

BAPI:

A Business Application Programming Interface is a precisely defined interface providing access process and data in Business Applications Systems Such as SAP R/3

Benefits of BAPI:

- **Can be used in diverse languages / Development Environments**
(ABAP, Visual Basic, Java, C++, etc.)

- **Can be called from diverse platforms (COM, CORBA, Unix)**
- **Reduced development cost**
- **Reduced maintenance cost**
- **“Best-of-both-worlds” approach**
- **Rich functionality of the R/3 system**
- **User-specific front-ends**

Programming a BAPI consists of 6 major tasks:

- 1. Defining BAPI Data structures in SE11**
- 2. Program a RFC enabled BAPI function module for each method**
- 3. Create a Business object for the BAPI in the BOR**
- 4. Documentation of the BAPI**
- 5. Generate ALE interface for asynchronous BAPIs**
- 6. Generate and release**

BAPI CONVENTIONS:

Methods

- **If the BAPI to be implemented is a standardized BAPI, use the generic names, for example, GetList, GetDetail.**
- **The method name must be in English (maximum 30 characters).**
- **The individual components of a BAPI name are separated by the use of upper and lower case. Example: GetList**
Underscores ("_") are not allowed in BAPI names.
- **Each BAPI has a return parameter that is either an export parameter or an export table.**
- **So that customers can enhance BAPIs, each BAPI must have an ExtensionIn and an ExtensionOut parameter.**

Parameters

- **If standardized parameters are used, you have to use the names specified for standardized parameters.**
- **BAPI parameter names should be as meaningful as possible. Poorly chosen names include abbreviations and technical names (e.g. "flag", table names, etc.).**
The parameter and field names must be in English with a maximum of 30 characters.
- **The components of a parameter name in the BOR are separated by upper and lower case letters to make them easier to read. Example:**
CompanyCodeDetail
- **Values that belong to each other semantically should be grouped together in one structured parameter, instead of using several scalar parameters.**
- **For ISO-relevant fields (country, language, unit of measure, currency), additional fields for ISO codes are provided.**
- **Unit of measure fields must accompany all quantity fields and currency identifiers must accompany currency amount fields.**

Standardized BAPIs

Some BAPIs provide basic functions and can be used for most SAP business object types. These BAPIs should be implemented the same for all business object types. Standardized BAPIs are easier to use and prevent users having to deal with a number of different BAPIs. Whenever possible, a standardized BAPI must be used in preference to an individual BAPI.

The following standardized BAPIs are provided:

Reading instances of SAP business objects

GetList () With the BAPI GetList you can select a range of object key values, for example, company codes and material numbers.
The BAPI GetList() is a class method.

GetDetail() With the BAPI GetDetail() the details of an instance of a business object type are retrieved and returned to the calling program. The instance is identified via its key. The BAPI GetDetail() is an instance method.

BAPIs that can create, change or delete instances of a business object type

The following BAPIs of the same object type have to be programmed so that they can be called several times within one transaction. For example, if, after sales order 1 has been created, a second sales order 2 is created in the same transaction, the second BAPI call must not affect the consistency of the sales order 2. After completing the transaction with a COMMIT WORK, both the orders are saved consistently in the database.

Create() and CreateFromData() The BAPIs Create() and CreateFromData() create an instance of an SAP business object type, for example, a purchase order. These BAPIs are class methods.

Change() The BAPI Change() changes an existing instance of an SAP business object type, for example, a purchase order. The BAPI Change () is an instance method.

Delete() and Undelete() The BAPI Delete() deletes an instance of an SAP business object type from the database or sets a deletion flag.
The BAPI Undelete() removes a deletion flag. These BAPIs are instance methods.

- Cancel ()** Unlike the BAPI Delete(), the BAPI Cancel() cancels an instance of a business object type. The instance to be cancelled remains in the database and an additional instance is created and this is the one that is actually canceled. The Cancel() BAPI is an instance method.
- Add<subobject> () and Remove<subobject> ()** The BAPI Add<subobject> adds a subobject to an existing object instance and the BAPI and Remove<subobject> removes a subobject from an object instance. These BAPIs are instance methods.

BAPIs for Mass Data Processing

The BAPIs listed above for creating and changing data can also be used for mass processing. For more information see BAPIs for Mass Data Transfer [Extern]

BAPIs for Replicating Business Object Instances

- Replicate() and SaveReplica()** The BAPIs Replicate() and SaveReplica() are implemented as methods of replicable business object types. They enable specific instances of an object type to be copied to one or more different systems. These BAPIs are used mainly to transfer data between distributed systems within the context of Application Link Enabling (ALE). These BAPIs are class methods.

