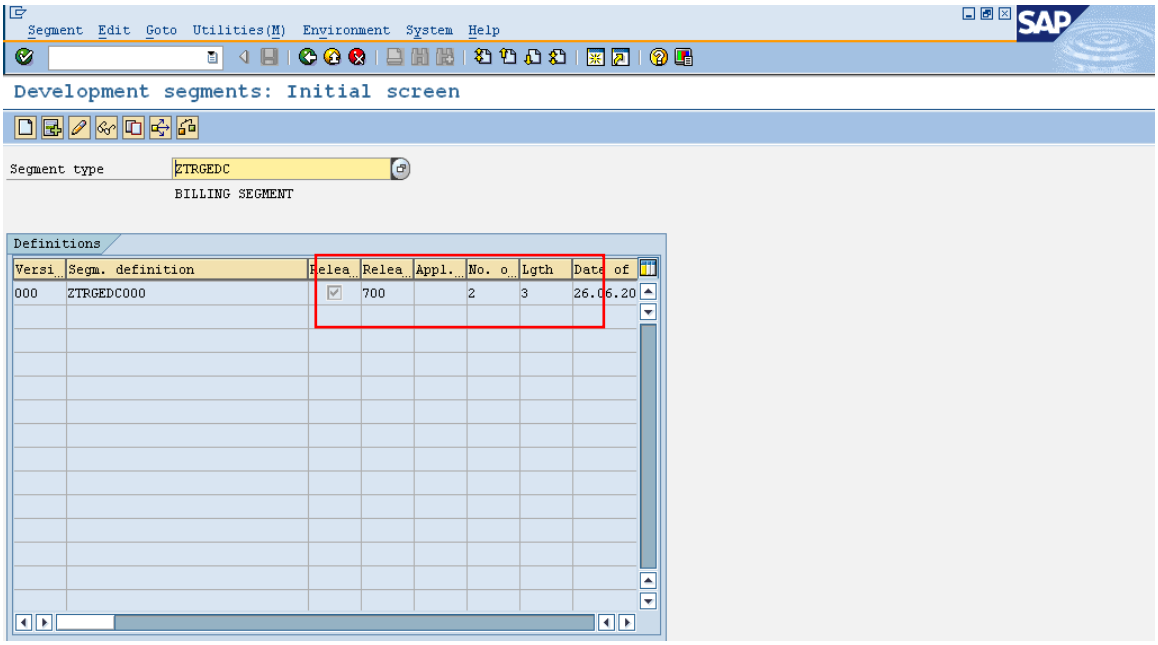

Segment definition Edit Goto System Help
Development segments: Create segment definition
New Fields
Segment type attributes
Segment type ZTRGEDC
Qualified segment
Short Description BILLING SEGMENT
Segment type ZTRGEDC: Change persons involved
Person responsible NTWDEV
Processing person NTWDEV
Segm. definition
Released
Last Changed By
Pos. Field Name Data element ISO c... Exp...
1 KTGRD KTGRD
2 MANSP MANSP


Segment definition Edit Goto System Help
Development segments: Create segment definition
New Fields
Segment type attributes
Segment type ZTRGEDC
Qualified segment
Short Description BILLING SEGMENT
Create Object Directory Entry
Object R3TR TABL ZTRGEDC
Attributes
Package Z15988_PACKAGE
Person Responsible NTWDEV
Original System EC6
Original language EN English
Local Object
Lock Overview
Click here
Segm. definition
Released
Last Changed By
Pos. Field Name Data element
1 KTGRD KTGRD
2 MANSP MANSP


Segment definition Edit Goto System Help
Development segments: Create segment definition
New Fields
Segment type attributes
Segment type ZTRGEDC
Qualified segment
Short Description BILLING SEGMENT
Segm. definition
Last Changed By
Prompt for transportable Workbench request
Table ZTRGEDC
Request EC6K904902
Workbench request
Short Description purple's request
Own Requests
Pos. Field Name Data element
1 KTGRD KTGRD
2 MANSP MANSP


Segment definition Edit Goto System Help
Development segments: Change segment definition ZTRGEDC000
New Fields
Segment type attributes
Segment type ZTRGEDC
Qualified segment
Short Description BILLING SEGMENT
Segm. definition ZTRGEDC000
Released
Last Changed By NTWDEV
Pos. Field Name Data element ISO c... Exp...
1 KTGRD KTGRD 2
2 MANSP MANSP 1
Segment is saved and assigned request number
Segment definition ZTRGEDC000 was saved with transport request EC6K904902
EC6 (2) 800
IGINECC6
INS



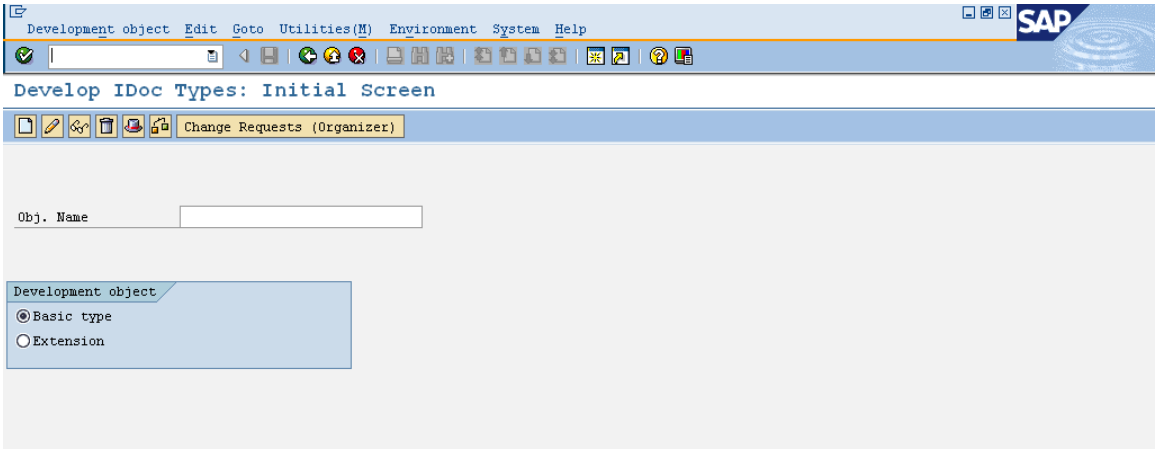


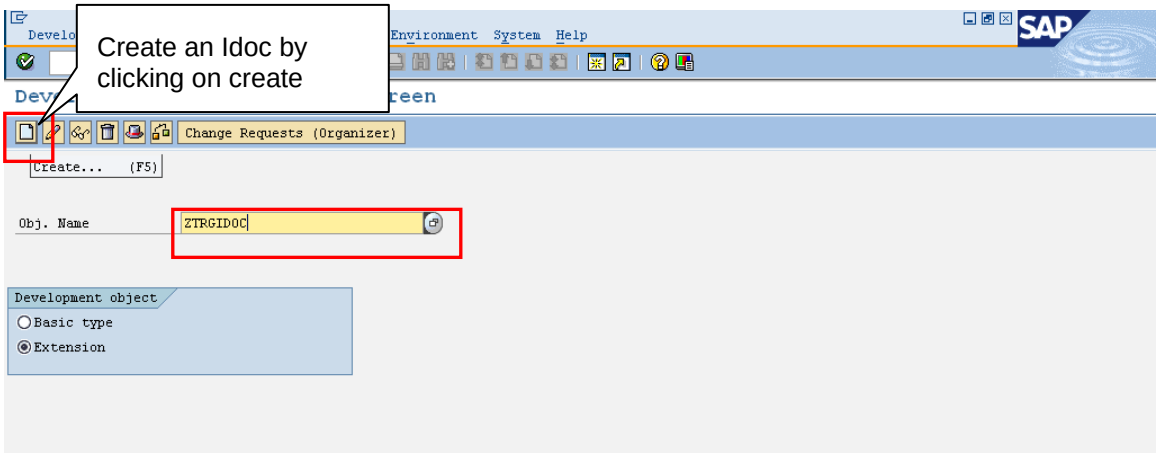
So segment is created and set to release. After you release the system automatically updates the **version release** (Here it is 700 in above diagram). Only one unique version segment is released per version i.e. here ZTRGEDC with release 700 exists once only and no changes are allowed until it is in released status (the Tick mark in Released tab).

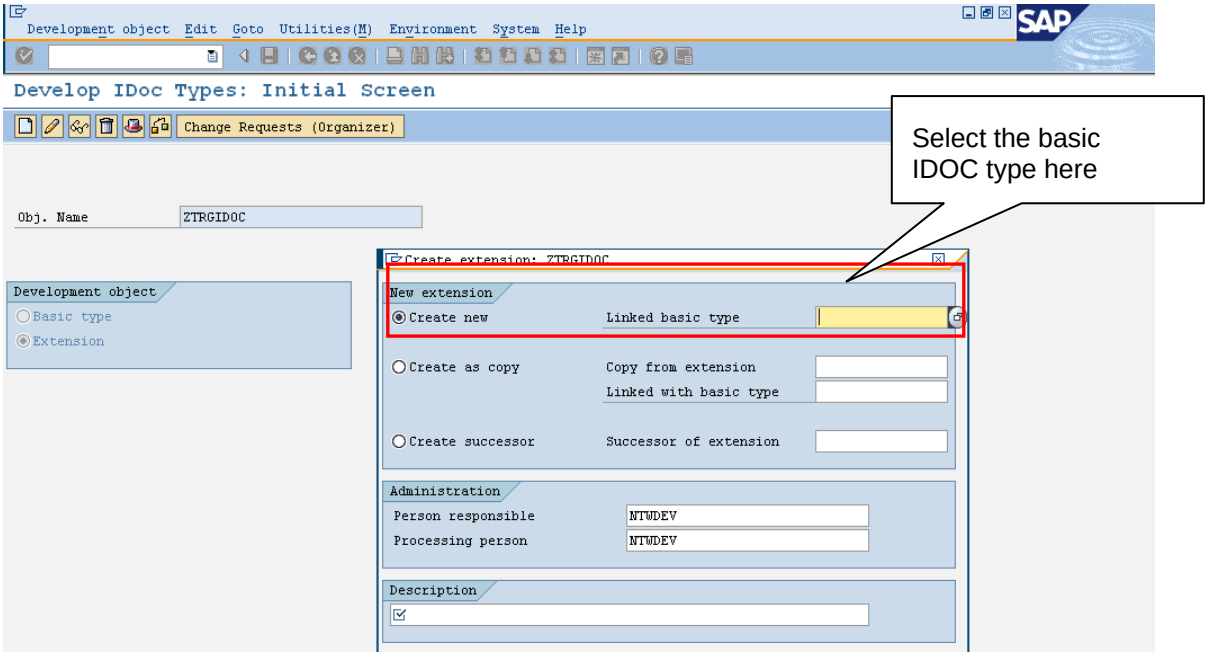
So in order to make changes unrelease the segment and make changes and once again “set release” it. So now ZTRGEDC segment is created and set to release. Now we need to create the IDOC type.

Create IDOC Type

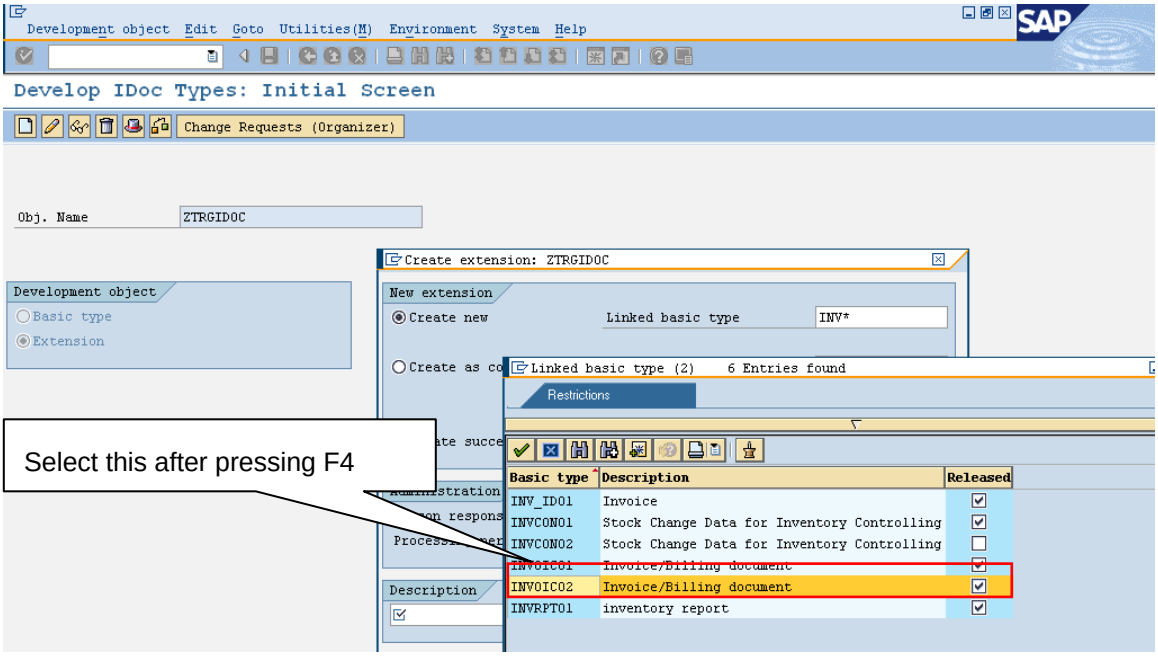
Go to WE30 to Create IDOC Type screen is as shown

