FM – Function Module

Interviewer may ask you for Step by Step Implementation of FM

Here is simple guide, Let’s revise it …

Function Modules

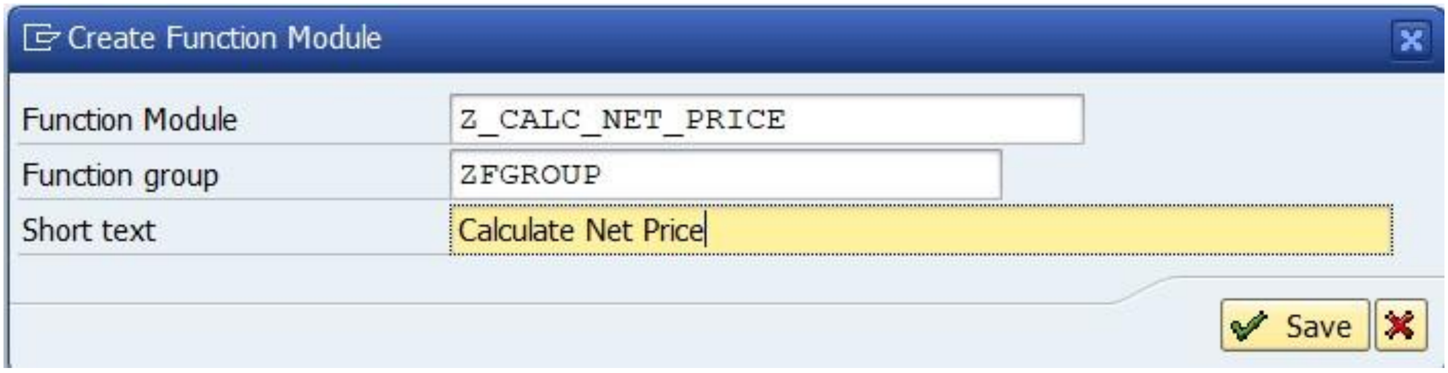
Scenario 1 : Sales Order Processing

Purpose: Create a function module to calculate the net price of an item after applying a tax rate, useful for pricing scenarios in SAP systems.

The function module Z_CALC_NET_PRICE calculates the net price of an item given its gross price and tax rate.

Steps to Create:

- Go to SE37 (Function Builder), enter a function module name like **Z_CALC_NET_PRICE**, and assign it to a function group (create one in SE80 if needed, e.g., **ZFGROUP**).

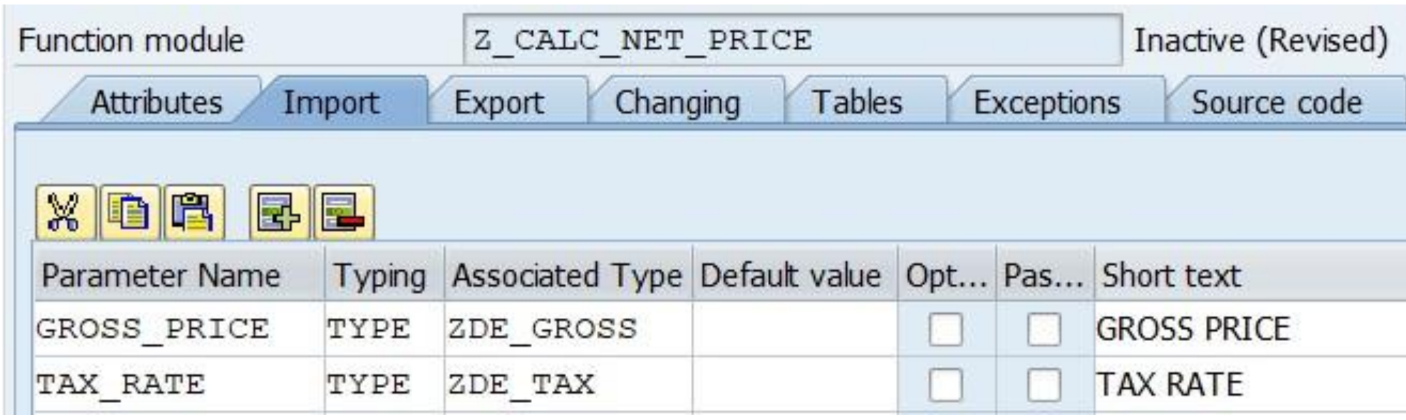


Function Module	Z_CALC_NET_PRICE
Function group	ZFGROUP
Short text	Calculate Net Price

- Define parameters:
  - Importing: GROSS_PRICE, TAX_RATE .
    - **GROSS_PRICE** (type ZDE_PRICE)
    - **TAX_RATE** (type ZDE_TAX)
  - Both Importing parameters of type data elements with data

•

type pack, length 5 , decimals 2

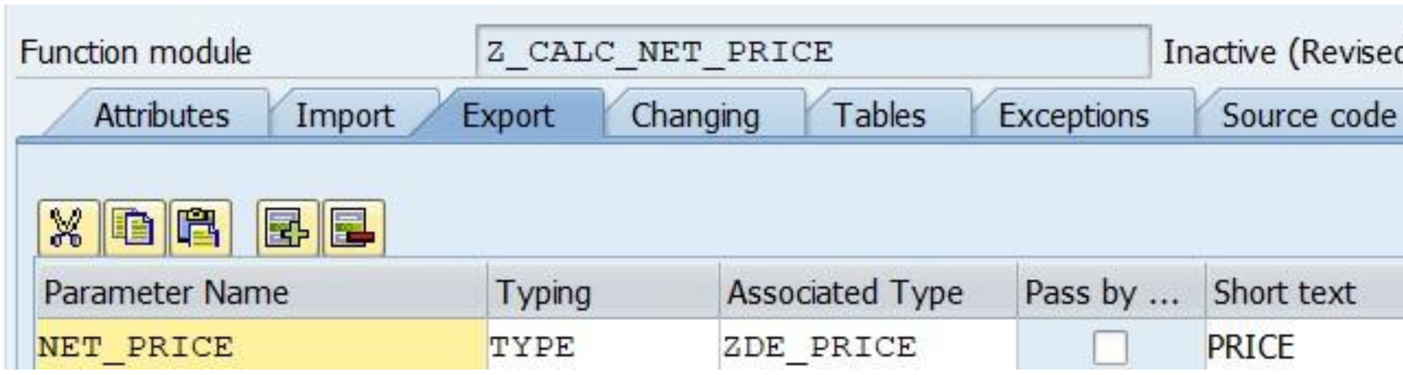


Function module Z_CALC_NET_PRICE Inactive (Revised)

Attributes | Import | Export | Changing | Tables | Exceptions | Source code

Parameter Name	Typing	Associated Type	Default value	Opt...	Pas...	Short text
GROSS_PRICE	TYPE	ZDE_GROSS		☐	☐	GROSS PRICE
TAX_RATE	TYPE	ZDE_TAX		☐	☐	TAX RATE

- **Exporting**
  - NET_PRICE (type ZDE_PRICE)



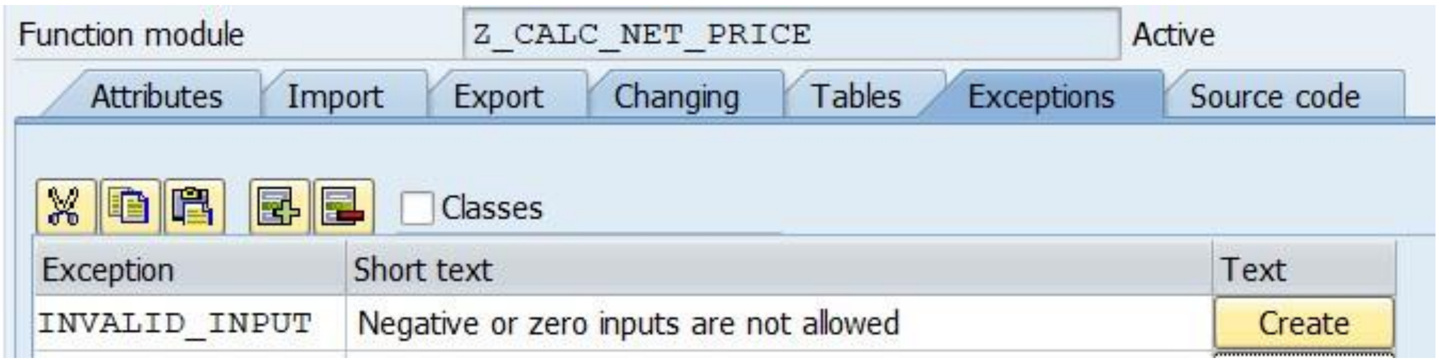
Function module Z_CALC_NET_PRICE Inactive (Revised

Attributes | Import | Export | Changing | Tables | Exceptions | Source code

Parameter Name	Typing	Associated Type	Pass by ...	Short text
NET_PRICE	TYPE	ZDE_PRICE	☐	PRICE

- **Exceptions**
  -

INVALID_INPUT: Raised if GROSS_PRICE is less than or equal to zero or TAX_RATE is negative.

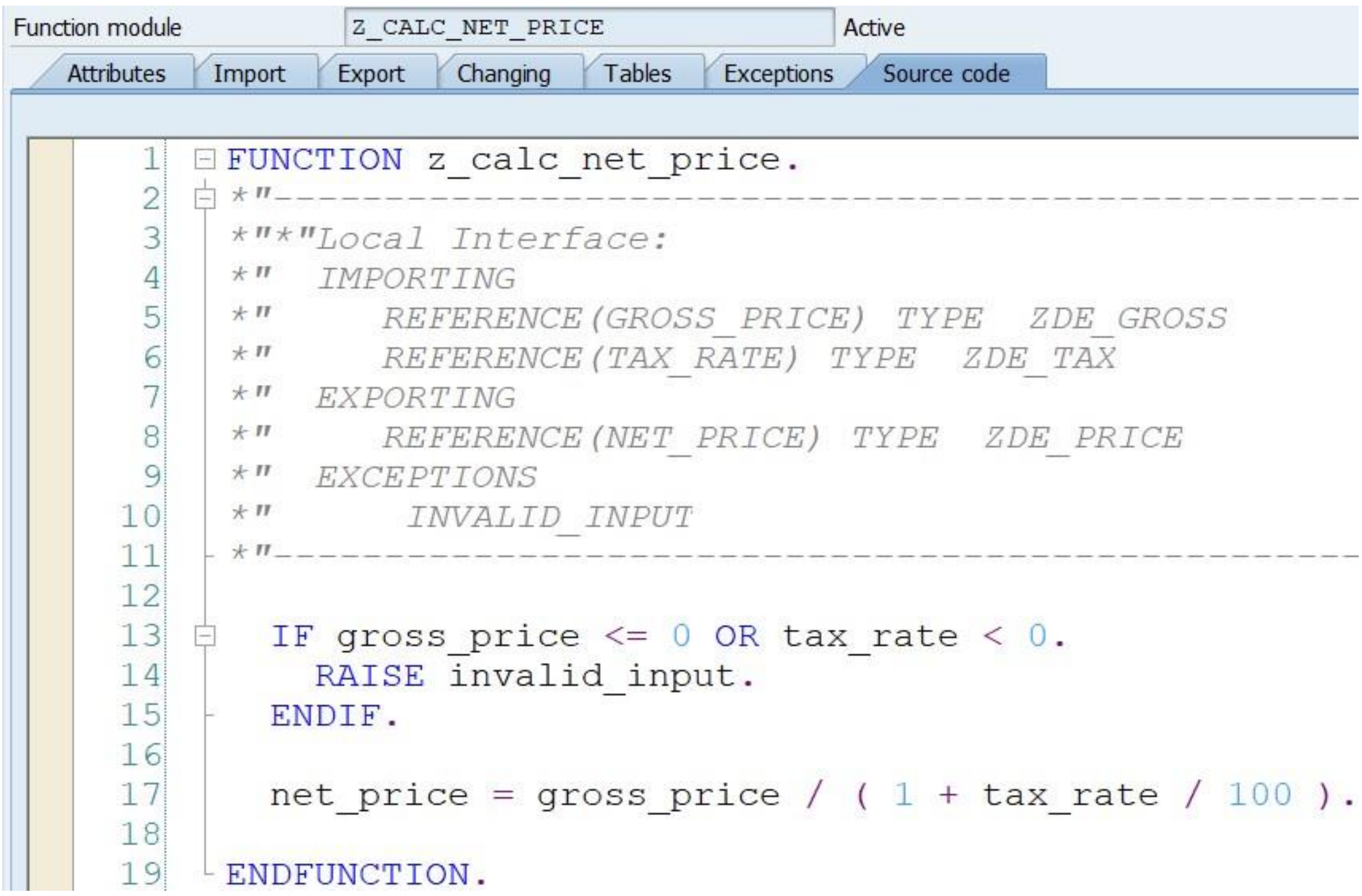

Function module Z_CALC_NET_PRICE Active

Exception	Short text	Text
INVALID_INPUT	Negative or zero inputs are not allowed	Create

- Exceptions tab allows you to define non-class-based exceptions that a function module can raise to signal specific error conditions back to the calling program.

  You list these exceptions and provide a short text description for each, which are then triggered using the **RAISE or MESSAGE RAISING** statements within the function module
- Now add the below code under the source code tab of the function modules

•



Z_CALC_NET_PRICE

```
FUNCTION z_calc_net_price.
*"----------------------------------------------------------------------
*"*"Local Interface:
*"  IMPORTING
*"     REFERENCE(GROSS_PRICE) TYPE  ZDE_GROSS
*"     REFERENCE(TAX_RATE) TYPE  ZDE_TAX
*"  EXPORTING
*"     REFERENCE(NET_PRICE) TYPE  ZDE_PRICE
*"  EXCEPTIONS
*"      INVALID_INPUT
*"----------------------------------------------------------------------

  IF gross_price <= 0 OR tax_rate < 0.
    RAISE invalid_input.
  ENDIF.

  net_price = gross_price / ( 1 + tax_rate / 100 ).

ENDFUNCTION.
```

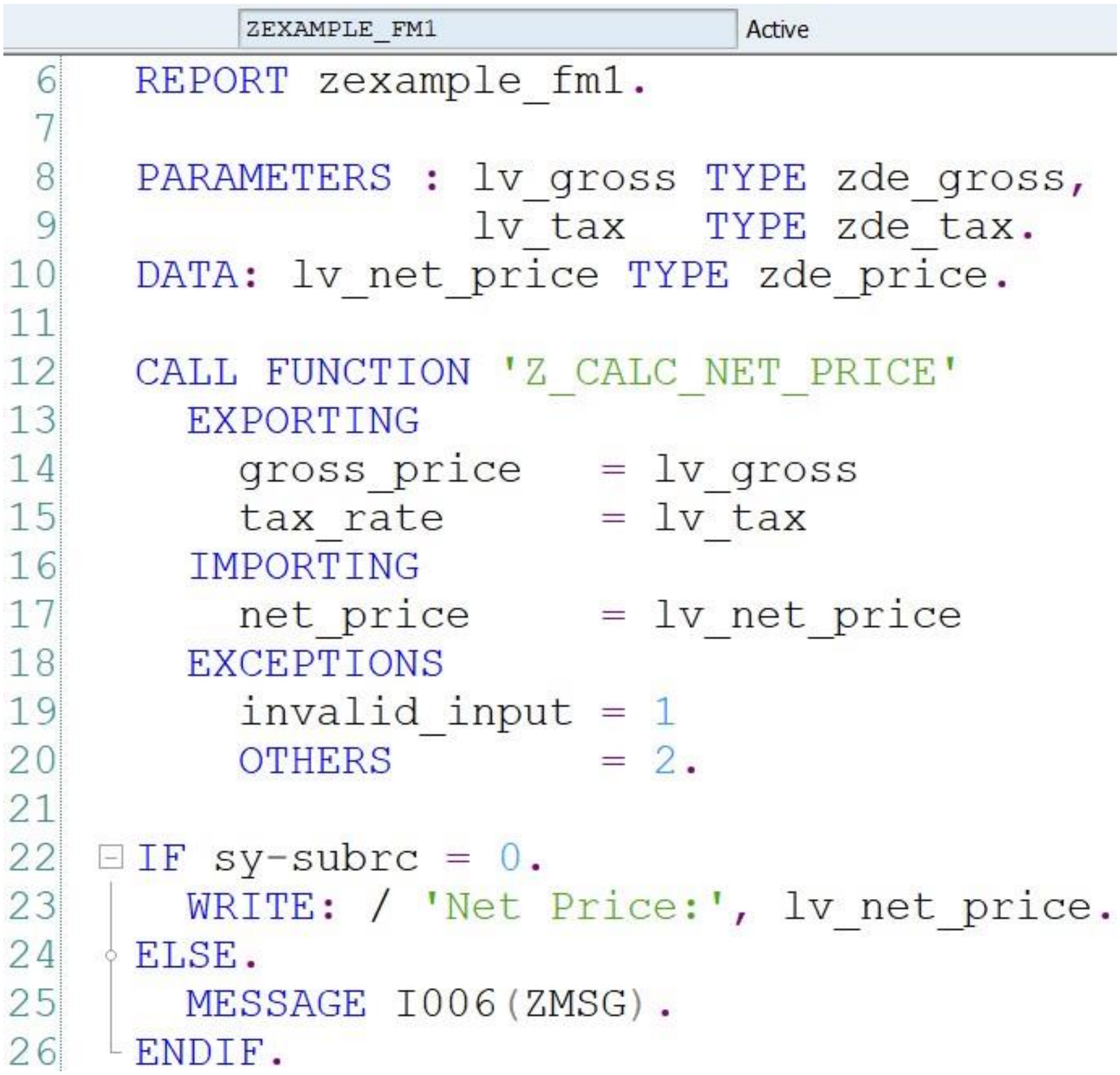
•

If either condition is true, the **INVALID_INPUT** exception is raised, which stops execution and returns control to the calling program with **sy-subrc equals to 1** the net price is calculated using the above formula Now we can create a report program & make use of this function module

•

•

- Create a report program (ZEXAMPLE_FM1) using SE38 (ABAP Editor)



ZEXAMPLE_FM1 Active

```
REPORT zexample_fm1.

PARAMETERS : lv_gross TYPE zde_gross,
             lv_tax   TYPE zde_tax.
DATA: lv_net_price TYPE zde_price.

CALL FUNCTION 'Z_CALC_NET_PRICE'
  EXPORTING
    gross_price   = lv_gross
    tax_rate      = lv_tax
  IMPORTING
    net_price     = lv_net_price
  EXCEPTIONS
    invalid_input = 1
    OTHERS        = 2.

IF sy-subrc = 0.
  WRITE: / 'Net Price:', lv_net_price.
ELSE.
  MESSAGE I006(ZMSG).
ENDIF.
```

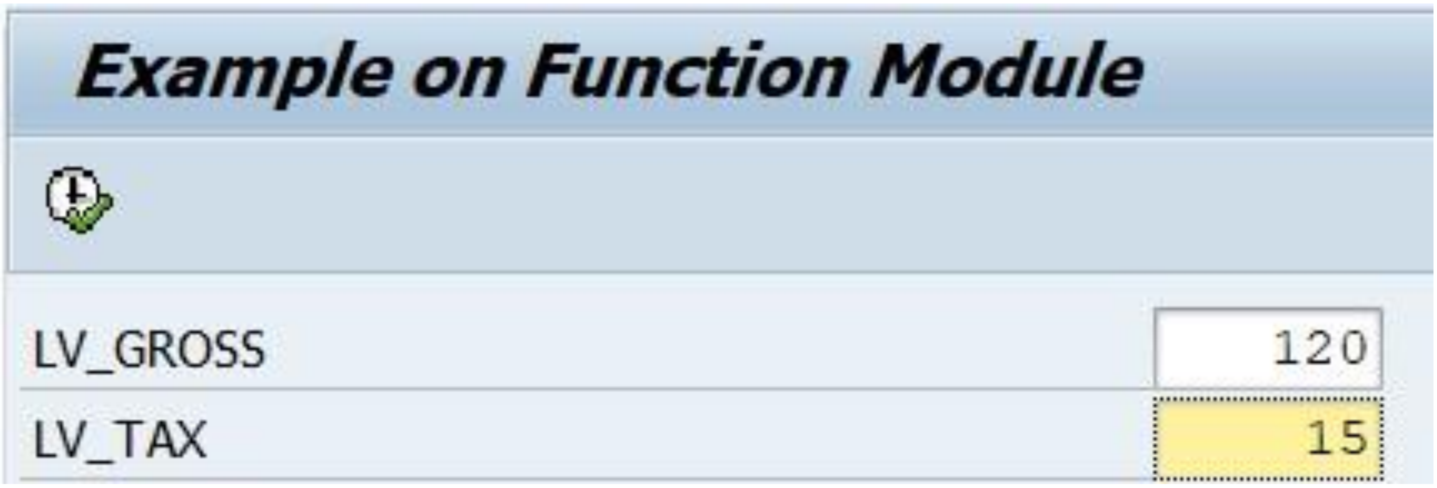
- This program takes user inputs for **gross price** and **tax rate**, uses

-

  a custom function module to calculate net price, and then displays the result or an error message.