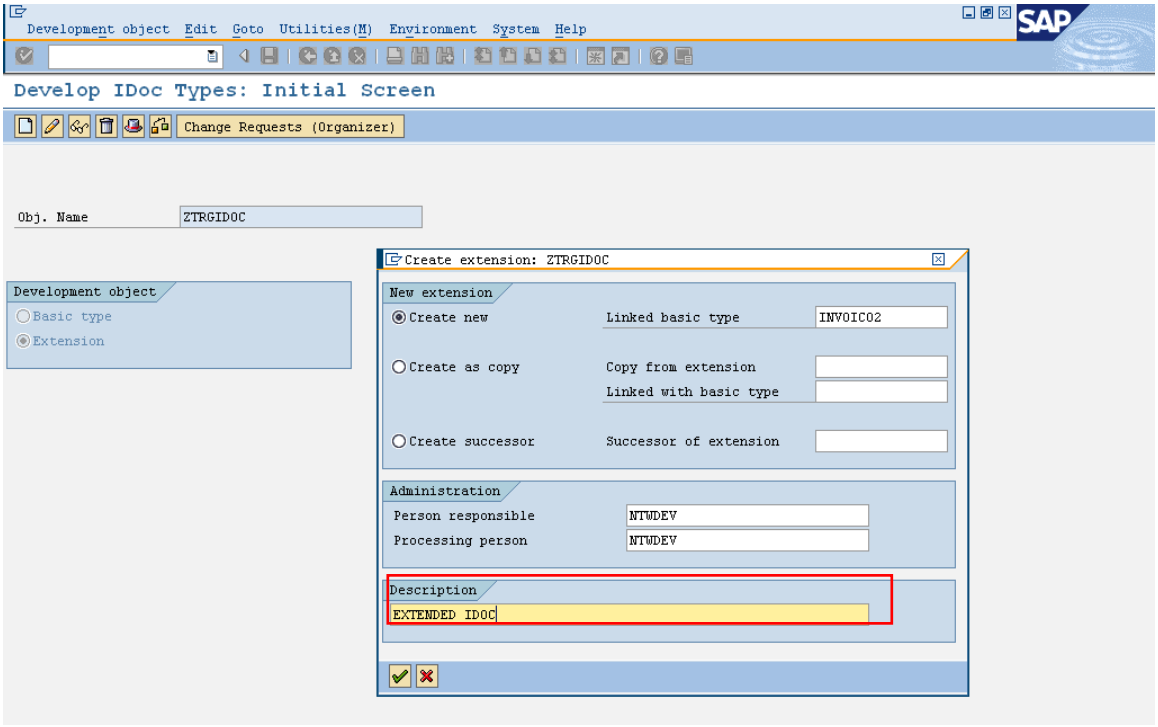




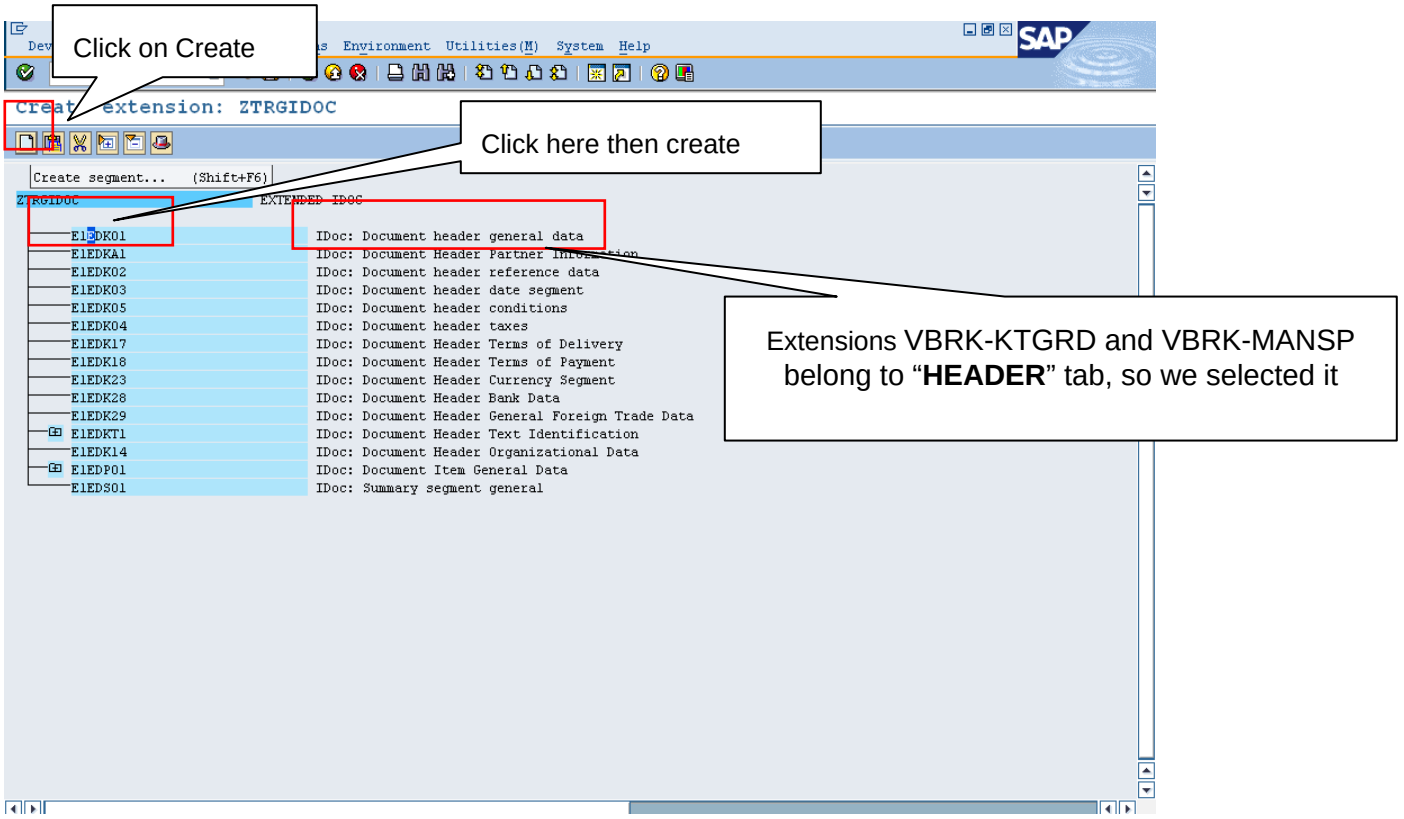


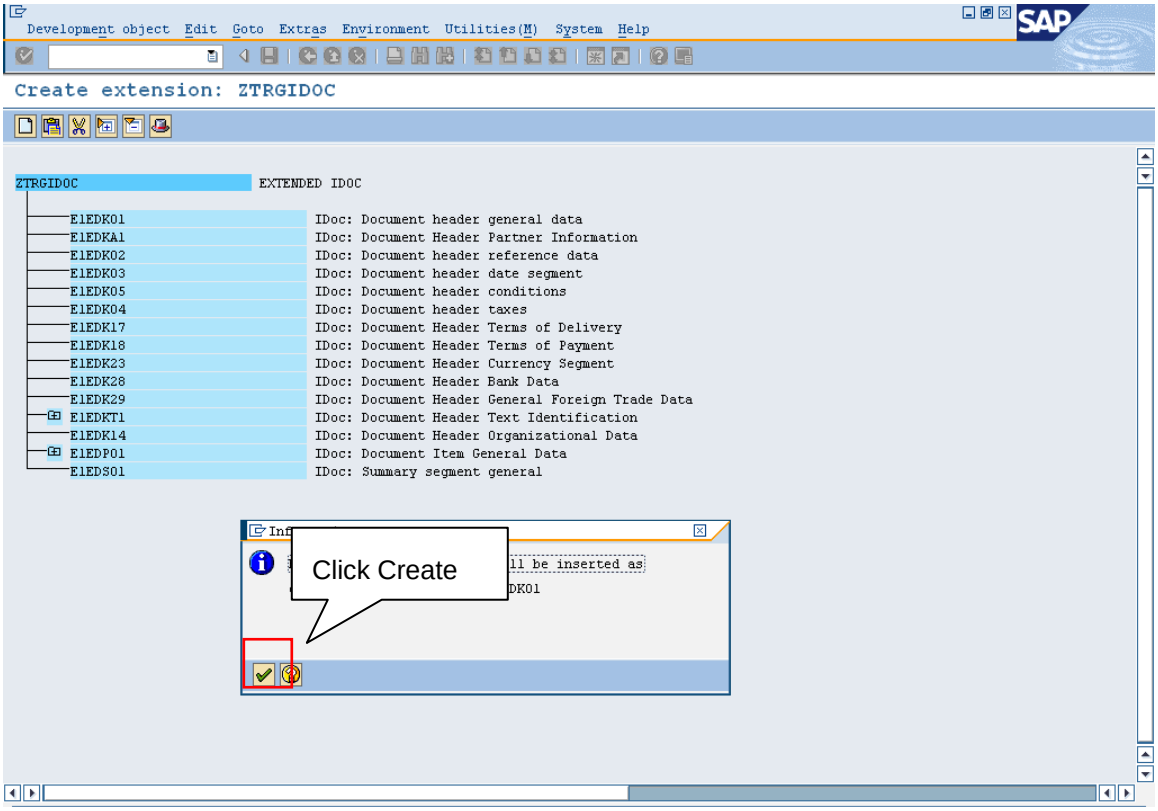
Since we are creating the existing Idoc, we assign invoice02 here.



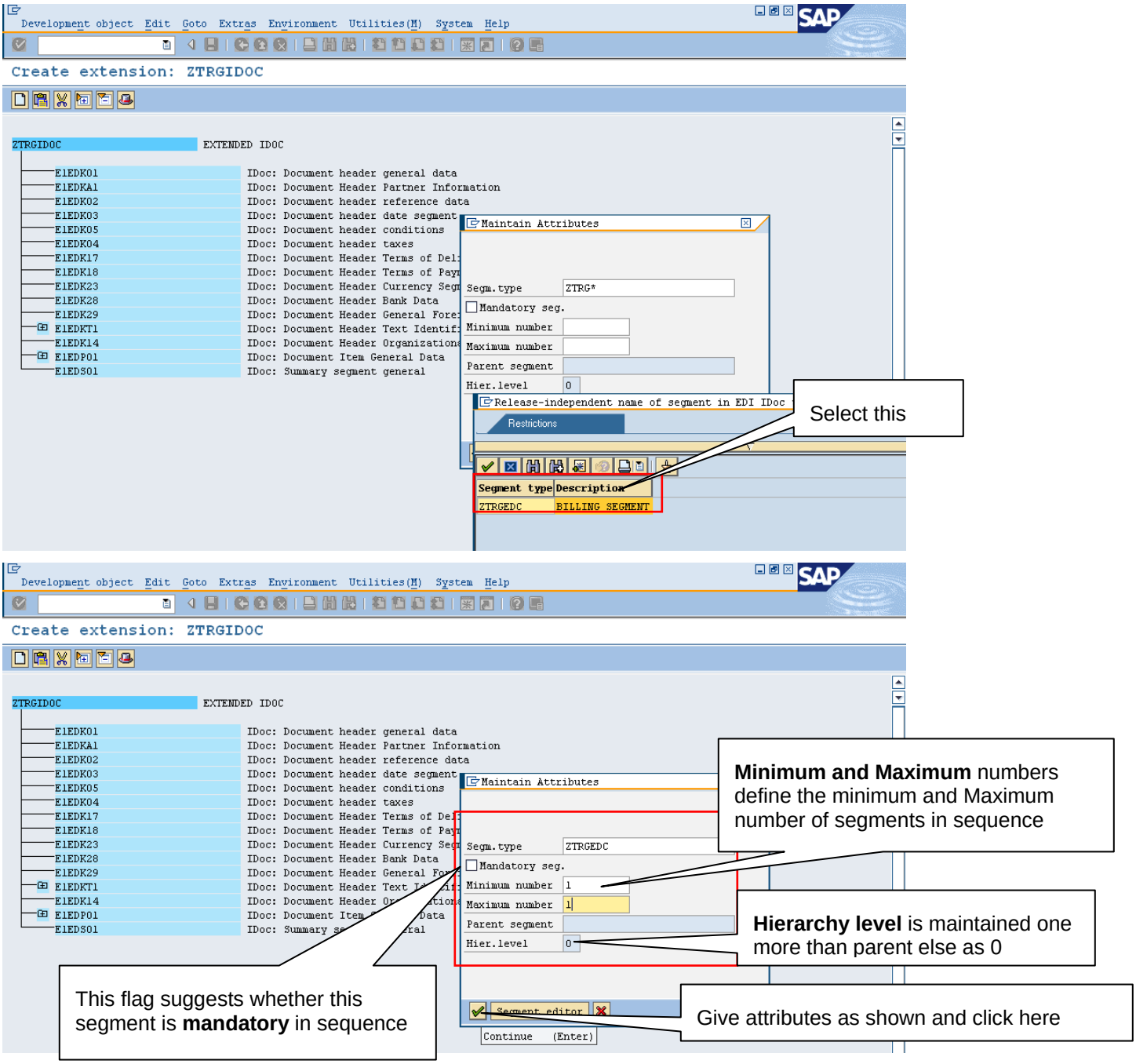


We get the following screen showing the set of segments. Since the extensions VBRK-KTGRD and VBRK-MANSP belong to “**HEADER**” tab in transaction VF02 the extension is done for the relevant segment type E1EDK01 related to “**Header General Data**”. We extend it as shown.

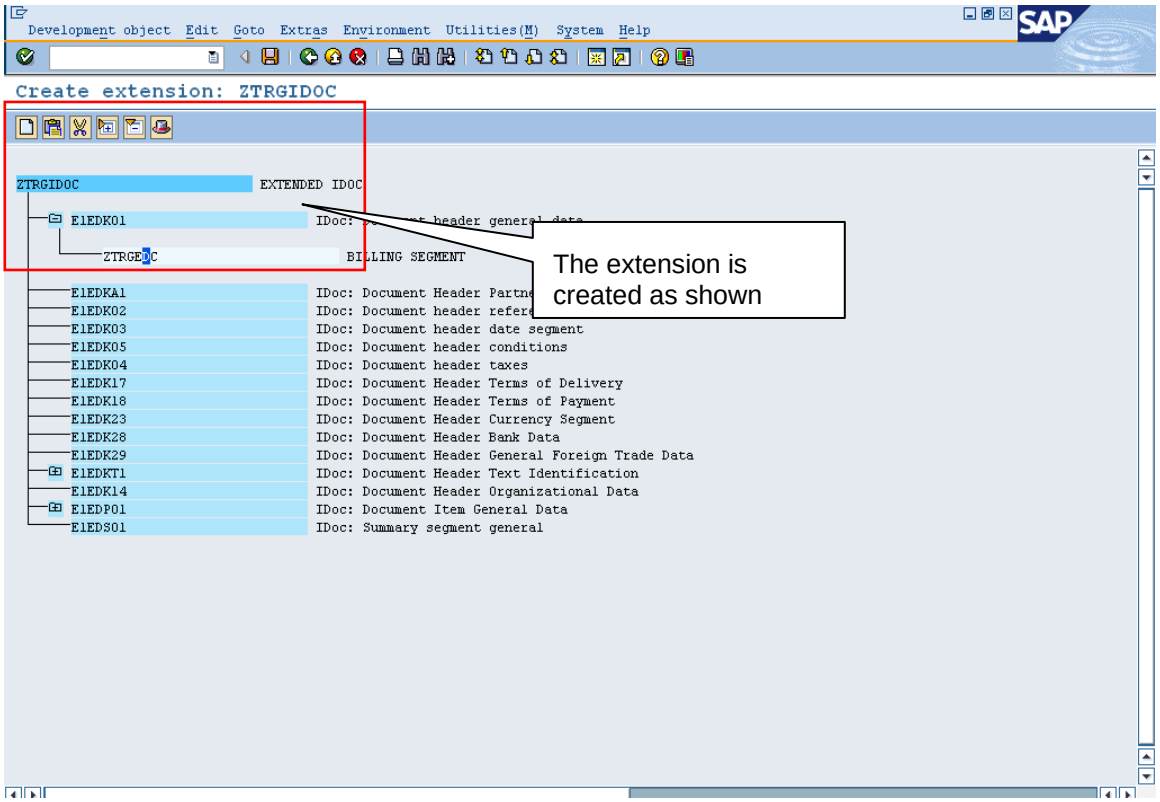


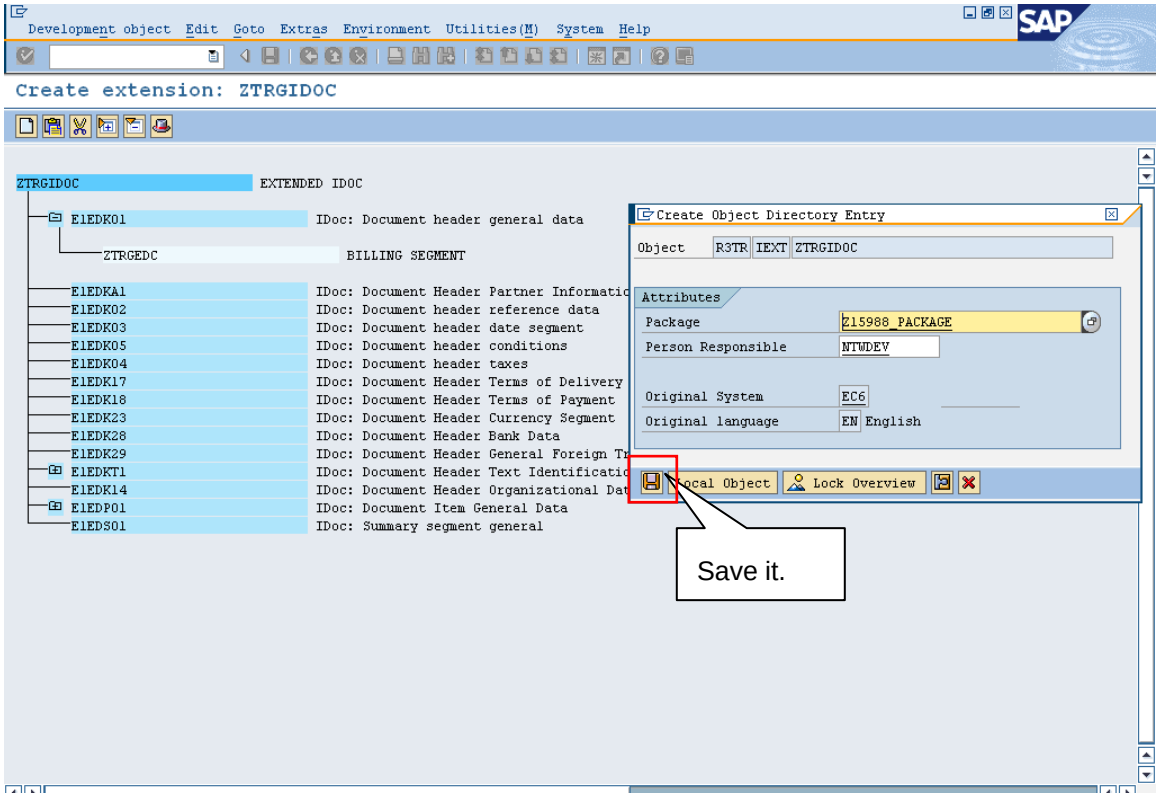


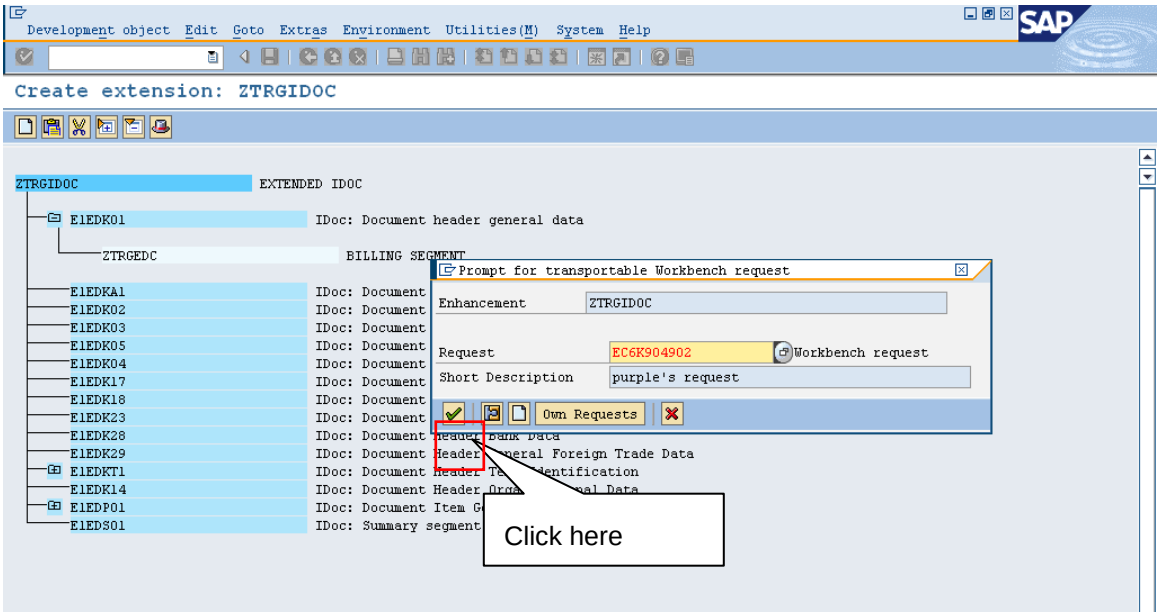
We get a popup to maintain attributes as shown



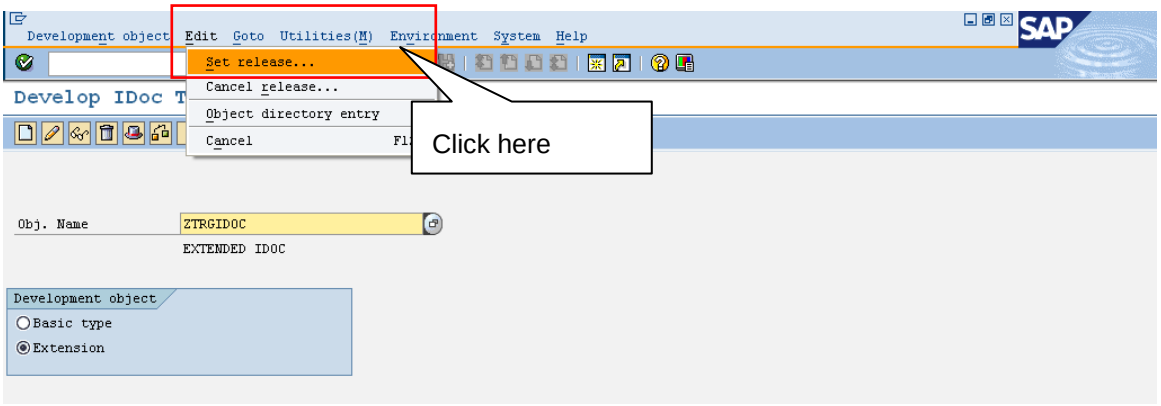
The attributes include whether it is a mandatory Segments. The Maximum and minimum numbers specify the no of times the segment can be repeated in sequence. The Hierarchy level suggests the parent relation ships. For a segment which do not have a parent segment either have hierarchy level 1 (if they cannot be repeated) or hierarchy level 2 (if they can be repeated). Segments which have a parent segment have a hierarchy level which is one higher than that of their parent.