  The function module is expected to handle the calculation logic and input validation.

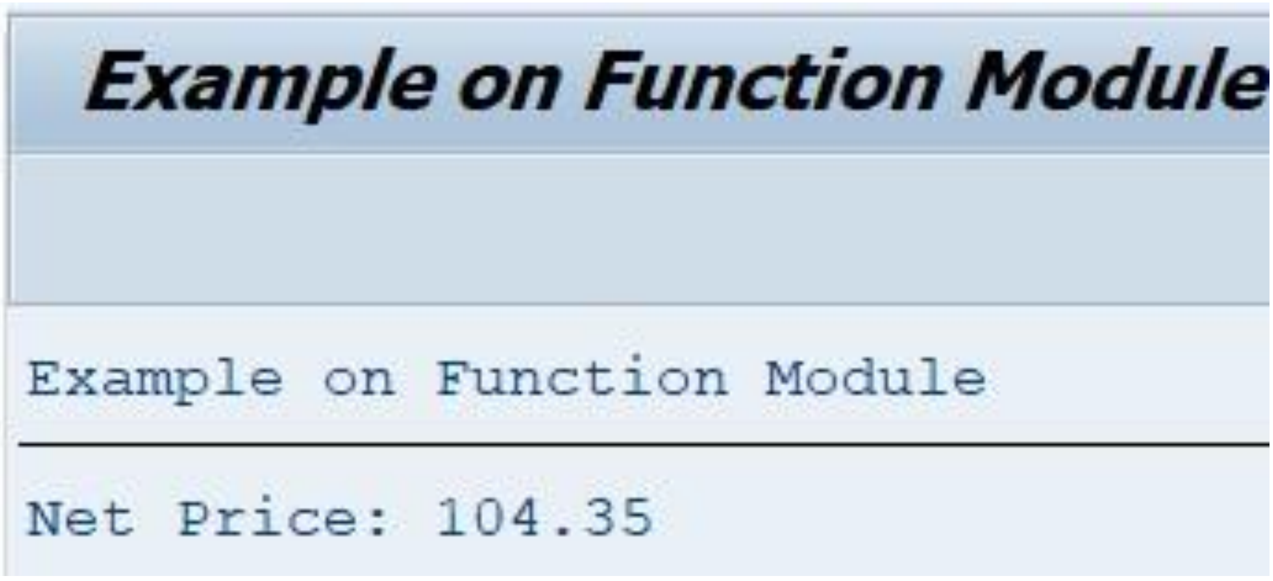
-



Example on Function Module

LV_GROSS	120
LV_TAX	15

- Provide the parameter values - GROSS, Tax
- Then execute & it displays the output of net price



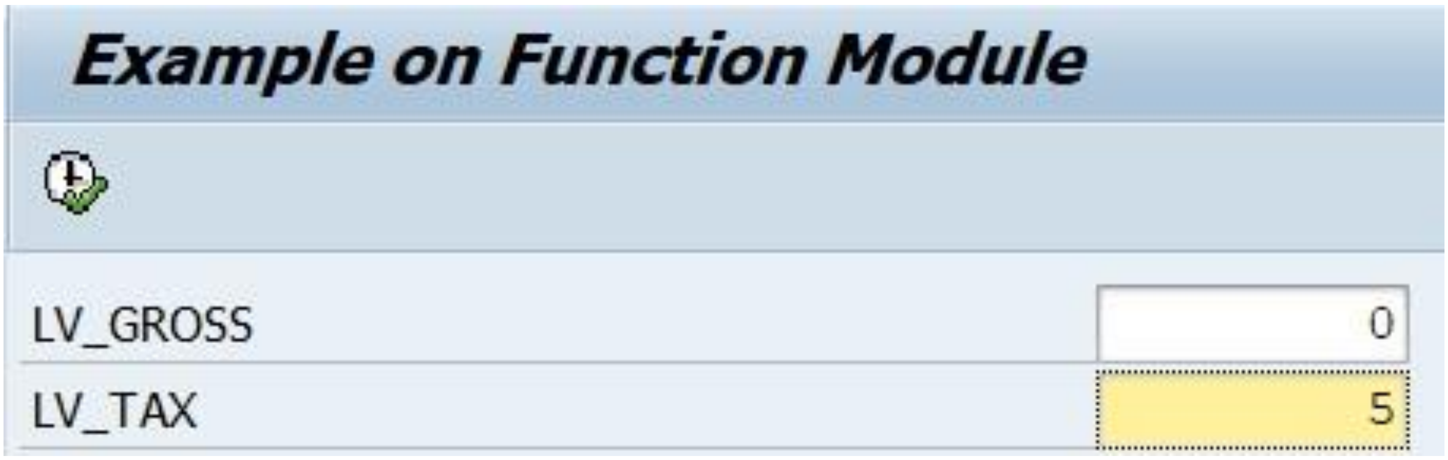
```
Example on Function Module

Net Price: 104.35
```

- If we provide parameter values as zero or negative

Upon Activation & Execute the report program values

- it raises a message with **MESSAGE I006(ZMSG)** to notify there was an issue (likely invalid input or other errors).

"MESSAGE I006(ZMSG)" is a SAP ABAP statement that displays an information message with the ID 006 from the message class ZMSG, indicating a user-defined message

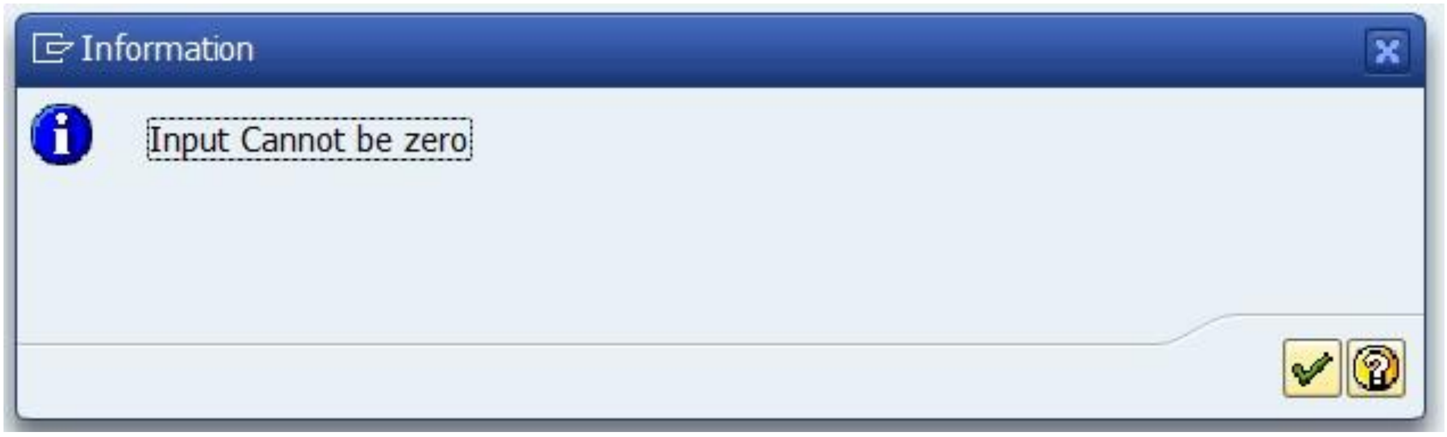

Example on Function Module

LV_GROSS	0
LV_TAX	5

-

Here we provide zero value to the parameter (LV_GROSS), so it raises an exception

- Which results as (sy-subrc = 1): Triggered for invalid inputs.


Information

Input Cannot be zero

-

So it gives the error message in the information dialog box with error message as - Input Cannot be zero

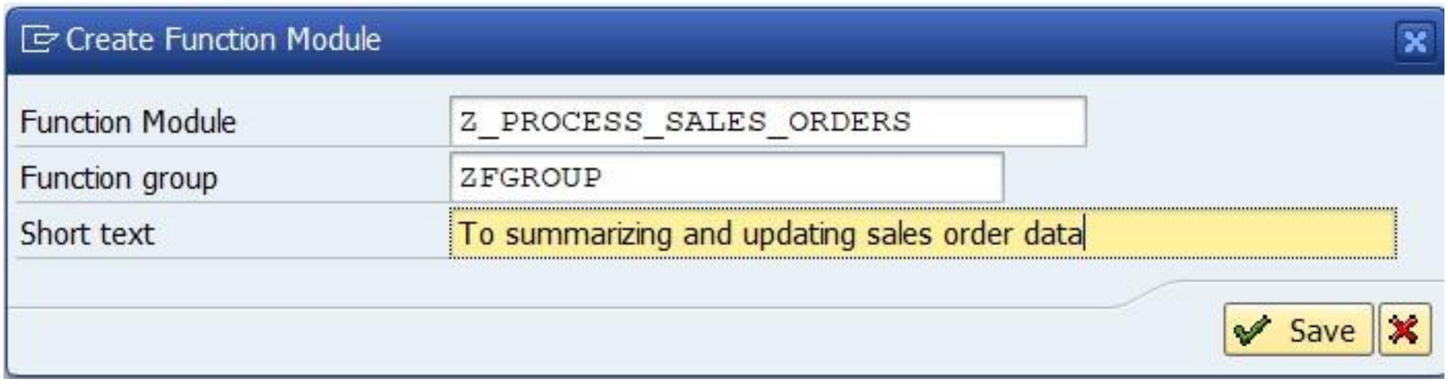
-
- If any exception raised i.e (sy-subrc > 1), then it leads to run-time error as dump

Scenario 2 : Sales Order Processing

Purpose : The function module will process a list of sales orders, calculate their total value, update quantities, and handle errors, making it a good fit for demonstrating all interface components.

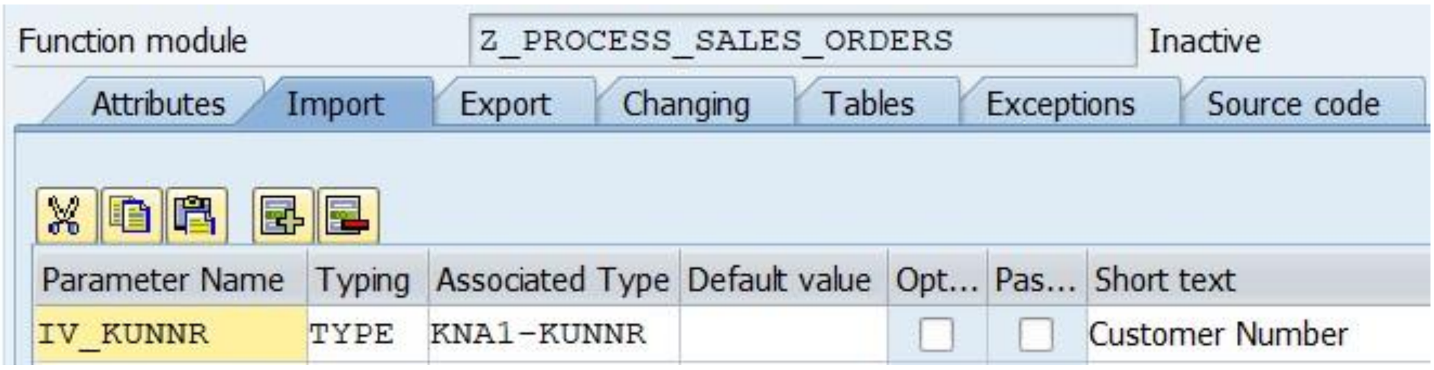
Steps to Create:

- Go to SE37 (Function Builder), enter a function module name like **Z_PROCESS_SALES_ORDERS**, and assign it to a function group (create one in SE80 if needed, e.g., **ZFGROUP**).



Function Module	Z_PROCESS_SALES_ORDERS
Function group	ZFGROUP
Short text	To summarizing and updating sales order data

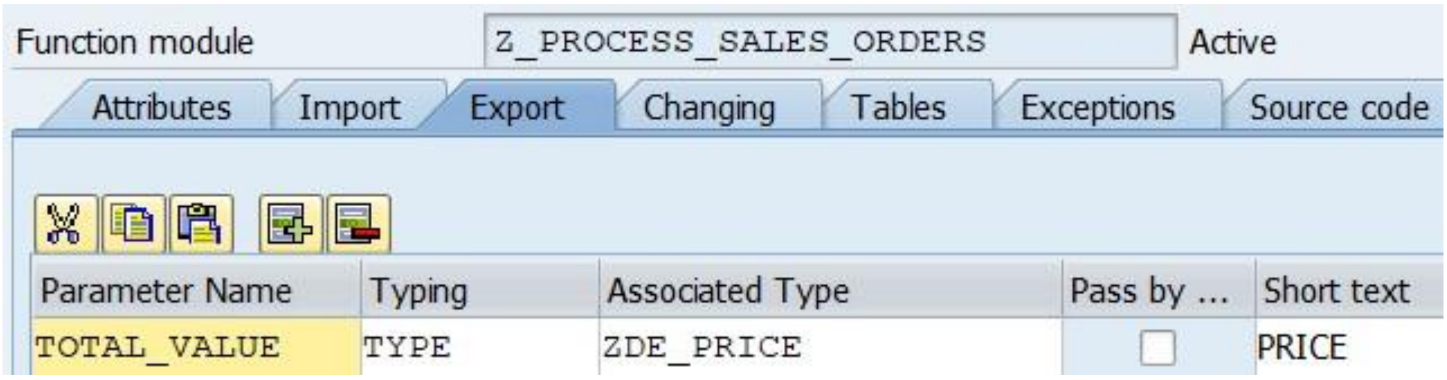
- **Importing Parameters**
  - IV_KUNNR (type KNA1-KUNNR): Customer number to filter sales orders.



Parameter Name	Typing	Associated Type	Default value	Opt...	Pas...	Short text
IV_KUNNR	TYPE	KNA1-KUNNR		☐	☐	Customer Number

- **Exporting**:
  - TOTAL_VALUE (type ZDE_PRICE, assuming

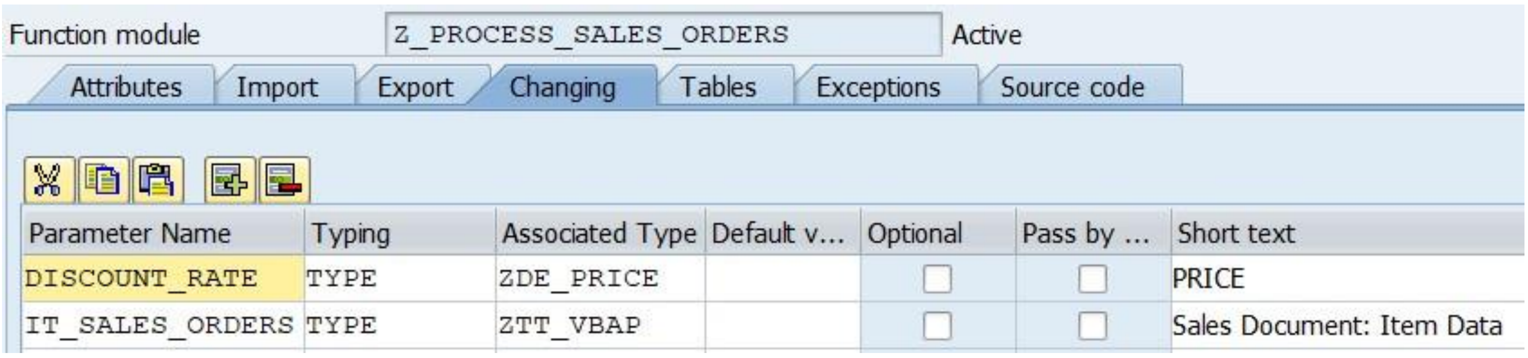
ZDE_PRICE is a data element of type DEC LENGTH 5 DECIMALS 2): Total value of all sales orders.



Function module: Z_PROCESS_SALES_ORDERS — Active

Parameter Name	Typing	Associated Type	Pass by ...	Short text
TOTAL_VALUE	TYPE	ZDE_PRICE	☐	PRICE

- **Changing**:
  - DISCOUNT_RATE (type ZDE_PRICE): Discount rate to apply, which the function module can modify.



Function module: Z_PROCESS_SALES_ORDERS — Active

Parameter Name	Typing	Associated Type	Default v...	Optional	Pass by ...	Short text
DISCOUNT_RATE	TYPE	ZDE_PRICE		☐	☐	PRICE
IT_SALES_ORDERS	TYPE	ZTT_VBAP		☐	☐	Sales Document: Item Data

- **Tables**:
  - IT_SALES_ORDERS (type VBAP): Table of sales order items to process.
  - But In SAP ABAP Latest version, If you declare an internal table under tab **TABLES**, it gives error as **Parameters TABLES are obsolete**
- It indicates that you should replace the **old TABLES tab** with the modern **CHANGING tab** for passing

internal tables to a function module • So for that, i declared a parameter name IT_SALES_ORDER under CHANGING tab only

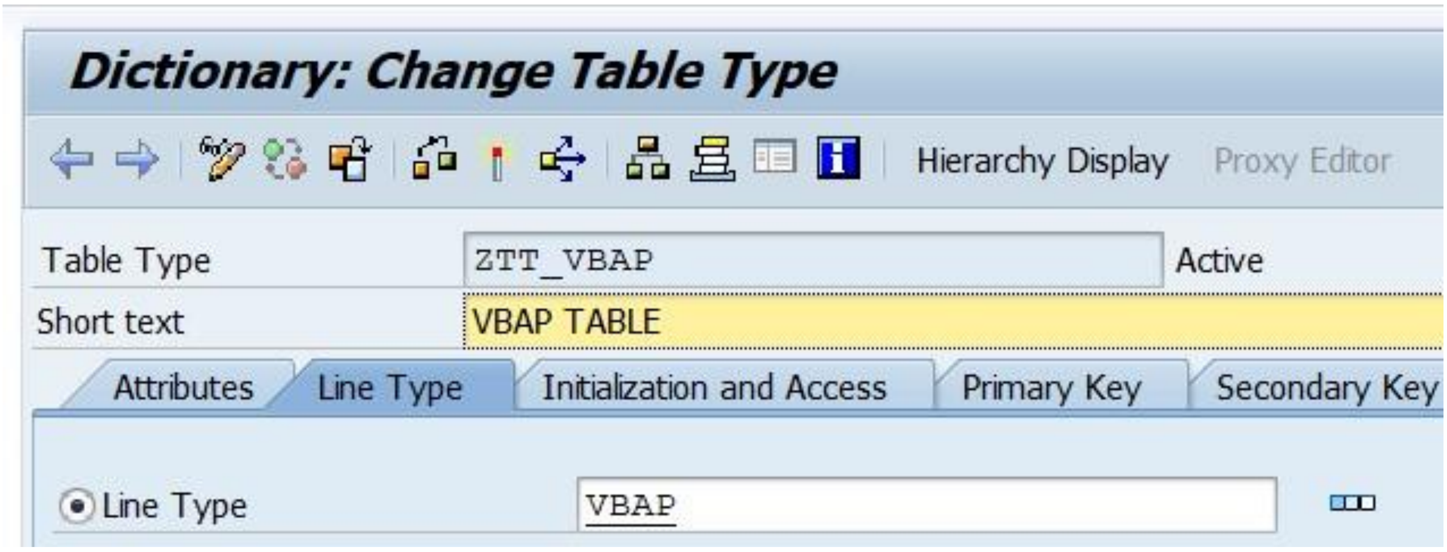

Dictionary: Change Table Type

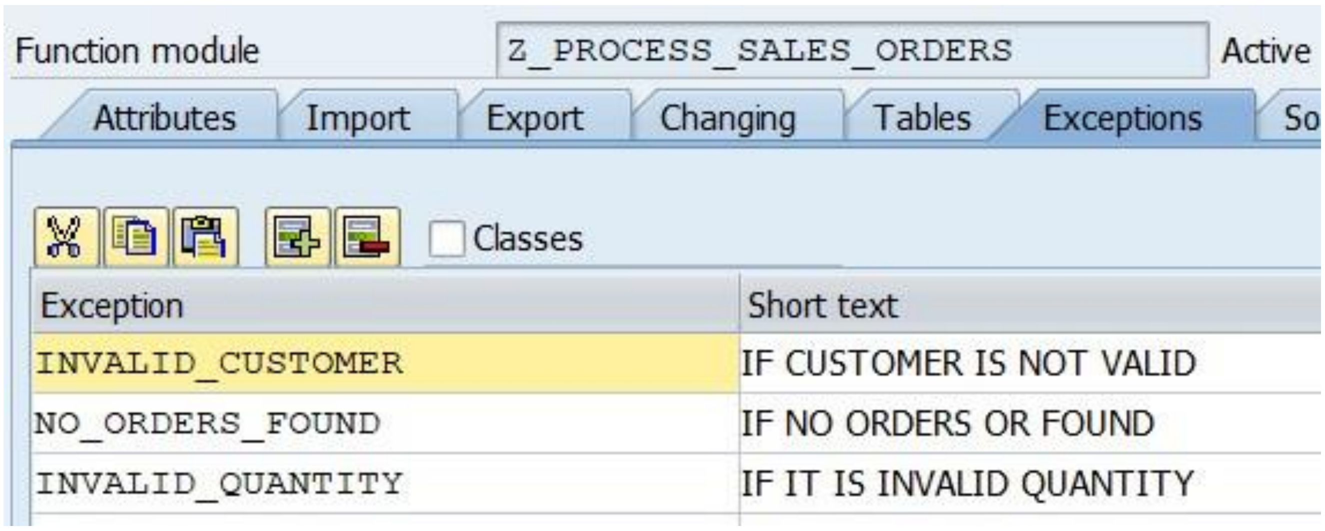
Table Type ZTT_VBAP Active
Short text VBAP TABLE