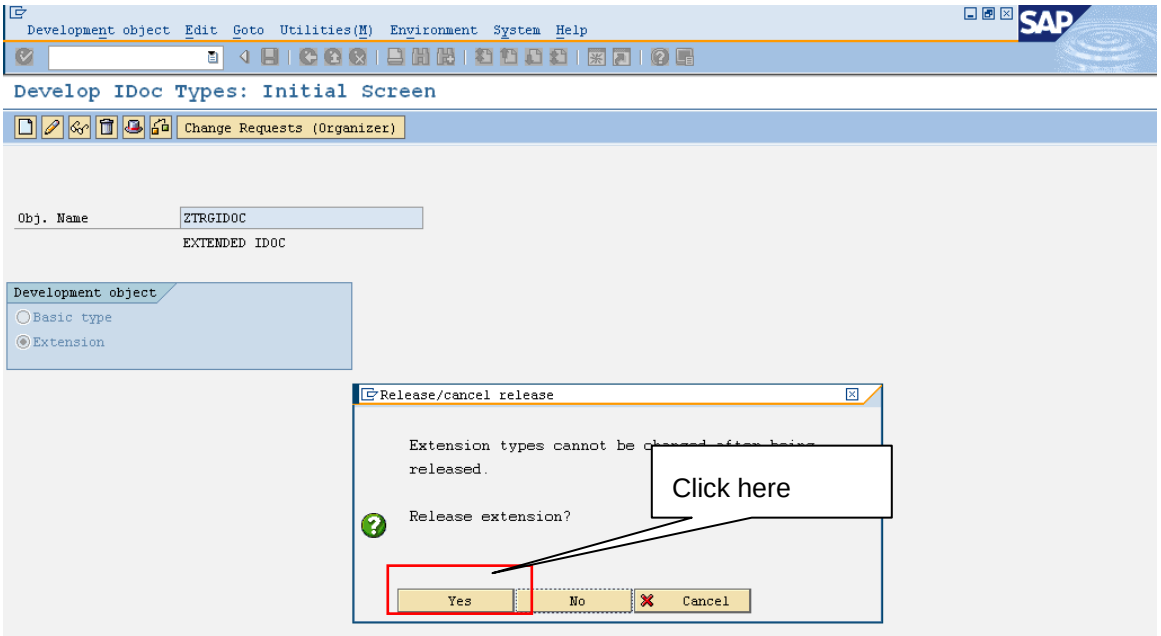






Press on “back” and Release the Idoc as shown.





So **Idoc type** is created and released.

Create Logical Message Type

The message type determines the technical structure of the message. Message type determines data contained and also determines process involved in flow of message across Distributed environment. The Message Type controls Process Code, in turn drives Function Module to determine the content of the message. It also controls IDocs processing (batch, immediately etc).

There exist standard message types predefined by SAP. Some of them are

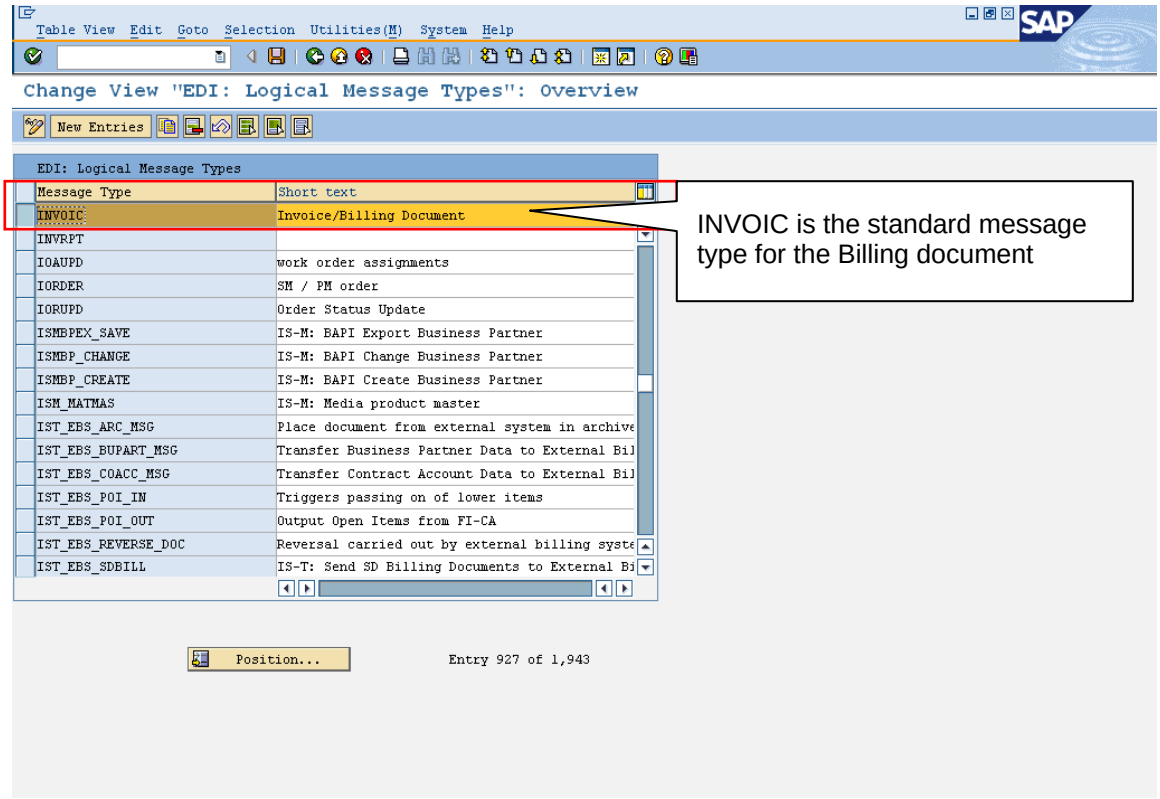
CUSTOMER – **DEBMAS** SALES ORDER – **ORDRSP**

VENDOR – **CREMAS** PURCHASE ORDER – **ORDERS**

MATERIAL – **MATMAS** INVOIC - **INVOICE**

We can also create the customized logical message type according to the requirements. Since we are using the invoice here we can use the existing message type INVOIC.

In transaction WE81- Logical message type we can see for message type INVOIC



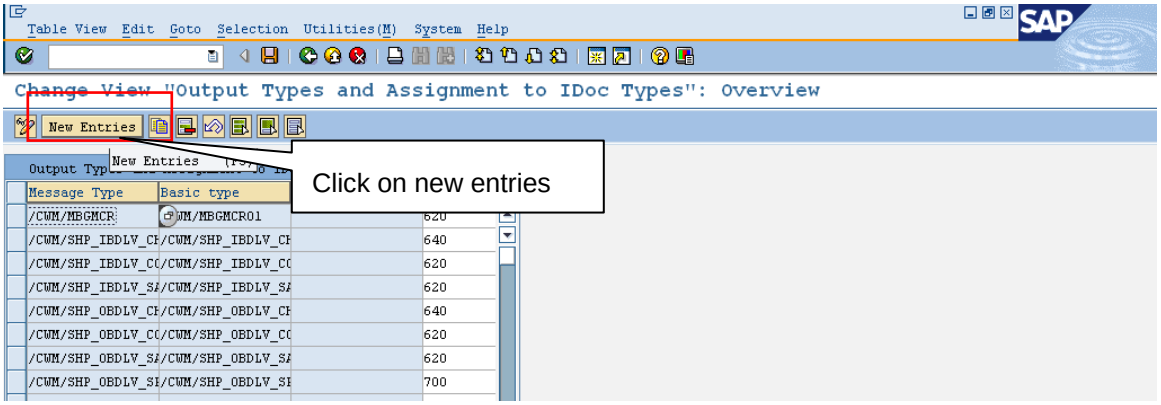
Since we have a standard message type we did not create it else we can create using NEW ENTRIES tab.

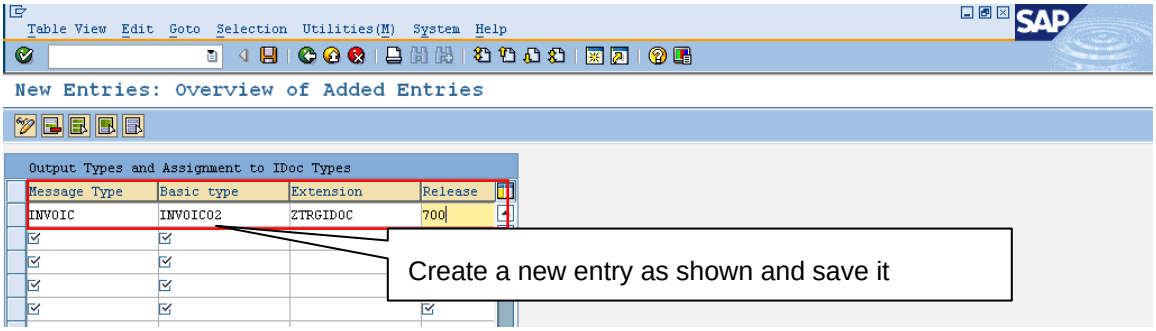
Create Output Types and Assignment to IDOC Type

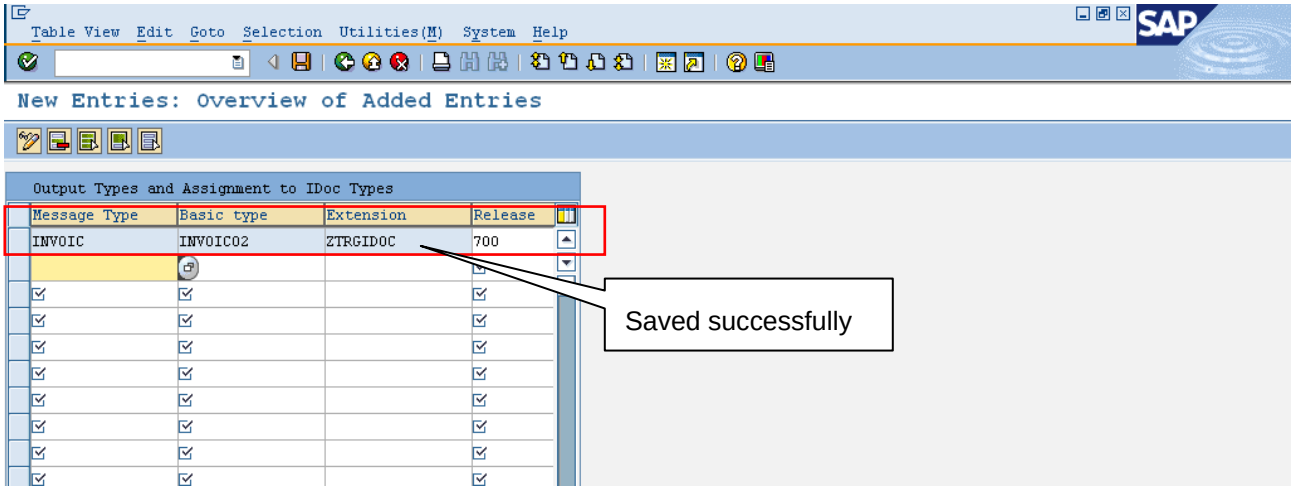
Go to transaction WE82- Output types and Assignment to IDOC types. Check for the entry with

Message Type	**INVOIC**
Basic type	**INVOIC02**
Extension	**ZTRGIDOC**

If it exists already we will use it else create it using new entries.



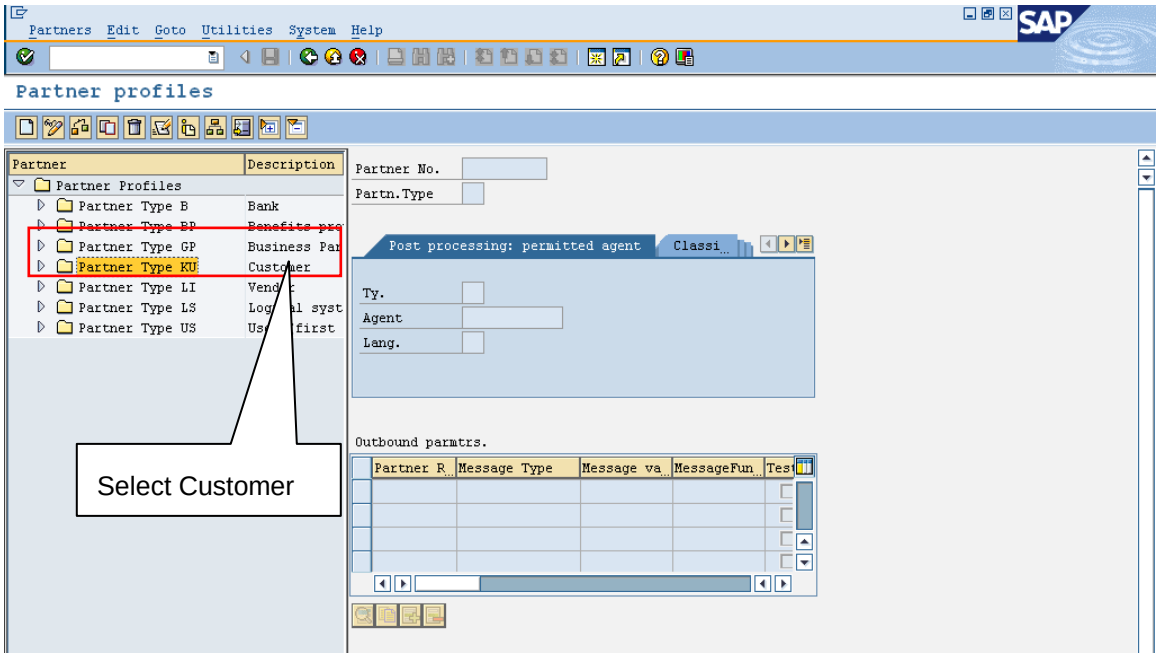




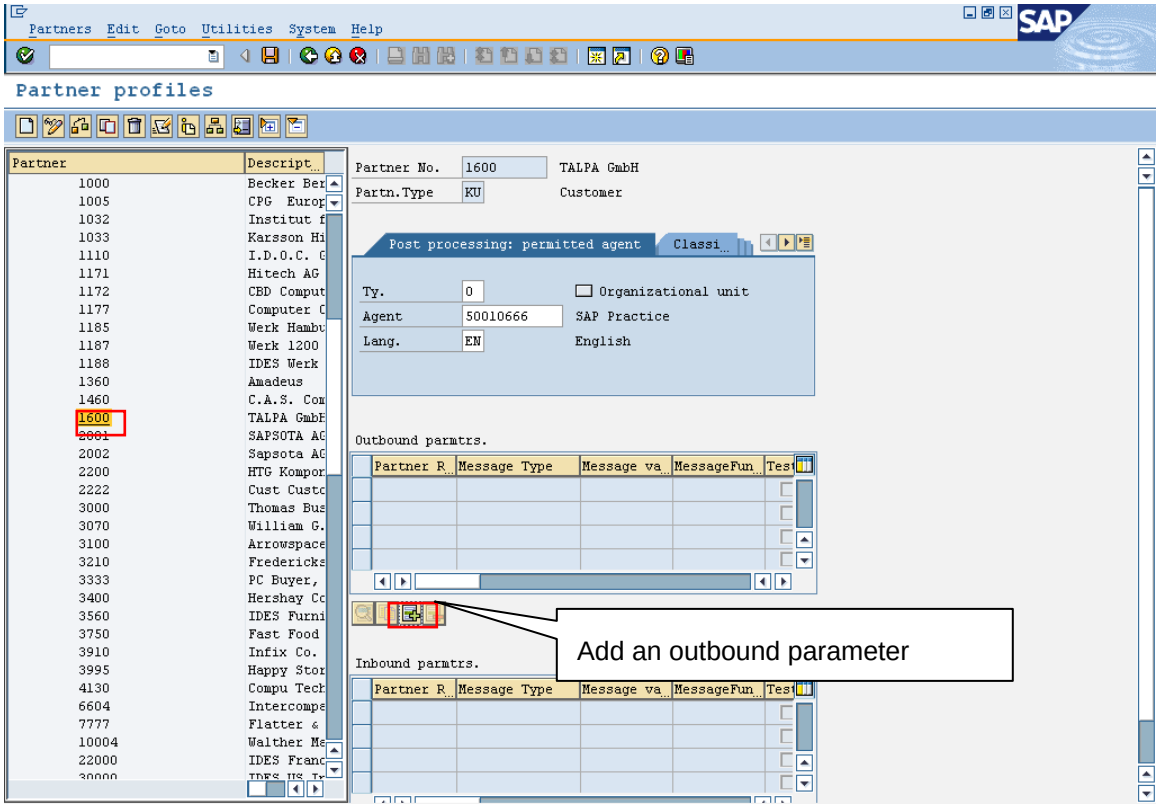
So we created Output types and Assignment to IDOC type as an entry. This is used in the Idoc processing.