● Line Type VBAP

•

To declare that internal table , firstly Create a table type in se11 of type **VBAP**, before adding into CHANGING tab **Exceptions**:

- In the exceptions tab, we have to declare 3 exceptions as
- Invalid customer, zero order, invalid quantity
  - INVALID_CUSTOMER: Raised if the customer number is invalid.
  - NO_ORDERS_FOUND: Raised if no sales orders exist for the customer.
  - INVALID_QUANTITY: Raised if any order has an invalid quantity.

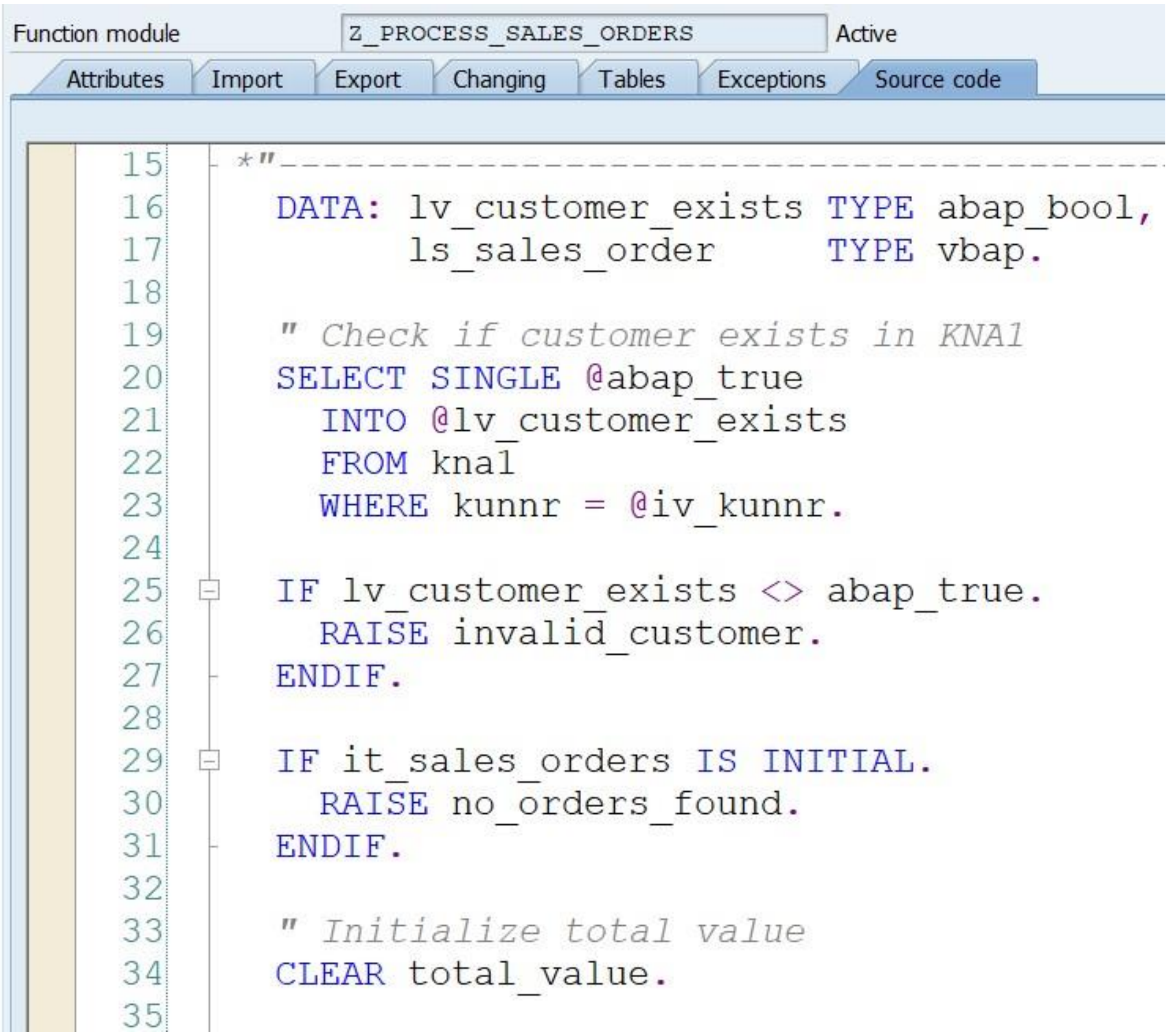

Function module Z_PROCESS_SALES_ORDERS Active

Attributes | Import | Export | Changing | Tables | Exceptions | So

☐ Classes

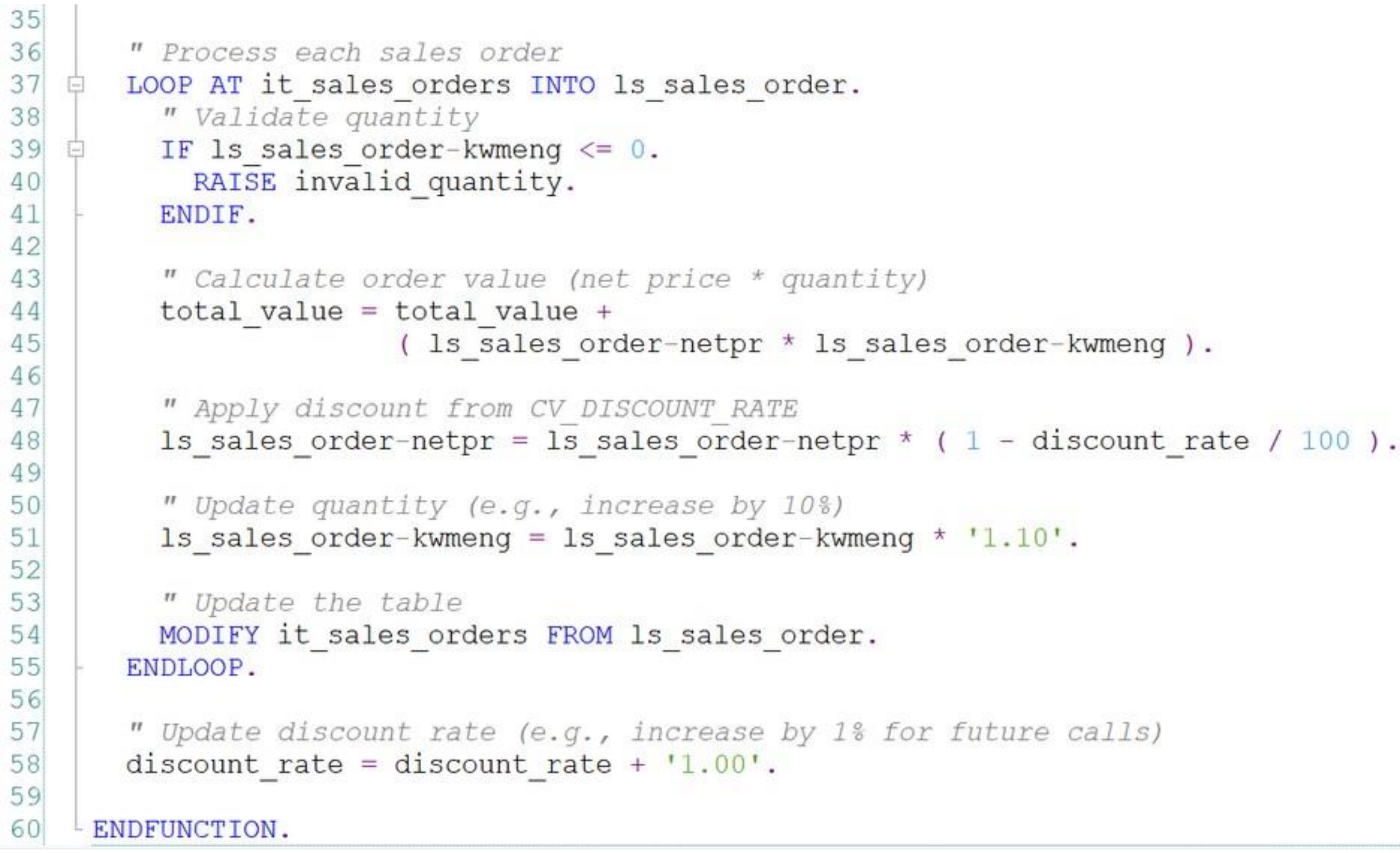
Exception	Short text
INVALID_CUSTOMER	IF CUSTOMER IS NOT VALID
NO_ORDERS_FOUND	IF NO ORDERS OR FOUND
INVALID_QUANTITY	IF IT IS INVALID QUANTITY

- And, finally add the below code into the source code of function module



```
*"-----------------------------------------
  DATA: lv_customer_exists TYPE abap_bool,
        ls_sales_order     TYPE vbap.

  " Check if customer exists in KNA1
  SELECT SINGLE @abap_true
    INTO @lv_customer_exists
    FROM kna1
    WHERE kunnr = @iv_kunnr.

  IF lv_customer_exists <> abap_true.
    RAISE invalid_customer.
  ENDIF.

  IF it_sales_orders IS INITIAL.
    RAISE no_orders_found.
  ENDIF.

  " Initialize total value
  CLEAR total_value.

```

The source code of the function module **Z_PROCESS_SALES_ORDERS,** which processes a list of sales orders, calculates their total value, applies discounts, updates quantities, and adjusts the discount rate for future calls.



```

    " Process each sales order
    LOOP AT it_sales_orders INTO ls_sales_order.
      " Validate quantity
      IF ls_sales_order-kwmeng <= 0.
        RAISE invalid_quantity.
      ENDIF.

      " Calculate order value (net price * quantity)
      total_value = total_value +
                    ( ls_sales_order-netpr * ls_sales_order-kwmeng ).

      " Apply discount from CV_DISCOUNT_RATE
      ls_sales_order-netpr = ls_sales_order-netpr * ( 1 - discount_rate / 100 ).

      " Update quantity (e.g., increase by 10%)
      ls_sales_order-kwmeng = ls_sales_order-kwmeng * '1.10'.

      " Update the table
      MODIFY it_sales_orders FROM ls_sales_order.
    ENDLOOP.

    " Update discount rate (e.g., increase by 1% for future calls)
    discount_rate = discount_rate + '1.00'.

ENDFUNCTION.
```

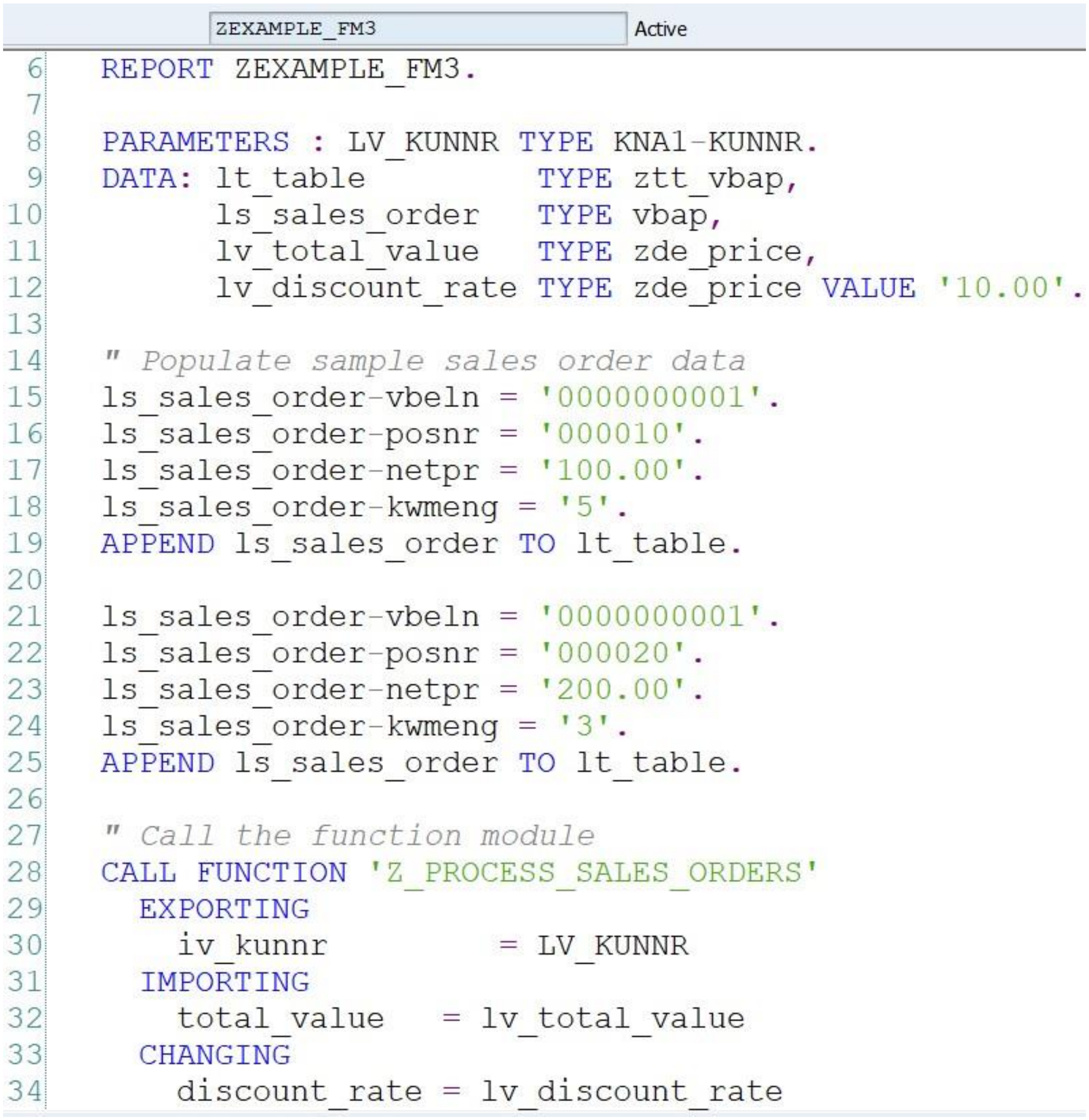
•

The code iterates over all sales orders to:

- Validate each quantity is positive.
- Calculate and accumulate the total order value.
- Apply a discount on net price.
- Increase the quantity by 10%.
- Update the internal table with modified data.
- Increase the discount rate for potential future recalculations.

It combines validation, arithmetic operations, and table update statements inside a loop for processing sales order data.

- Now create a report program & make use of the function module, to process the sales order of a customer



```
REPORT ZEXAMPLE_FM3.

PARAMETERS : LV_KUNNR TYPE KNA1-KUNNR.
DATA: lt_table          TYPE ztt_vbap,
      ls_sales_order    TYPE vbap,
      lv_total_value    TYPE zde_price,
      lv_discount_rate  TYPE zde_price VALUE '10.00'.

" Populate sample sales order data
ls_sales_order-vbeln = '0000000001'.
ls_sales_order-posnr = '000010'.
ls_sales_order-netpr = '100.00'.
ls_sales_order-kwmeng = '5'.
APPEND ls_sales_order TO lt_table.

ls_sales_order-vbeln = '0000000001'.
ls_sales_order-posnr = '000020'.
ls_sales_order-netpr = '200.00'.
ls_sales_order-kwmeng = '3'.
APPEND ls_sales_order TO lt_table.

" Call the function module
CALL FUNCTION 'Z_PROCESS_SALES_ORDERS'
  EXPORTING
    iv_kunnr          = LV_KUNNR
  IMPORTING
    total_value    = lv_total_value
  CHANGING
    discount_rate = lv_discount_rate
```

- It starts by declaring parameters and data structures, including a customer number **(LV_KUNNR),** a table for sales orders **(LT_TABLE),**
- **LS_SALES_ORDER**: A structure to hold individual sales order details (order number VBELN, item number POSNR, net price NETPR, and quantity KWMENG).

- **LV_TOTAL_VALUE and LV_DISCOUNT_RAT**E: Variables to track the total value of orders and a discount rate, with LV_DISCOUNT_RATE initialized to 10.00.
- The code then populates LT_TABLE with sample sales order data using the LS_SALES_ORDER structure. Two entries are added:
  - **First entry:** Order 0000000001, item 000010, net price 100.00, quantity 5.
  - **Second entry:** Order 0000000001, item 000020, net price 200.00, quantity 3.
- The APPEND statement adds these records to **LT_TABLE,**

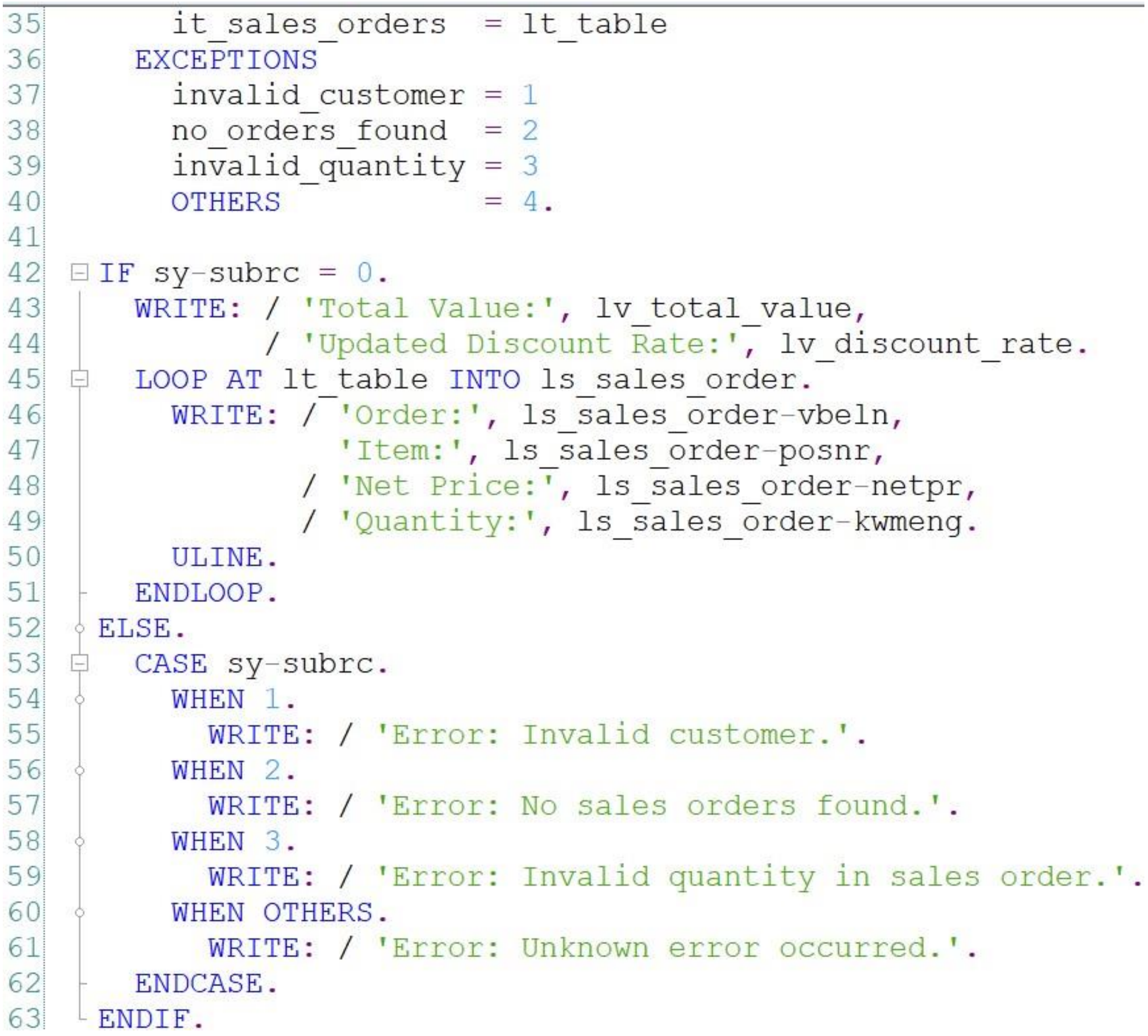
creating a dataset for processing. • the program calls a custom function module **Z_PROCESS_SALES_ORDERS**. This module likely performs business logic on the sales orders, such as calculations or validations. The call passes: **IV_KUNNR:** The customer number from LV_KUNNR.