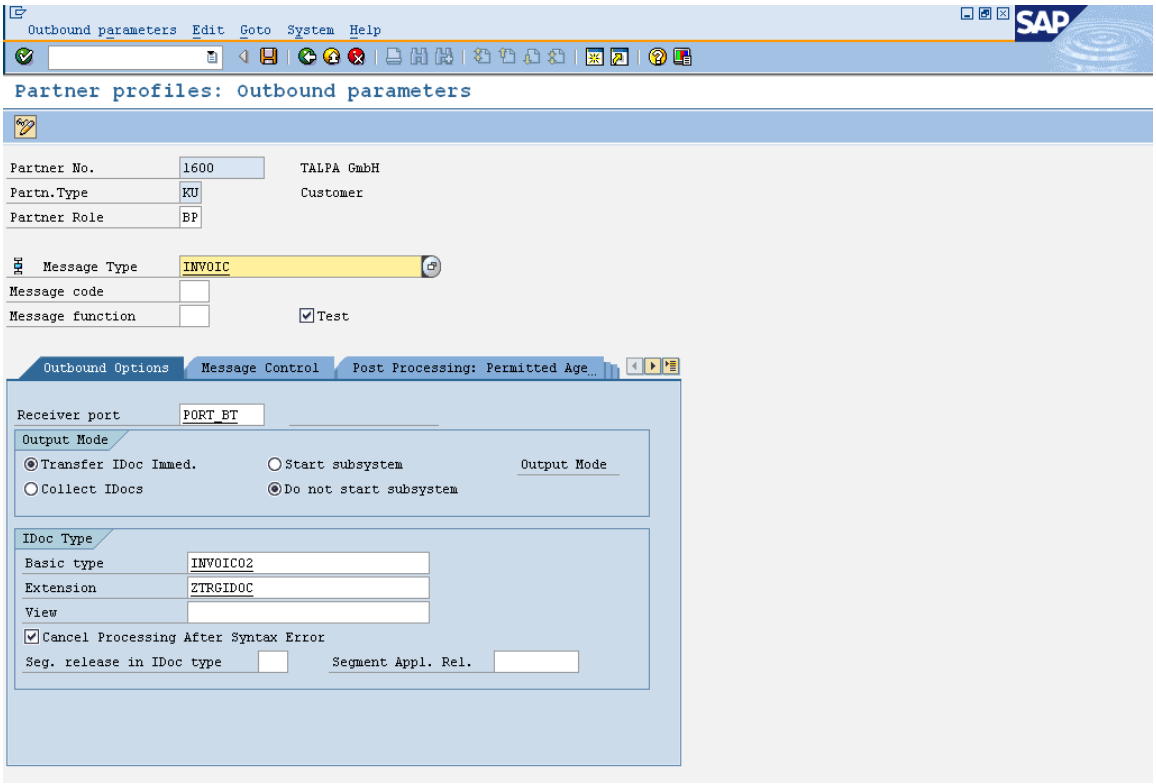
Creating Partner Profile

Then go to WE20 to Creating partner profile, as it is an invoice, Select **customer**



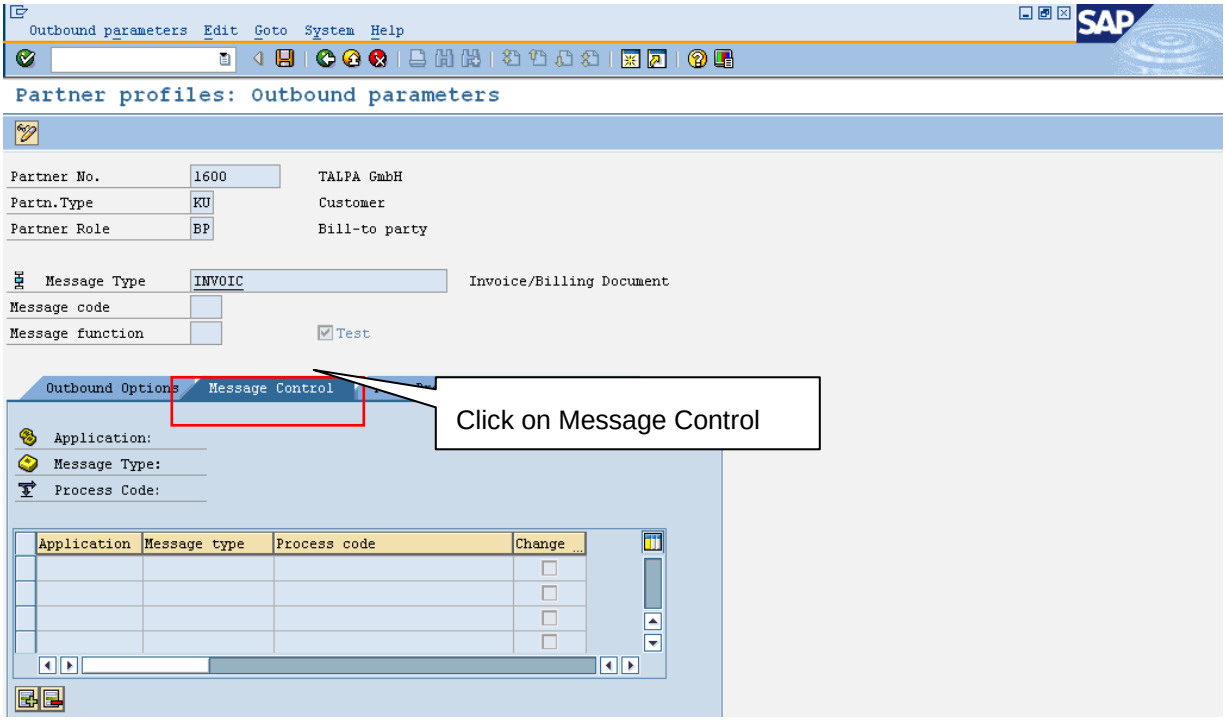
We chose 1600 as an example, for general you can choose any customer



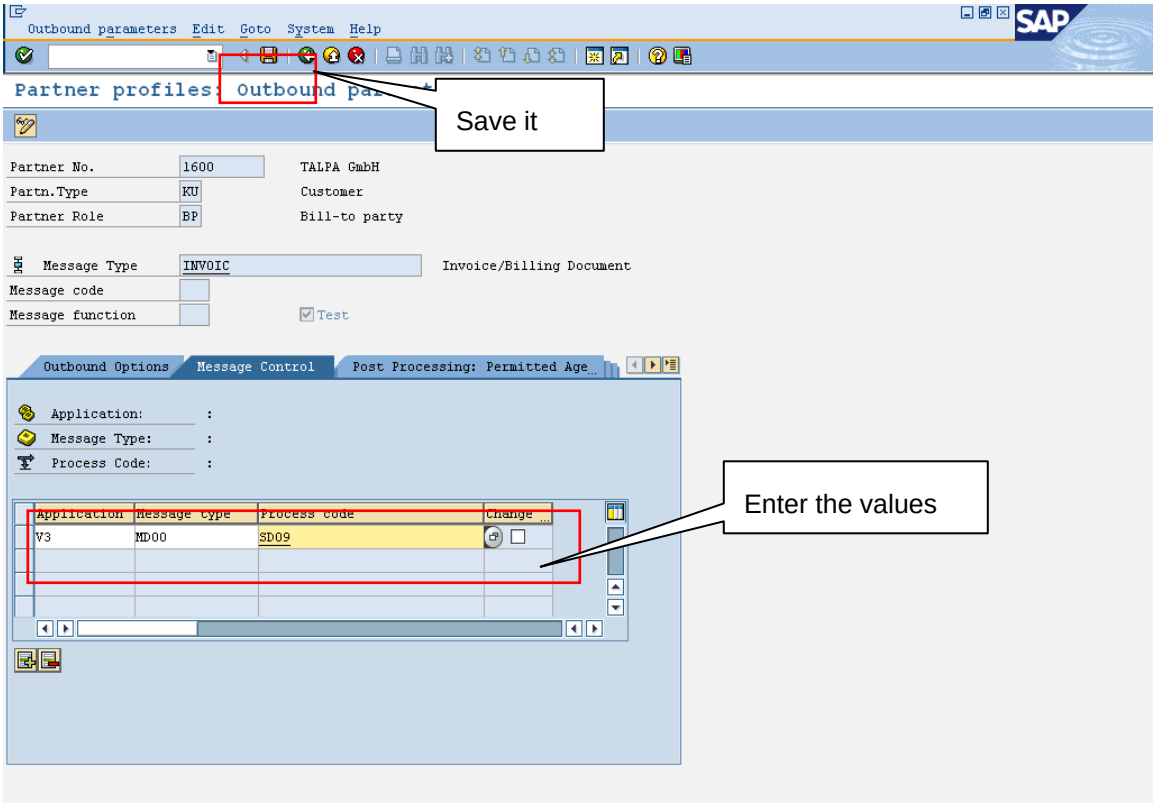


In real time, we do not use test flag in message control to send an Idoc. We used this test flag enabled only to show that we are creating the IDoc for 'testing' purpose on the outbound side.

The Message type, Partner Role, Receiver port, Basic and extended Idoc type with output mode "Transfer Idoc immediately" is given as a partner profile as shown in above document.



Click on the message control to give the values of application, Message type and process code.



These key fields application, Message type and process code assigned in the message control together uniquely identify a message type which uniquely identifies an IDoc type.

The key fields application, Message type and process code are used as

Application: The Application specified in message control determines the output type and uniquely identifies a message type which can be assigned uniquely to an IDoc type.

For Example: 'EA' is used for 'Purchasing RFQ' in Materials Management (MM).

V3' is used for 'Billing' in Sales and Distribution (SD).

Message type: Message type along with the application uniquely identifies a message type which can be assigned uniquely to an IDoc type.

For Example: 'LAVA' = 'Shipping notification' in dispatch (application 'V2').

Process code: The Process code is used by an IDoc Interface to determine the application selection module which converts the SAP document into an IDoc.

For Example: ME10: Purchase order (MM)

Here the values are chosen depending on the description having billing or invoice.

The application is **V3: Billing**

The Message type **MD00: MKS -Invoice O/P type**

The process code **SD09: INVOIC: Invoice**

We get the function module embedded in the process code where the functionality is coded. This is a **Top – Down** approach of finding the relevant function module to write User Exit. We can also find the process code and relevant function module in the **Bottom - Up** approach.

The function module which is embedded in each process code follows a naming convention **"IDOC_<OUTPUT / INPUT >_NAME OF BASIC TYPE"**. This helps in finding the process code and then the other attributes. For our example here we do an outbound with basic Idoc **INVOIC**. So the function module will be **"IDOC_OUTPUT_INVOIC".**

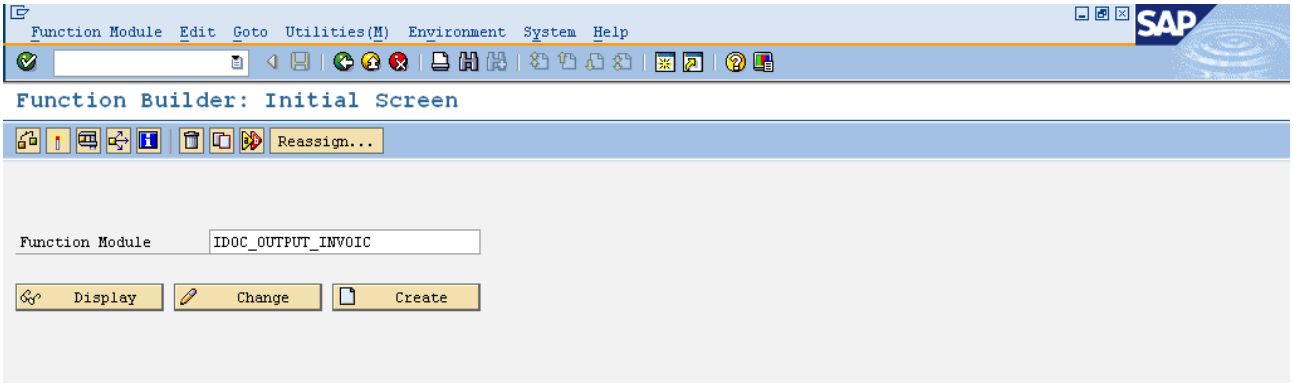
Search this in Function builder **SE37** and find the relevant process code in transaction codes **WE40** (inbound) or **WE41** (outbound). Here we followed the top approach. Many times depending on the description of the application, message type we find the relevant key fields and link them up.

But if we could not find the relevant function module then there is another method of finding it. The process is explained below using the Transactions **SMOD** and **CMOD**. We can use this method as an alternative. We choose among the **Bottom - Up, Top – Down** and **SMOD** depending on the scenario.

Finding and Updating Customer Exit Using SMOD

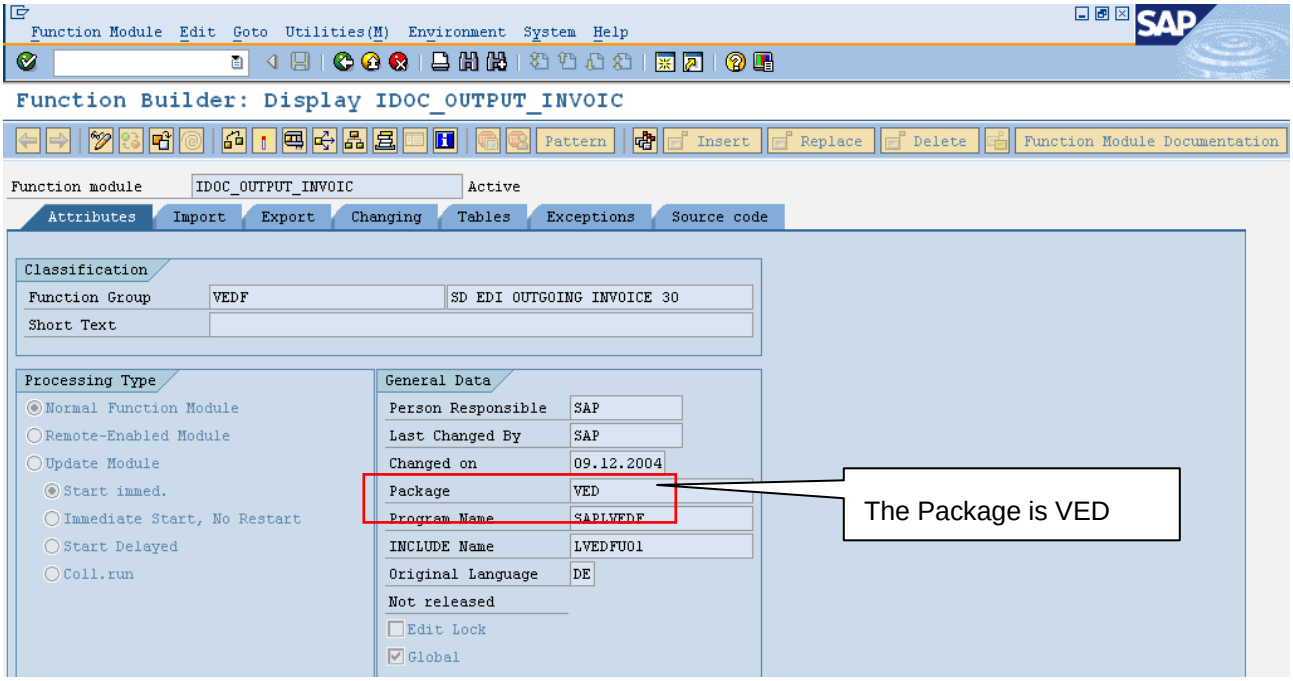
The Enhancement for the transaction can also be found using the transaction **SMOD.** Here we need to give the package to find the find the exact enhancement and the respective function module which serve our purpose.