- IMPORTING parameters: TOTAL_VALUE (set to
- LV_TOTAL_VALUE) and DISCOUNT_RATE (set to

LV_DISCOUNT_RATE).

  - The function returns the processed data, presumably updating LT_TABLE or the imported variables.

- And add below code continuation



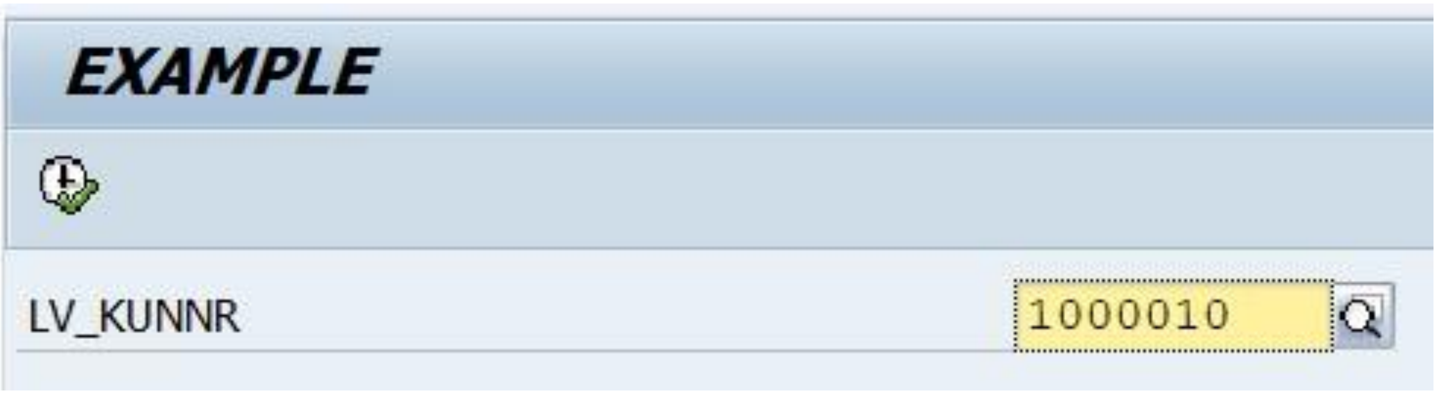
```
    it_sales_orders   = lt_table
  EXCEPTIONS
    invalid_customer = 1
    no_orders_found   = 2
    invalid_quantity = 3
    OTHERS           = 4.

IF sy-subrc = 0.
  WRITE: / 'Total Value:', lv_total_value,
         / 'Updated Discount Rate:', lv_discount_rate.
  LOOP AT lt_table INTO ls_sales_order.
    WRITE: / 'Order:', ls_sales_order-vbeln,
             'Item:', ls_sales_order-posnr,
           / 'Net Price:', ls_sales_order-netpr,
           / 'Quantity:', ls_sales_order-kwmeng.
    ULINE.
  ENDLOOP.
ELSE.
  CASE sy-subrc.
    WHEN 1.
      WRITE: / 'Error: Invalid customer.'.
    WHEN 2.
      WRITE: / 'Error: No sales orders found.'.
    WHEN 3.
      WRITE: / 'Error: Invalid quantity in sales order.'.
    WHEN OTHERS.
      WRITE: / 'Error: Unknown error occurred.'.
  ENDCASE.
ENDIF.
```

- After the function call, the code checks SY-SUBRC, which indicates the success or failure of the function module:
  - **Success (SY-SUBRC = 0)**: The program writes the LV_TOTAL_VALUE and LV_DISCOUNT_RATE to the output. It then loops through LT_TABLE, displaying each sales order’s

details (order number, item, net price, and quantity) line by line.

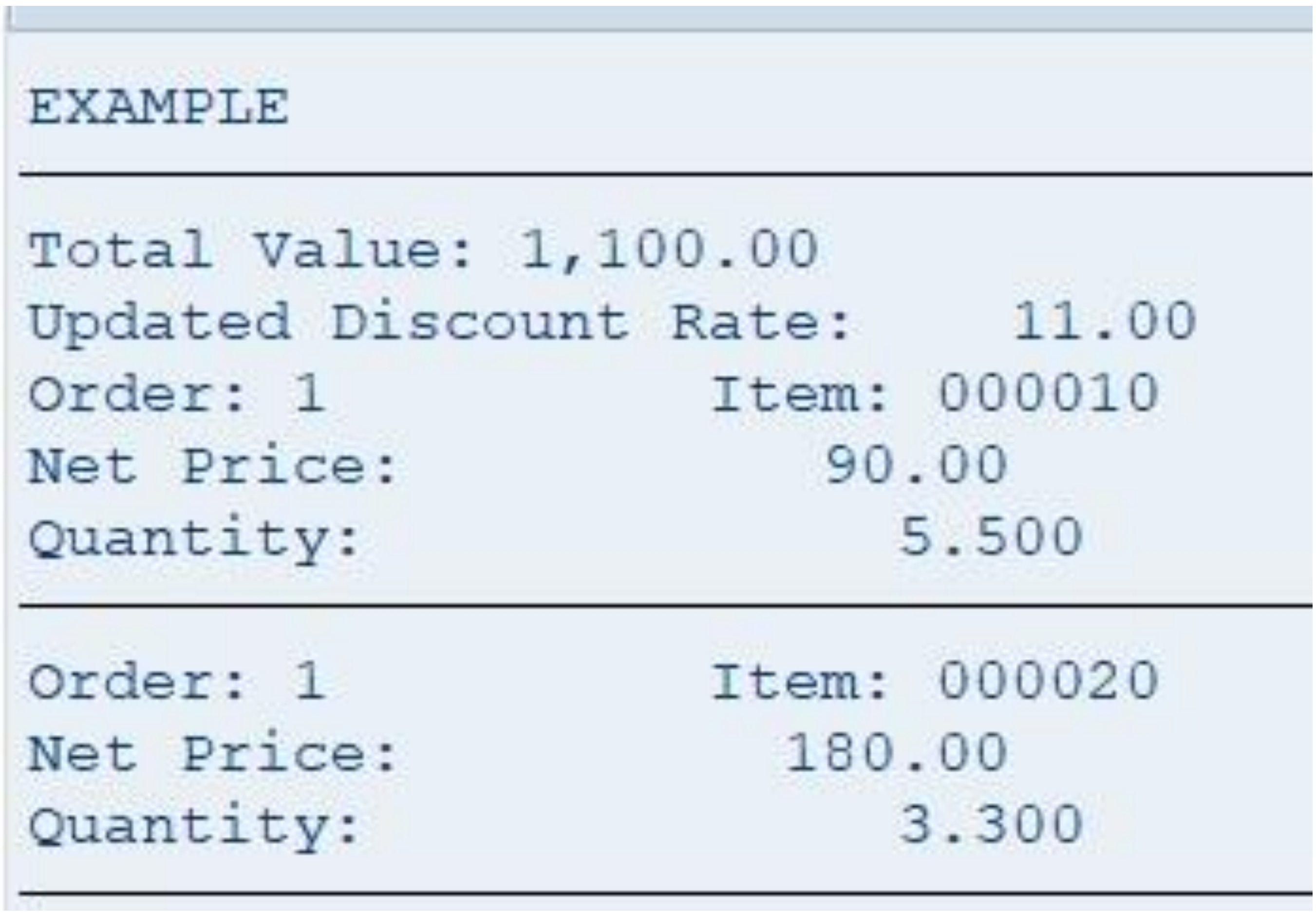
- **Failure (SY-SUBRC ≠ 0)**: The code uses a CASE statement to handle specific errors defined in the EXCEPTIONSsection of the function call:
  - INVALID_CUSTOMER = 1: Displays "Error: Invalid customer."
  - NO_ORDERS_FOUND = 2: Displays "Error: No sales orders found."
  - INVALID_QUANTITY = 3: Displays "Error: Invalid quantity in sales order."
  - OTHERS = 4: Displays "Error: Unknown error occurred."

- Up on Activation & execution of report program


EXAMPLE

LV_KUNNR 1000010

- After execution, the code checks the function module's return code (SY-SUBRC).
- IIf successful (SY-SUBRC = 0), it displays the total value, updated discount rate, and details of each sales order (order number, item, net price, and quantity).



```
EXAMPLE
------------------------------------------------------------
Total Value: 1,100.00
Updated Discount Rate:     11.00
Order: 1                 Item: 000010
Net Price:                 90.00
Quantity:                    5.500
------------------------------------------------------------
Order: 1                 Item: 000020
Net Price:                180.00
Quantity:                    3.300
------------------------------------------------------------
```

- If there's an error, it handles specific cases: invalid customer (1), no orders found (2), invalid quantity (3), or other unknown errors (4), displaying the appropriate error message.