So go to SE37 to find the package of “IDOC_OUTPUT_INVOIC”.

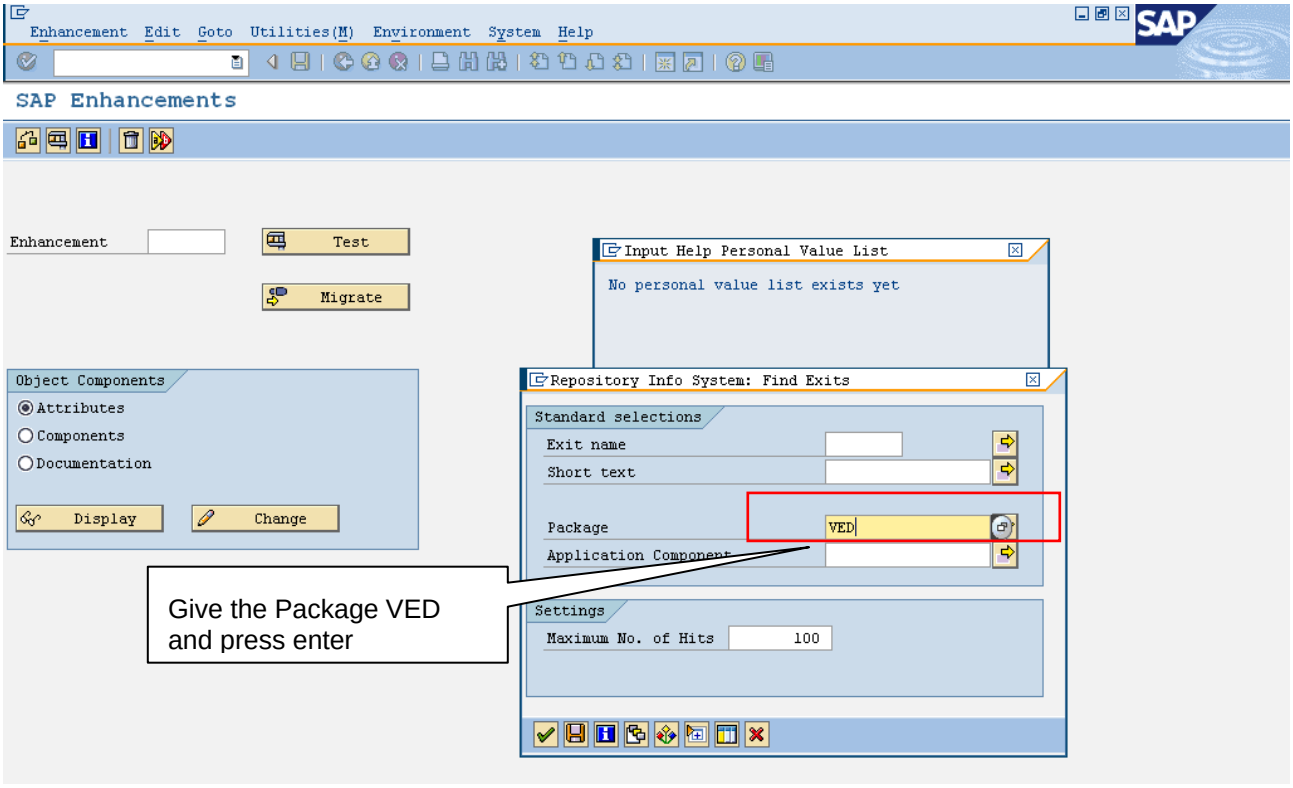


After clicking on display. Go to the “Attributes” tab and look for the package name.

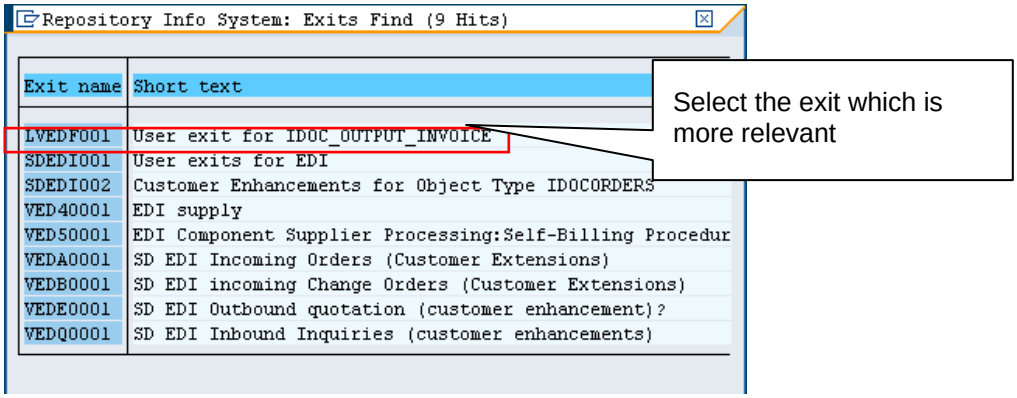
Here we get the Package as “VED”, Use this Package name in SMOD to find the respective function module.

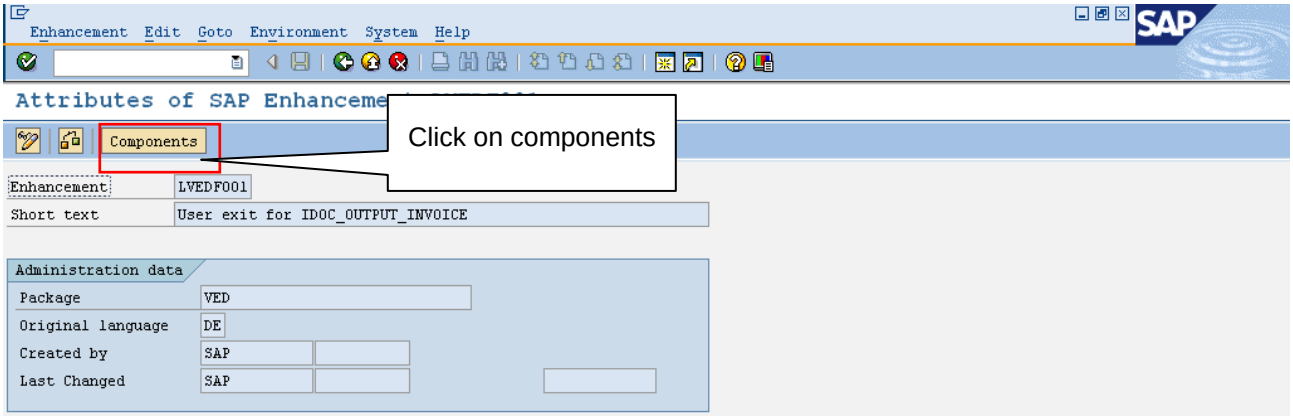


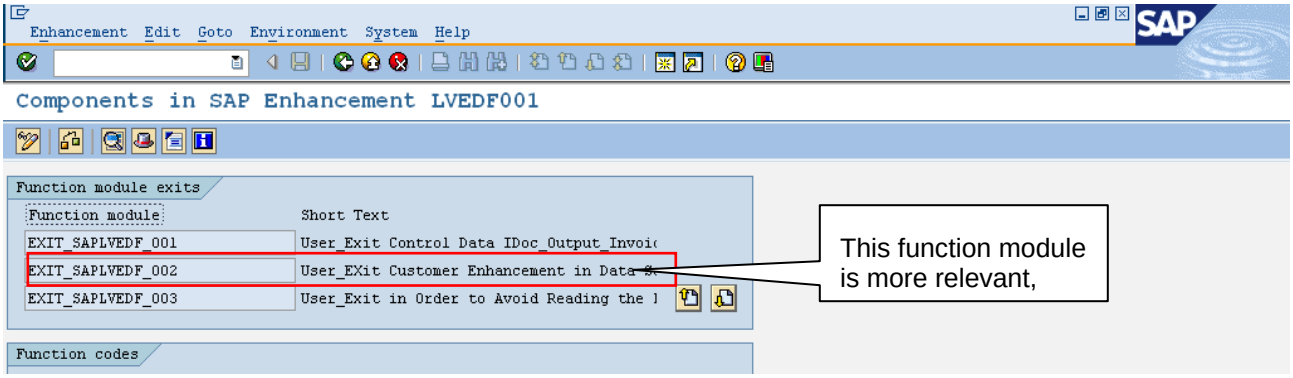
Go to Transaction SMOD and give the value of Package we got (here “VED”) in F4 help or utilities → Find.



We get the set of Exit names; Find the most suitable by the short text description. The description here ”User Exit for Idoc infotype” is more apt. So select the exit “LVEDF001” to get the list of function modules From which we select the option which is more apt.

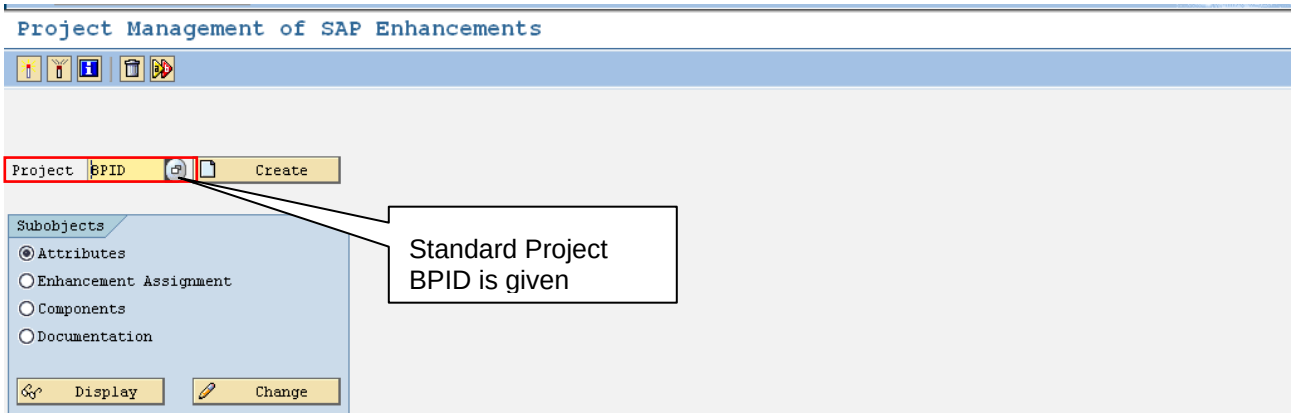


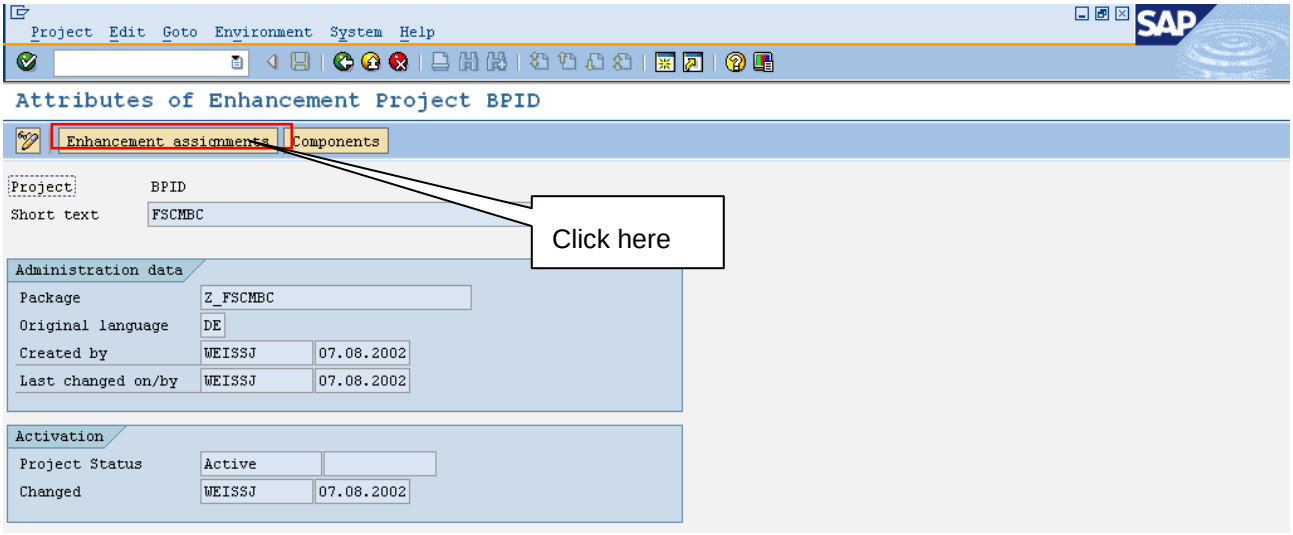


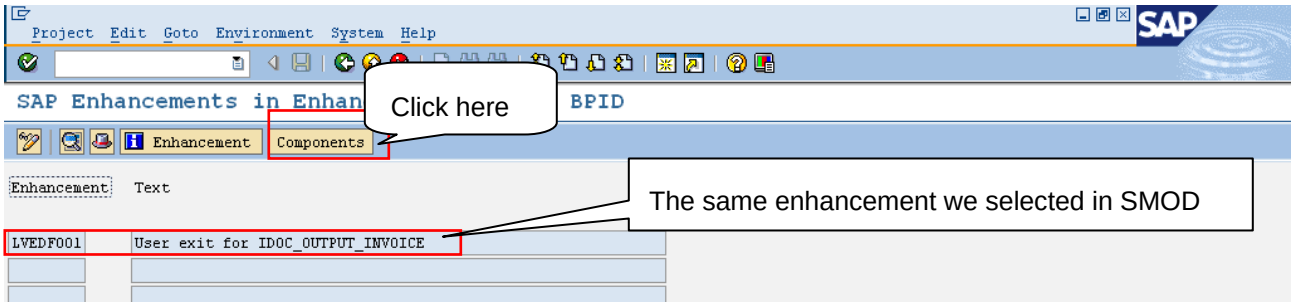


This is the function module where we are going to add the functional change.

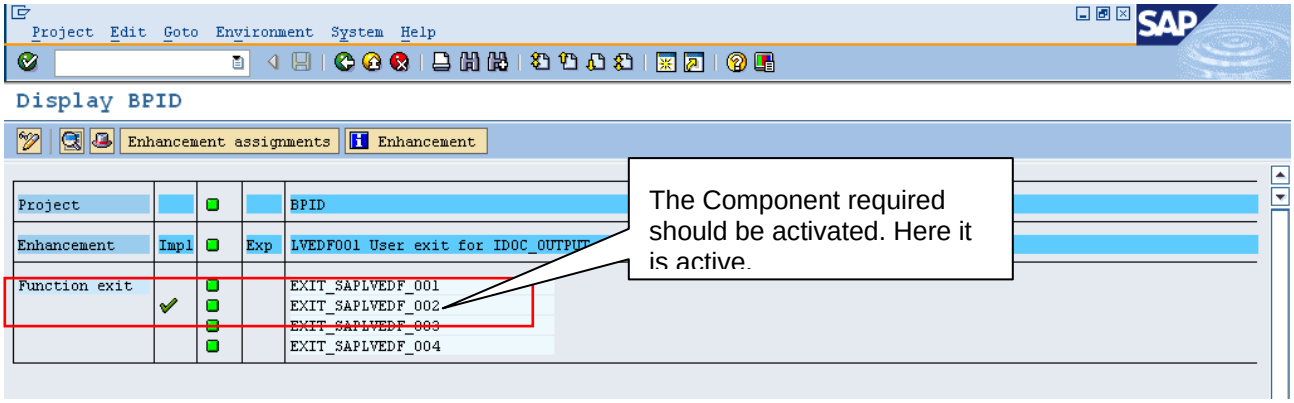
Go to CMOD and give “BPID” standard enhancement or you can create a new project with ‘LVEDF001’







The Enhancement is having our component active.



You can Click here to directly go to the function “exit_saplvedf_002” or follow the path as shown below.

You need to add the changes required in this customer exit. The process is explained in detail below.

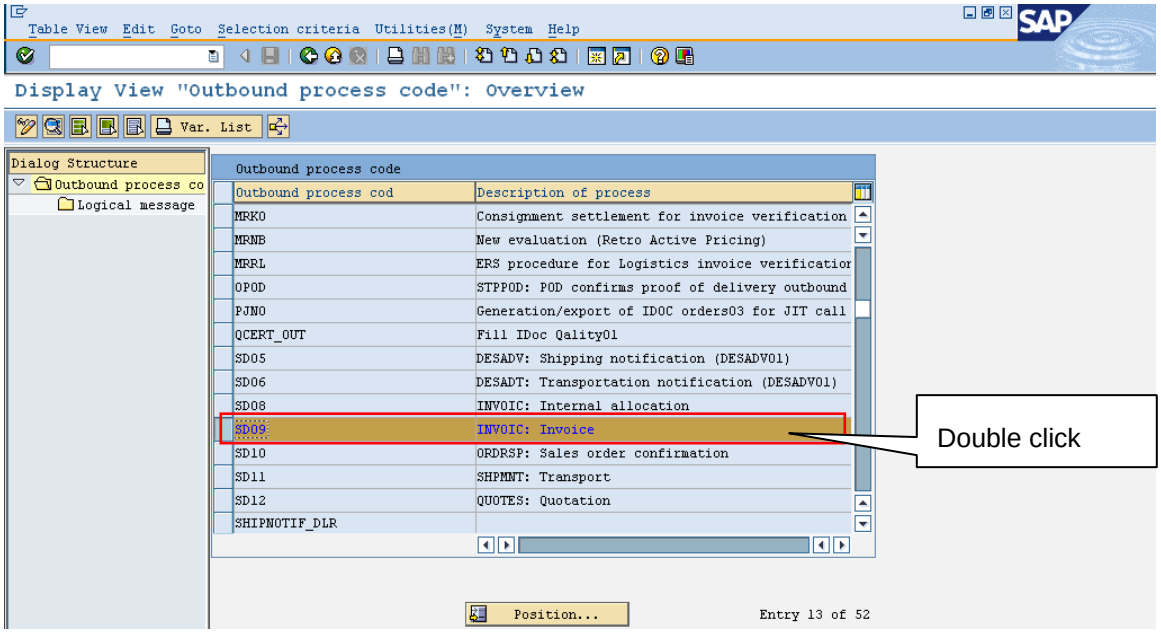
So the process using the Transactions **SMOD** and **CMOD** is explained above. Now we need to make the changes in the code according to the requirement.

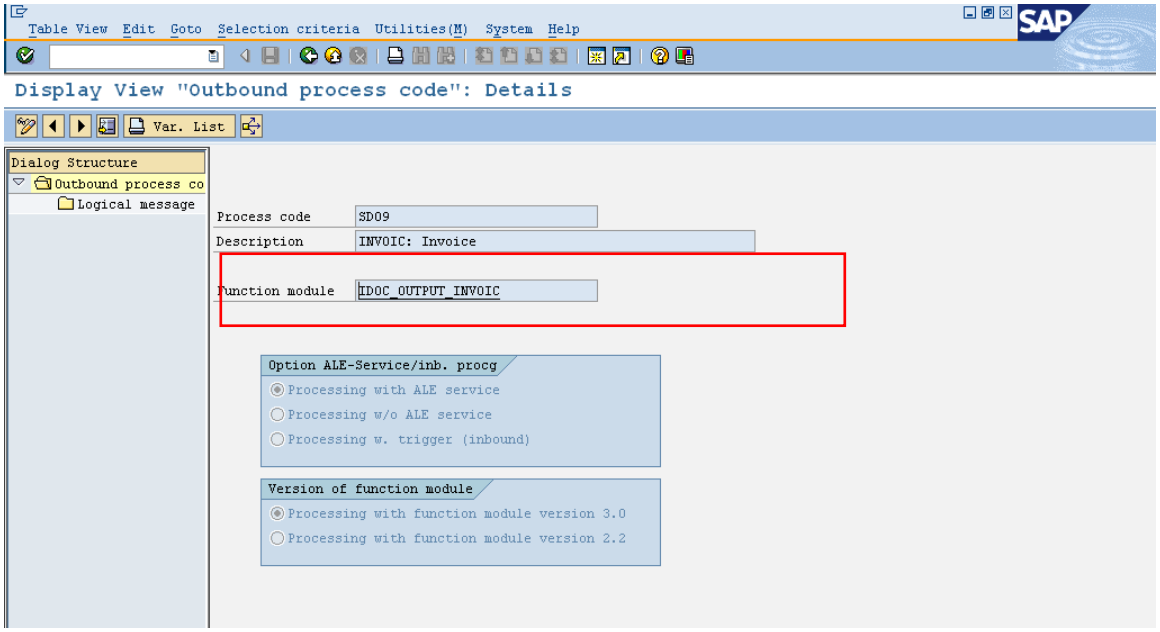
Outbound Process Code

We found the relevant user exit using SMOD and SMOD as “exit_saplvedf_002”.

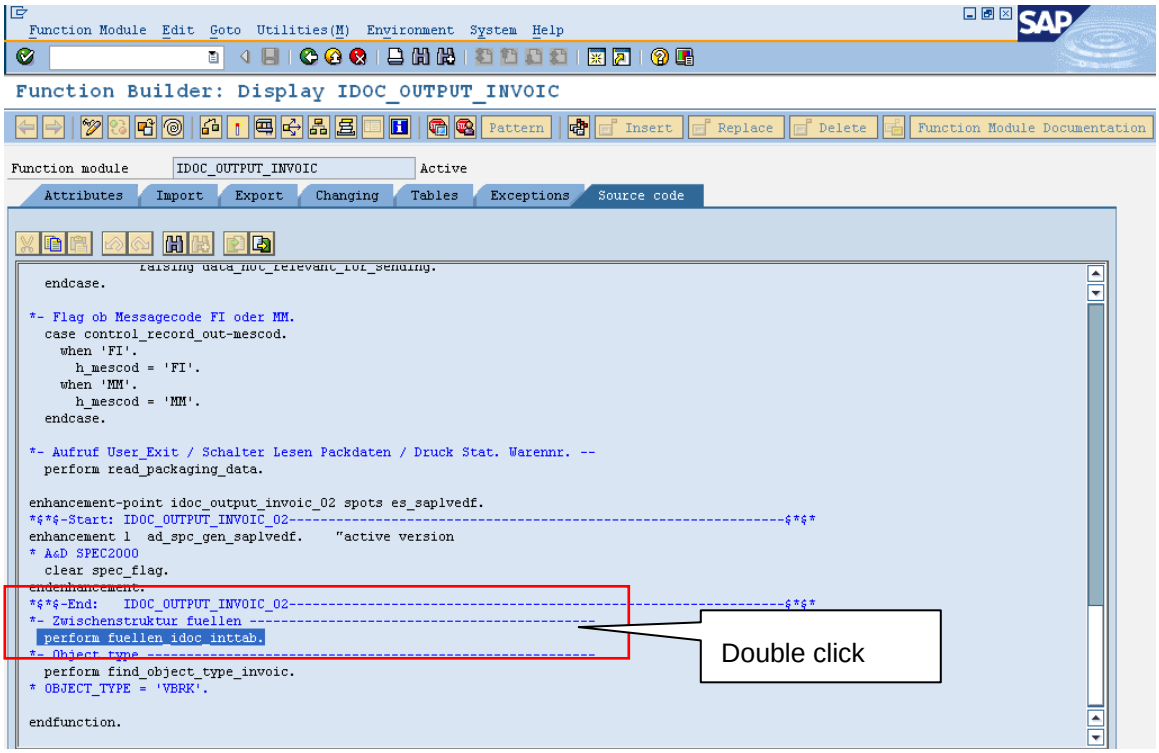
But in Top-Down or Bottom - Up we can find the relevant user exit do the following steps.

Go to transaction WE41 - Outbound process code. DOUBLE CLICK On SD09 as shown

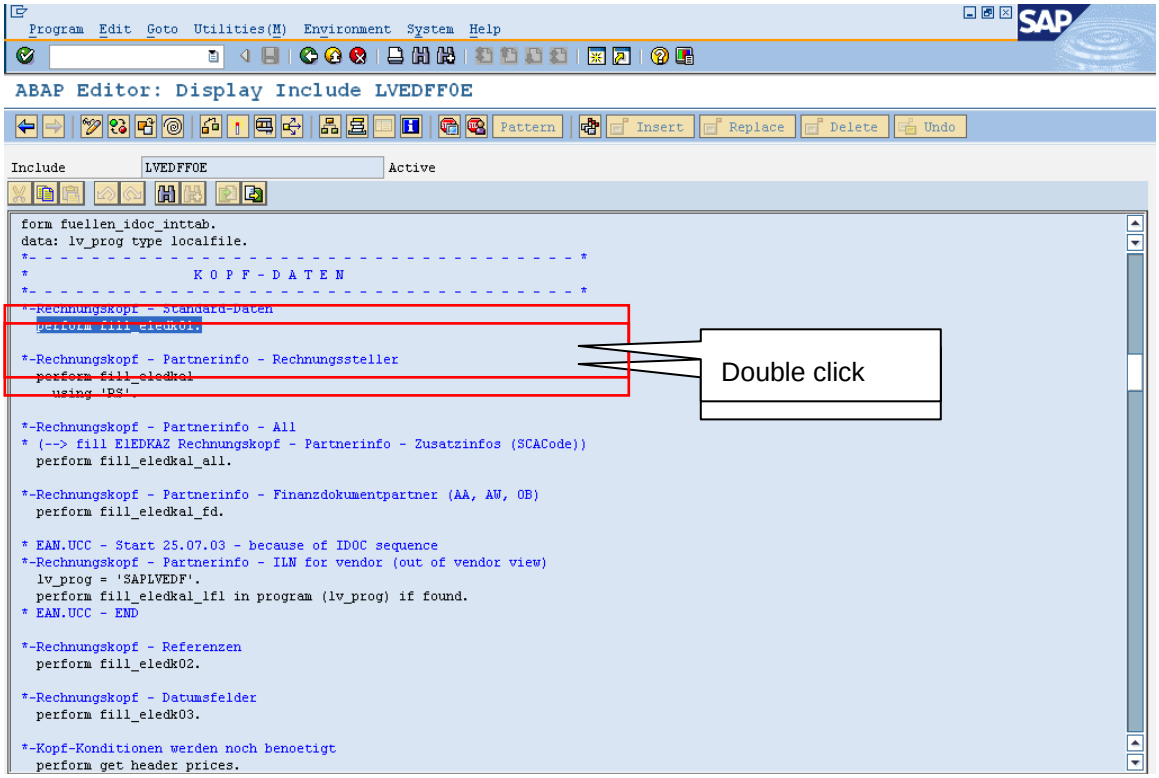




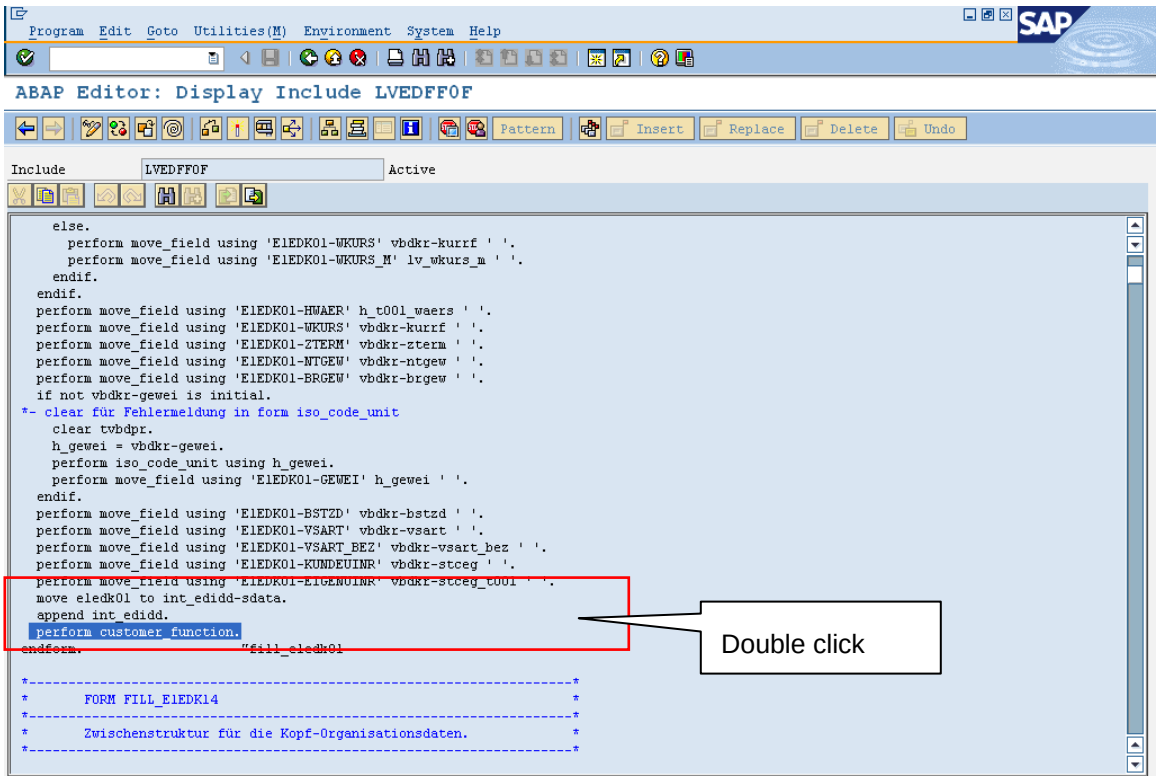
We can get into the function module to find the customer exit. So after getting into the function module how you find the relevant customer exit is defined below. Double click on the function module to get to the next screen.



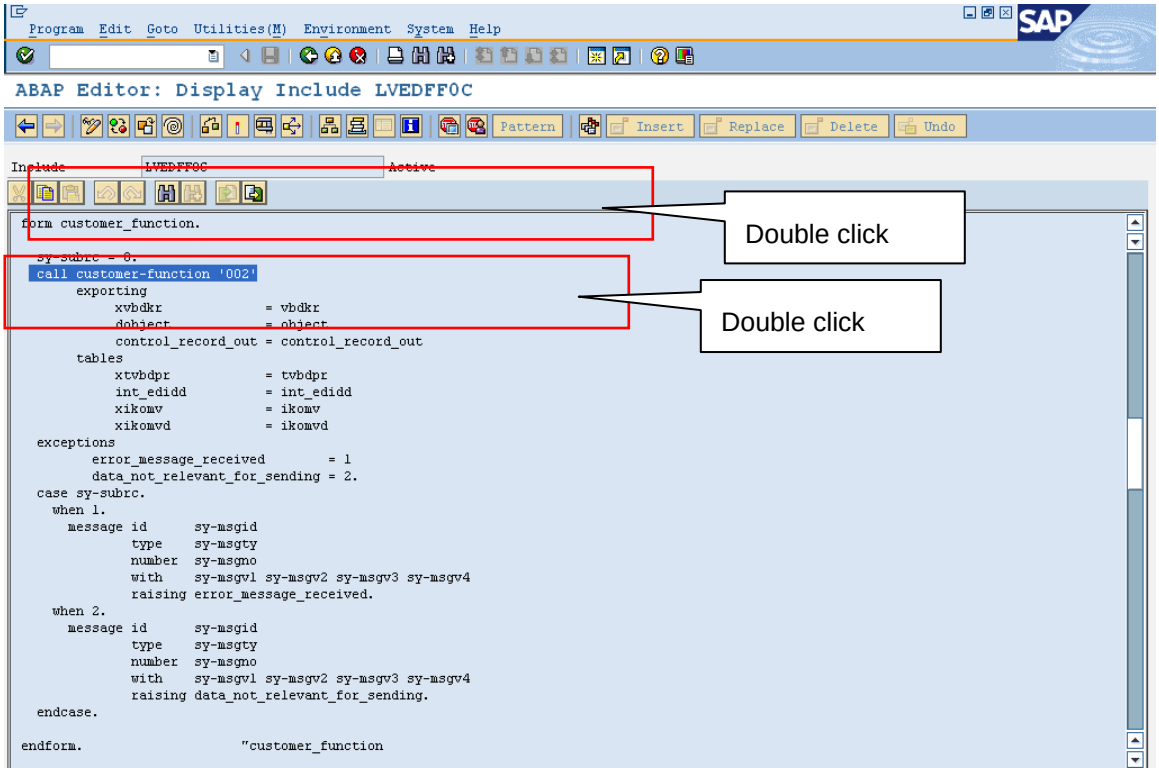
You will find a customer function for control record "customer-function '001'" which is not useful for us. Find the perform which normally come at the end of the Function module having a naming convention starting with "Fill" or "Fuellen". This is shown in the above screen shot.



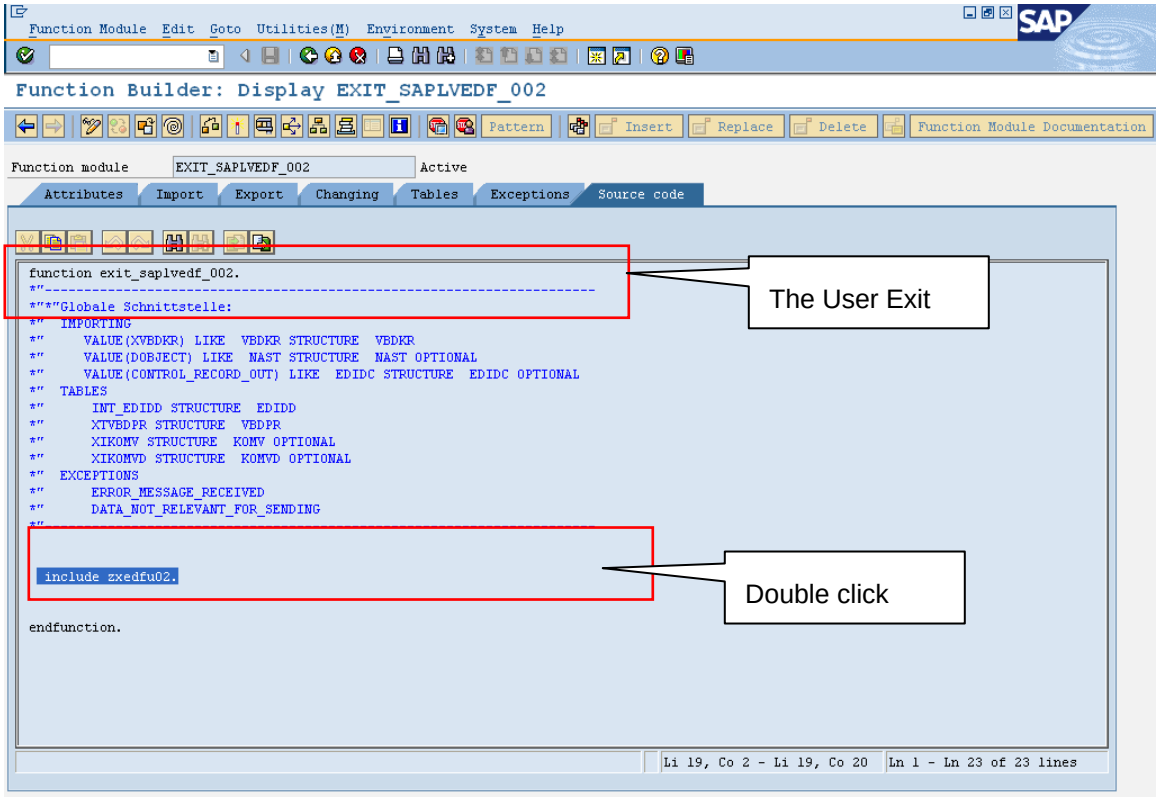
Since we are adding our segment to **e1edk01** find the perform having this string. Here it is **perform fill_e1edk01.** Double click on it.