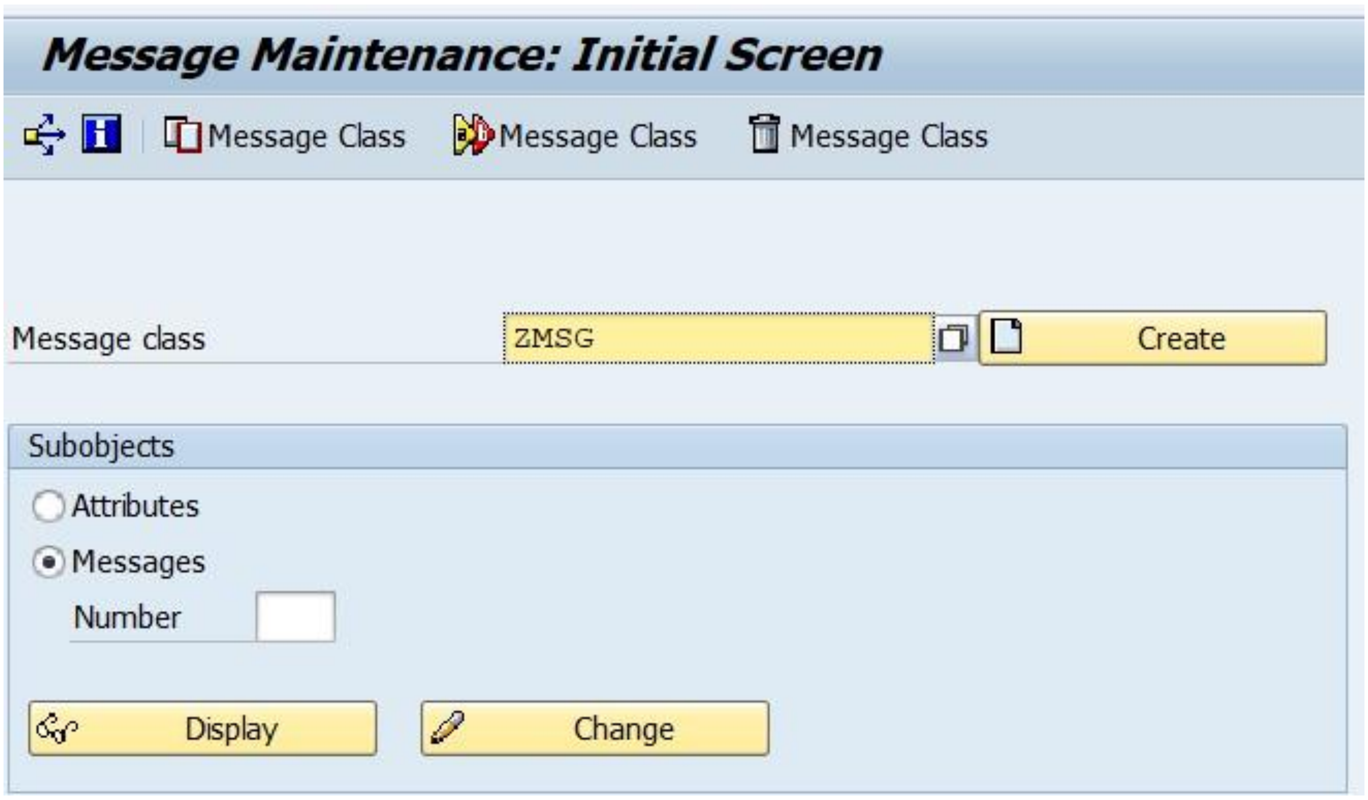
Message Class

- In SAP ABAP, a message class serves as a central repository for organizing and managing messages displayed to users during program execution.
- These messages can be of various types, such as errors, warnings, information, success, or abort messages.

- Each message within a message class is identified by a unique three-character message number (000999).

To create a custom message class like ZMSG, follow these steps:

- Open the transaction code SE91 in SAP.
- In the "Message Class" field, enter a name starting with 'Z' (e.g., ZMSG) to indicate it's a custom class. Click the "Create"
-



Message Maintenance: Initial Screen

Message class: ZMSG

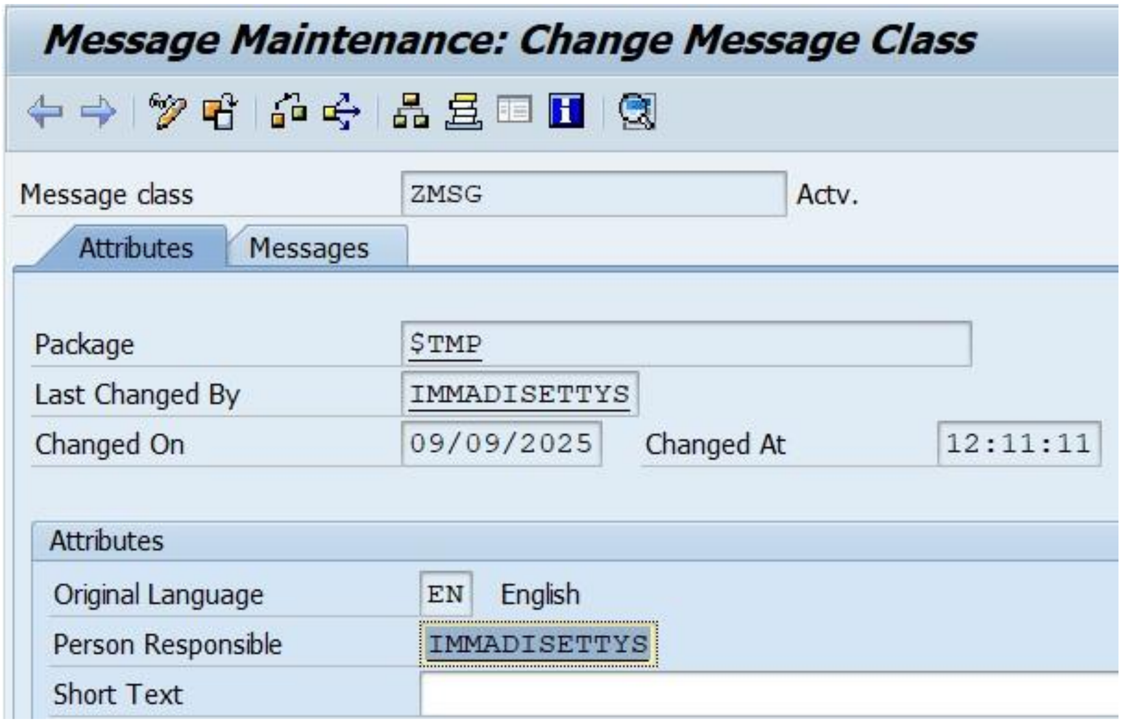
Subobjects

( ) Attributes
(●) Messages
Number

-

button.

Provide a short description for the message class (e.g., "Custom Messages for ZMSG").


Message Maintenance: Change Message Class

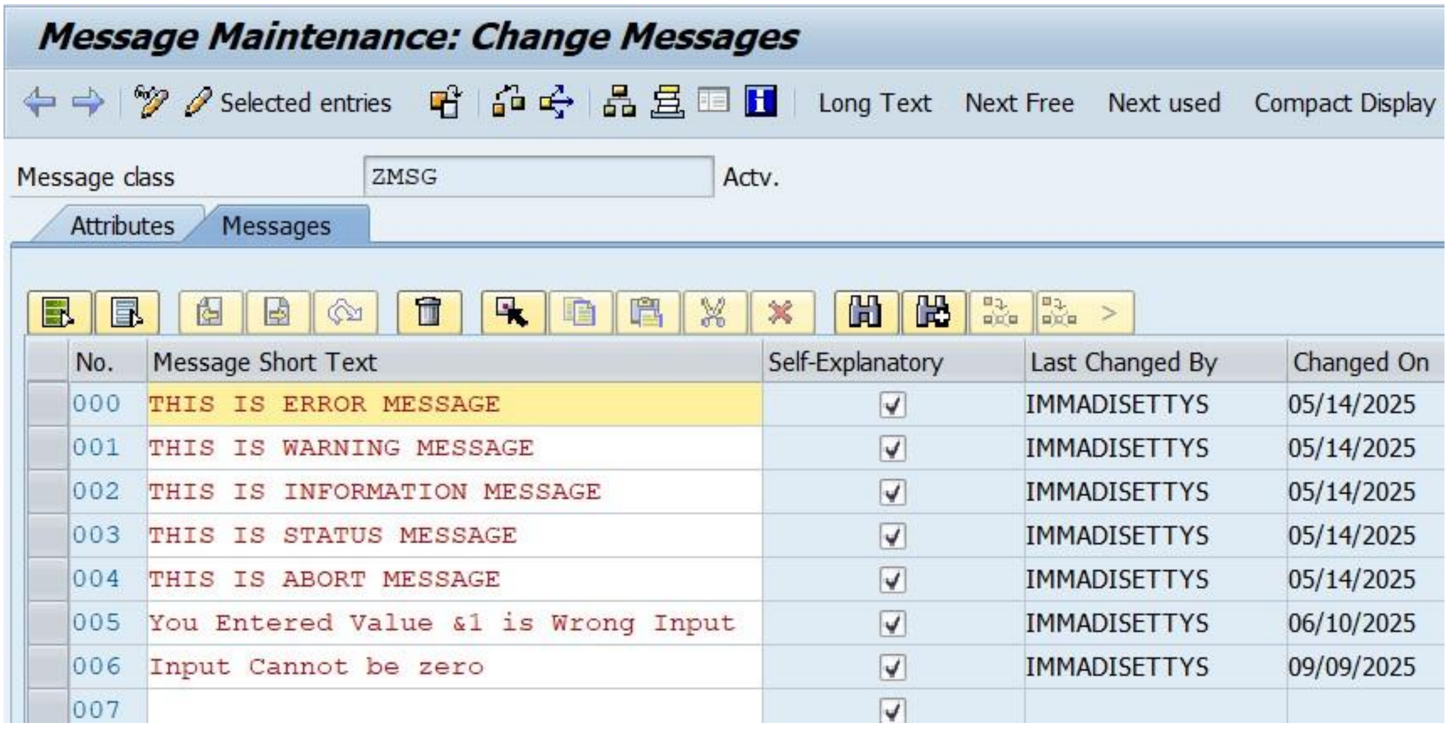
Message class: ZMSG Actv.

Package	$TMP		
Last Changed By	IMMADISETTYS		
Changed On	09/09/2025	Changed At	12:11:11

Attributes

Original Language	EN	English
Person Responsible	IMMADISETTYS	
Short Text		

- In the editor, define your messages by assigning a unique 3-digit number (e.g., 001, 002) to each message, along with the message text. You can include placeholders like '&' for variables if needed.


Message Maintenance: Change Messages

Selected entries   Long Text   Next Free   Next used   Compact Display

Message class ZMSG Actv.

Attributes   Messages

No.	Message Short Text	Self-Explanatory	Last Changed By	Changed On
000	THIS IS ERROR MESSAGE	✓	IMMADISETTYS	05/14/2025
001	THIS IS WARNING MESSAGE	✓	IMMADISETTYS	05/14/2025
002	THIS IS INFORMATION MESSAGE	✓	IMMADISETTYS	05/14/2025
003	THIS IS STATUS MESSAGE	✓	IMMADISETTYS	05/14/2025
004	THIS IS ABORT MESSAGE	✓	IMMADISETTYS	05/14/2025
005	You Entered Value &1 is Wrong Input	✓	IMMADISETTYS	06/10/2025
006	Input Cannot be zero	✓	IMMADISETTYS	09/09/2025
007		✓		

- Save the message class and assign it to a development package. or a local object ($TMP).