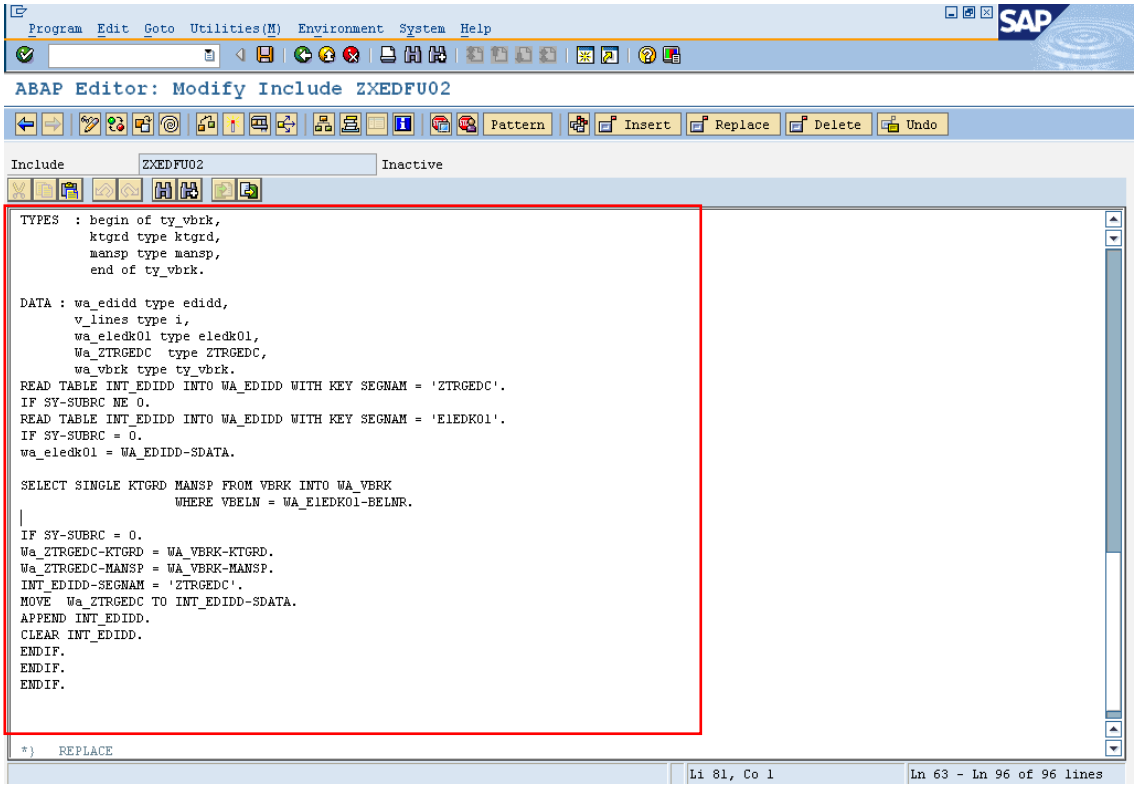
At the end of it you will have perform with name "**perform customer_function**." Double click it. You will get the "**customer-function '002'**" which is the required customer function. Double click it.



We will get the required customer function "**function exit_saplvedf_002**" with an include "**include zxedfu02**". This will lead you to the customer exit where we write the required changes.



We will write the code as shown below.



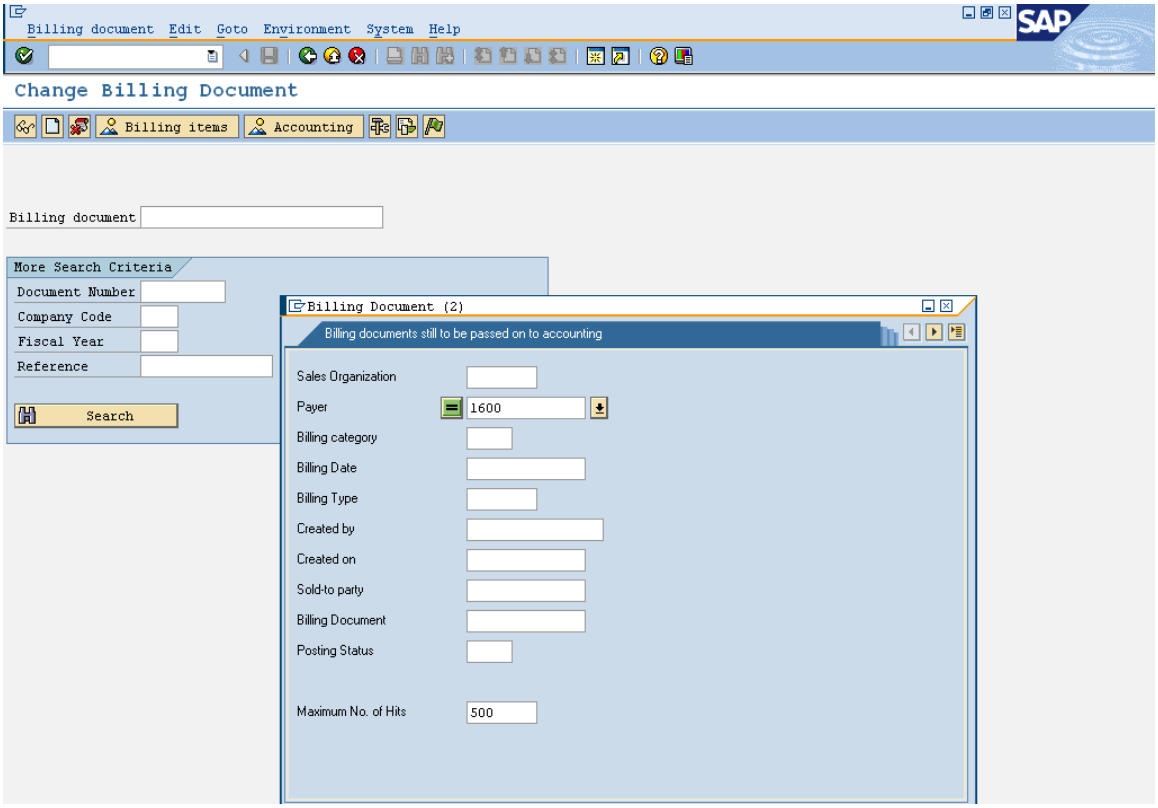
Please find the code below.

```
TYPES  : begin of ty_vbrk,
         ktgrd type ktgrd,
         mansp type mansp,
         end of ty_vbrk.

DATA : wa_edidd type edidd,
       v_lines type i,
       wa_e1edk01 type e1edk01,
       Wa_ZTRGEDC  type ZTRGEDC,
       wa_vbrk type ty_vbrk.
READ TABLE INT_EDIDD INTO WA_EDIDD WITH KEY SEGNAM = 'ZTRGEDC'.
IF SY-SUBRC NE 0.
READ TABLE INT_EDIDD INTO WA_EDIDD WITH KEY SEGNAM = 'E1EDK01'.
IF SY-SUBRC = 0.
wa_e1edk01 = WA_EDIDD-SDATA.

SELECT SINGLE KTGRD MANSP FROM VBRK INTO WA_VBRK
                    WHERE VBELN = WA_E1EDK01-BELNR.

IF SY-SUBRC = 0.
Wa_ZTRGEDC-KTGRD = WA_VBRK-KTGRD.
Wa_ZTRGEDC-MANSP = WA_VBRK-MANSP.
INT_EDIDD-SEGNAM = 'ZTRGEDC'.
MOVE  Wa_ZTRGEDC TO INT_EDIDD-SDATA.
APPEND INT_EDIDD.
CLEAR INT_EDIDD.
ENDIF.
ENDIF.
ENDIF.
```

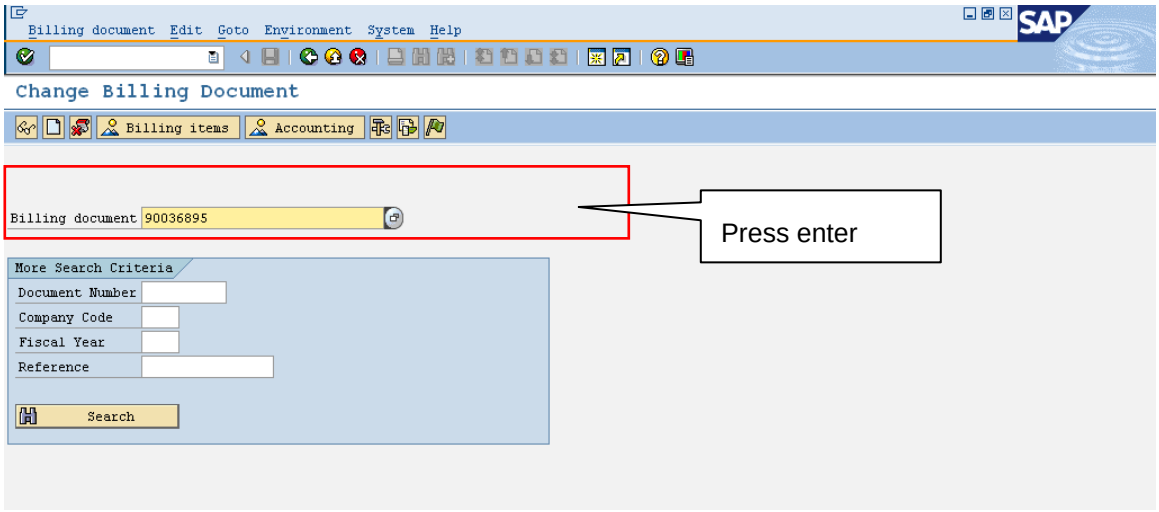
Save and activate it

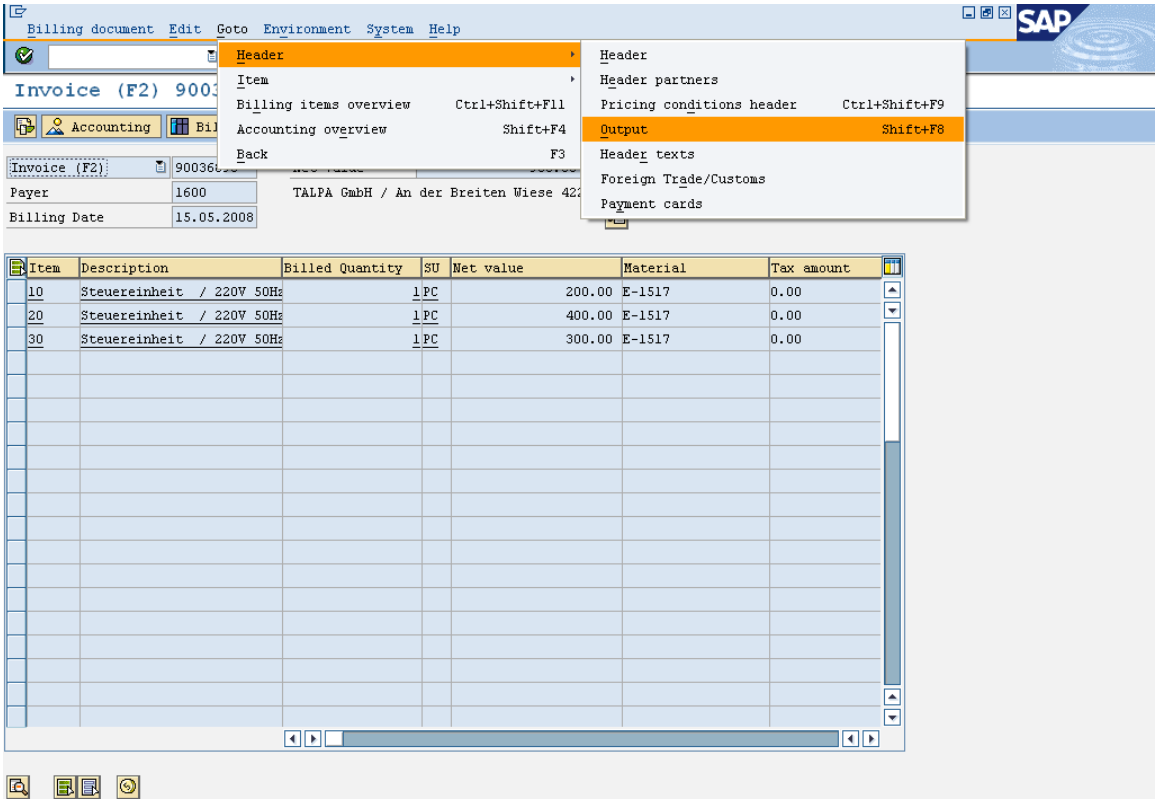
Create a Transaction and Send IDoc

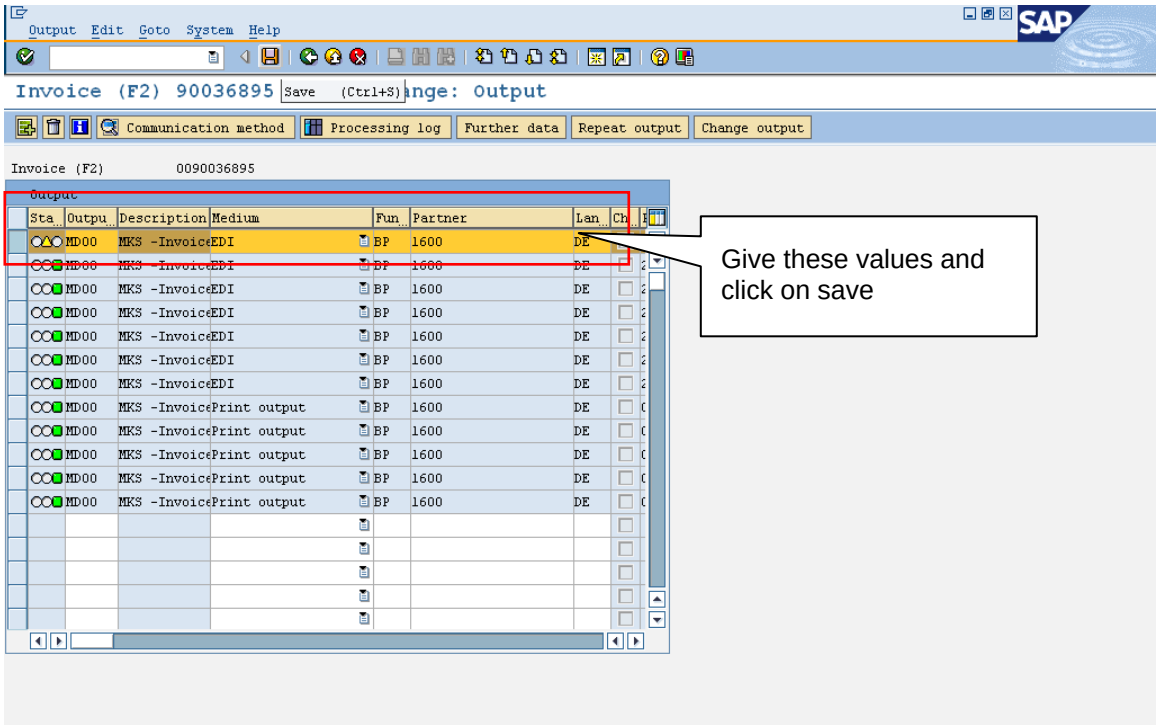
Go to transaction VF02 and search the bill for customer “1600” (we chose it)

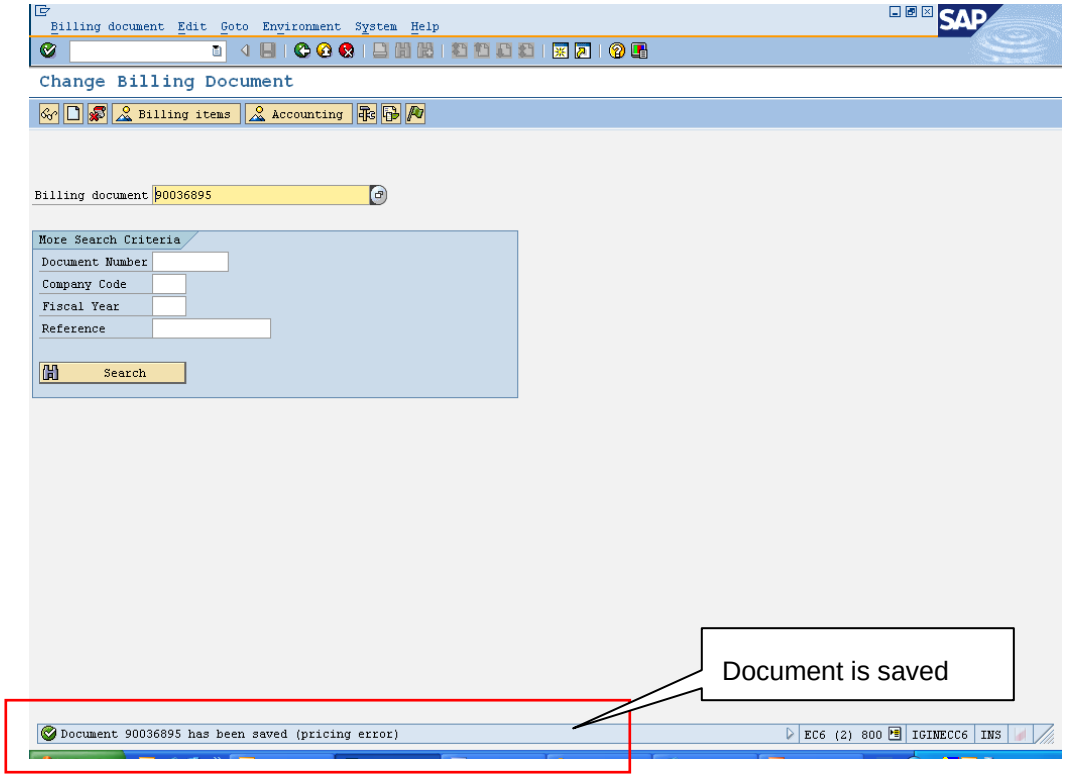


Select a bill as shown and press enter.

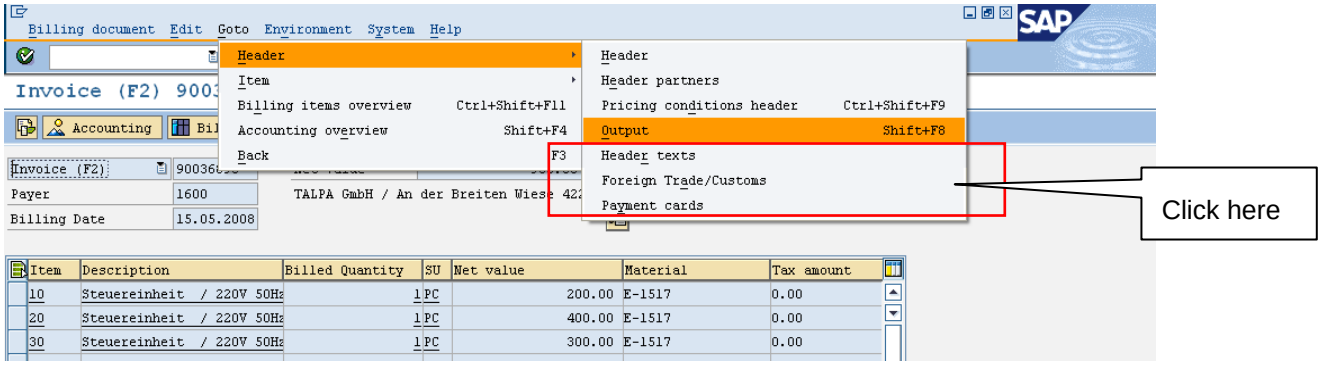


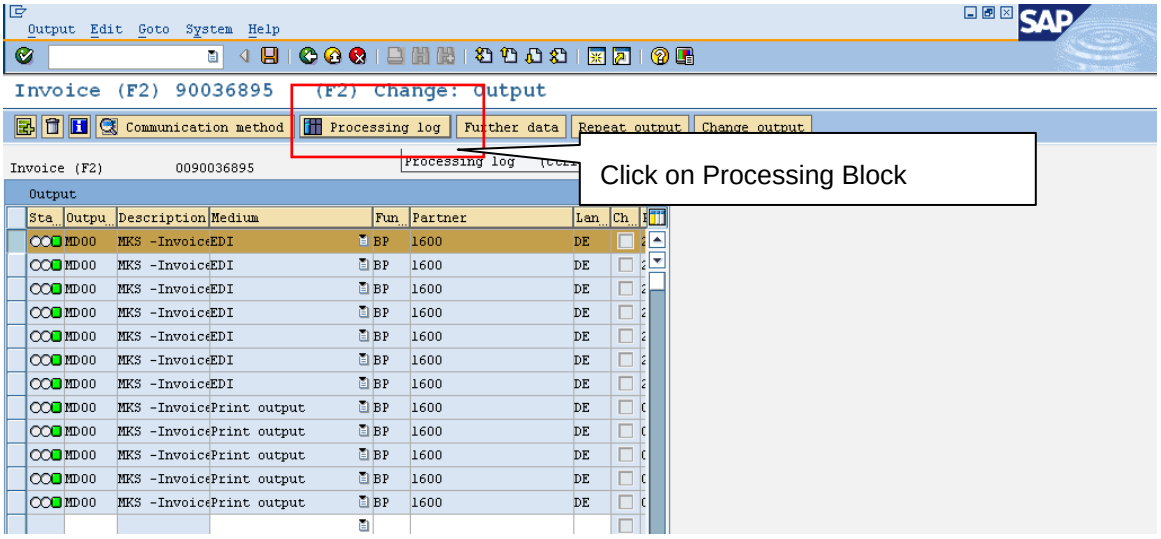

Billing document Edit Goto Environment System Help
Header
Item
Billing items overview Ctrl+Shift+F11
Accounting overview Shift+F4
Back F3
Header
Header partners
Pricing conditions header Ctrl+Shift+F9
Output Shift+F8
Header texts
Foreign Trade/Customs
Payment cards
Accounting
Payer 1600
Billing Date 15.05.2008
Item Description Billed Quantity SU Net value Material Tax amount
10 Steuereinheit / 220V 50Hz 1 PC 200.00 E-1517 0.00
20 Steuereinheit / 220V 50Hz 1 PC 400.00 E-1517 0.00
30 Steuereinheit / 220V 50Hz 1 PC 300.00 E-1517 0.00

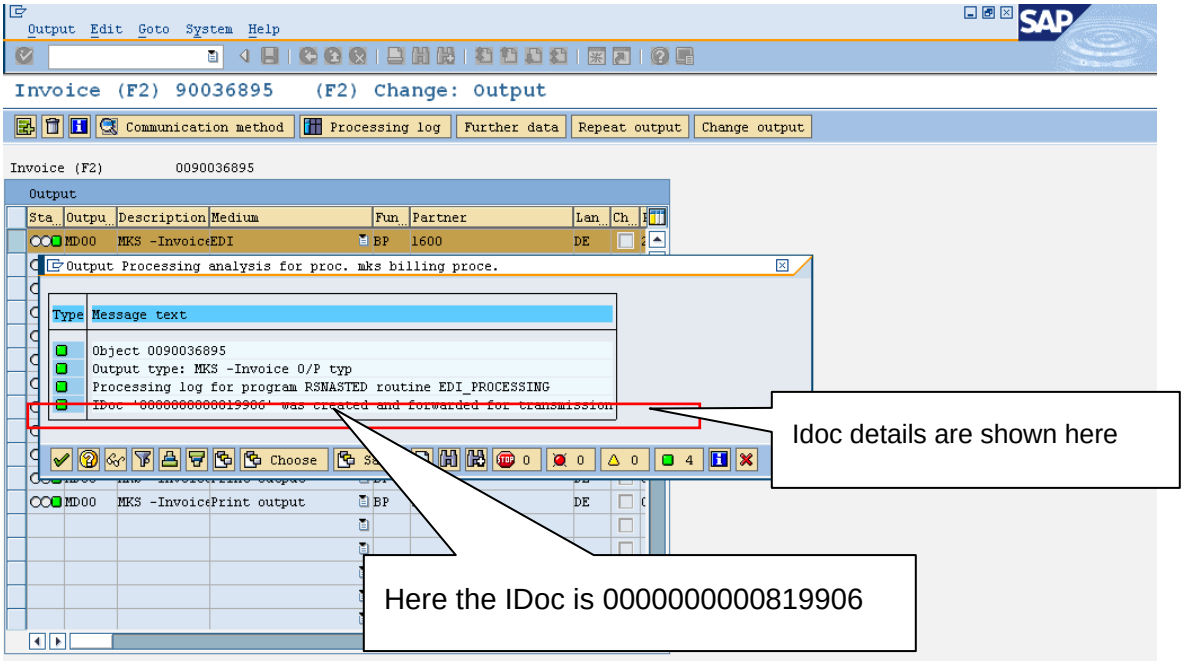

Output Edit Goto System Help
Invoice (F2) 90036895 Save (Ctrl+S) nge: Output
Communication method
Processing log
Further data
Repeat output
Change output
Invoice (F2) 0090036895
Output
Sta Outpu Description Medium Fun Partner Lan
MD00 MKS -Invoice EDI BP 1600 DE
MD00 MKS -Invoice Print output BP 1600 DE
Give these values and click on save



Once again go to the same screen to find the IDOC no.

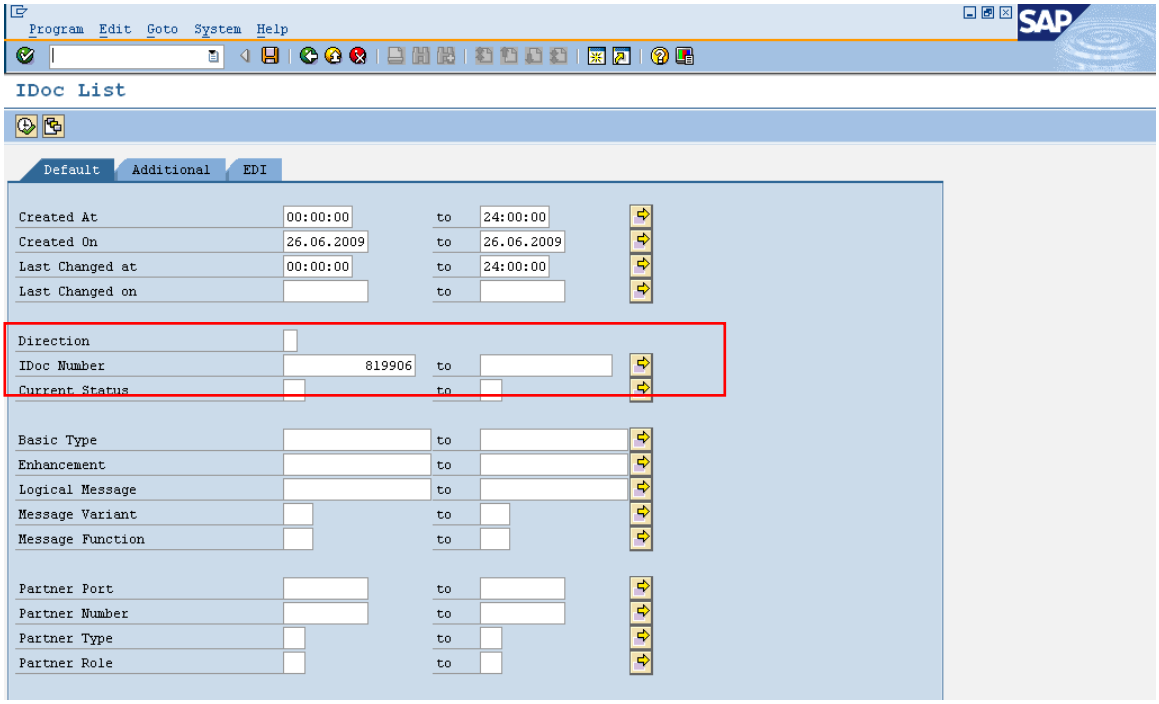


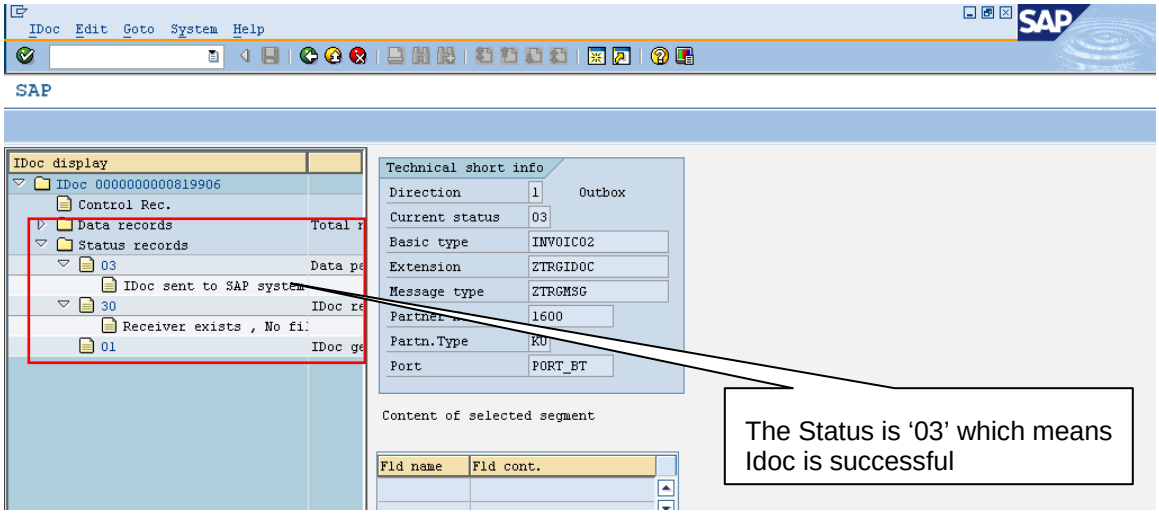


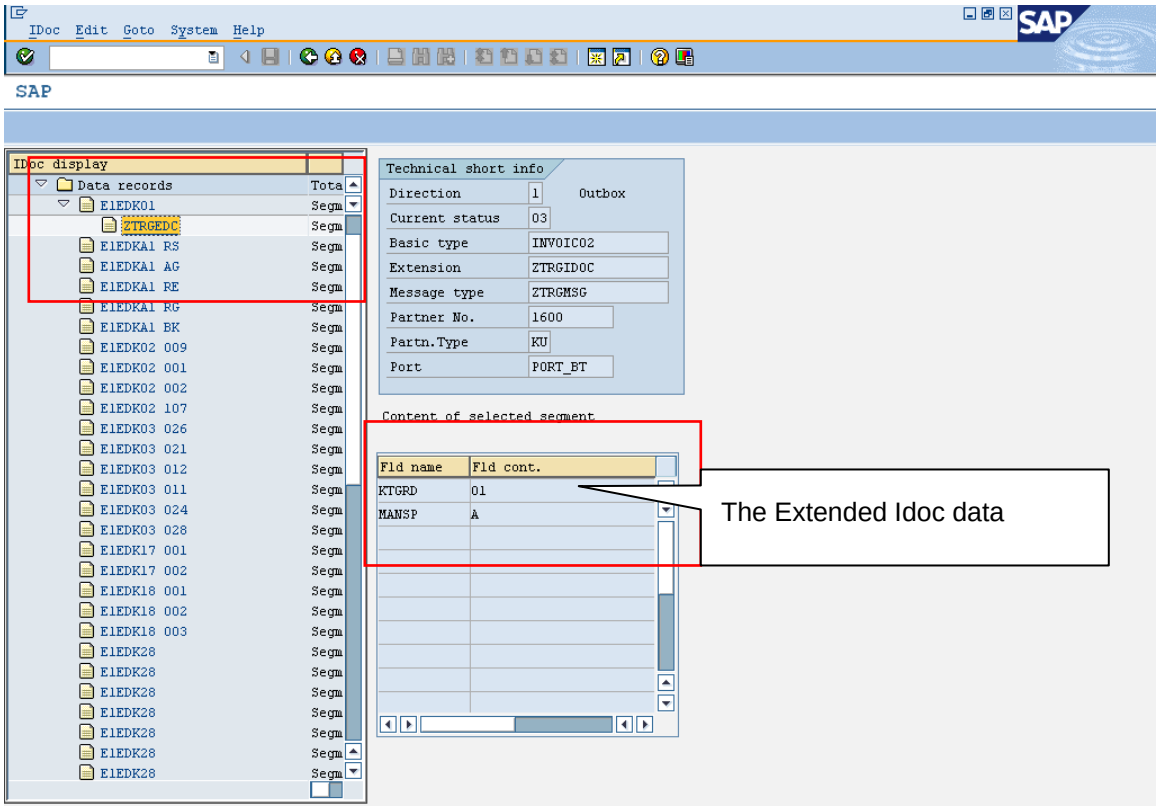


Monitoring IDoc

In WEo2 transaction gives the details as shown and finds the generated IDoc.







So the Extended IDOC is Created and successfully posted from Outbound.

In most cases basic Idoc will be suitable for your applications and if it is lacking with few set of fields then you can extend the Idoc by inserting with the fields you required. This helps the easy usage of the existing standard Idoc with little customization.

Related Content

Extending an IDoc Type

Extension of IDoc types and Processing

Idoc homepage

For more information, visit the ABAP homepage.

Disclaimer and Liability Notice

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

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

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