Other Less Used Standardized BAPIs

- Programming GetStatus() BAPIs [Extern]
- Programming ExistenceCheck() BAPIs [Extern]

Standardized Parameters

There are some parameters that can be created for various BAPIs because they contain the same or the equivalent data in all BAPIs. They should be implemented the same in all BAPIs.

Address parameters	Specific reference structures are defined for address parameters in BAPIs. You should copy these structures to use in your BAPI, especially if the underlying object type uses the central address management (CAM).
Change Parameters	In BAPIs that cause database changes (for example, Change() and Create() BAPIs) you must be able to distinguish between parameter fields that contain modified values and parameter fields that have not been modified. This distinction is made through the use of standardized parameters.
Extension parameters	The parameters ExtensionIn and ExtensionOut provides customers with a mechanism that enables BAPIs to be enhanced without modifications.
Return Parameters	Each BAPI must have an export return parameter for returning messages to the calling application. To provide application programmers with a consistent error handling process for BAPI calls, all return parameters must be implemented in the same, standardized way.
Selection Parameters	Standardized selection parameters are used in BAPIs that can be used to search for specific instances of a business object type (e.g. in GetList()). These parameters enable the BAPI caller to specify the relevant selection criteria.
Test Run Parameters	The parameter TestRun is used in write BAPIs (Create() and Change()), to check the entries for the object instance in the database before actually creating the object instance. The creation of the object instance is only simulated and data is not updated.
Text Transfer Parameters	To transfer BAPI documentation texts (e.g. the documentation of a business object type), you have to create standardized text transfer parameters.

Important things to remember..

It is important to follow the guidelines below when develop9ng BAPIs:

- **BAPIs must not contain CALL TRANSACTION or SUBMIT REPORT**
- **BAPIs must not invoke a COMMIT WORK. instead use the BAPI Transaction Commit to execute the commit after the BAPI has executed.**
- **BAPI structures must not use includes.**
- **There should be no functional dependencies between two BAPIs**
- **BAPIs must perform there own authorization check**
- **BAPIs should not use dialogs**
- **BAPIs must not cause the program to abort or terminate. Relevant messages must be communicated through the return parameter.**

.....
.....
Copied from <http://help.sap.com/>
.....
.....

BOR

Definition

The Business Object Repository (BOR) is the object-oriented repository in the R/3 System. It contains the SAP business object types and SAP interface types as well as their components, such as methods, attributes and events.

BAPIs are defined as methods of SAP business object types (or SAP interface types) in the BOR. Thus defined, the BAPIs become standard with full stability guarantees as regards their content and interface.

Use

The BOR has the following functions for SAP business object types and their BAPIs:

- Provides an object oriented view of R/3 System data and processes.

R/3 application functions are accessed using methods (BAPIs) of SAP Business Objects. Implementation information is encapsulated; only the interface functionality of the method is visible to the user.

- Arranges the various interfaces in accordance with the component hierarchy, enabling functions to be searched and retrieved quickly and simply.

This finds the functionality searched for quickly and simply.

- Manages BAPIs in release updates.

BAPI interface enhancements made by adding parameters are recorded in the BOR. Previous interface versions can thus be reconstructed at any time. When a BAPI is created the release version of the new BAPI is recorded in the BOR. The same applies when any interface parameter is created.

The version control of the function module that a BAPI is based on is managed in the Function Builder.

- Ensures interface stability.

Any interface changes that are carried out in the BOR, are automatically checked for syntax compatibility against the associated development objects in the ABAP Dictionary.

Integration

A BAPI is implemented as a function module, that is stored and described in the Function Builder. You should only define a BAPI as a method of an SAP business object type in the BOR, if the function module that the BAPI is based on has been fully implemented.

Access to the BOR is restricted at SAP.

Defining Methods in the BOR Using the BOR/BAPI Wizard

Prerequisites

If the function module which your BAPI is based on has been fully implemented or modified you can define it as a method of an SAP business object type or SAP interface type in the Business Object Repository (BOR). You use the BOR/BAPI Wizard to do this.

Procedure

First find the relevant SAP business object type in the BOR:

1. Choose *Tools* → *Business Framework* → *BAPI Development* → *Business Object Builder*.

On the initial Business Object Builder screen you can directly access the SAP business object type or interface type if you know the technical name of the object (object type). You have already identified the technical name of the object.

Otherwise choose *Business Object Repository*.

- To display object types, in the next dialog box indicate whether you want to display all object types or only business object types. Then choose *Continue*.
- To display SAP interface types, in the next dialog box choose *Other settings* and then select *Interface*.

The application hierarchy is displayed. Search for the required business object type or interface type in the application hierarchy and double click it to open it.

2. When the business object type or interface type is displayed, choose *Change*.

Creating BAPIs as Methods of Business Object Types or Interface Types

To define your BAPI as a method of a business object type or interface type:

1. Select *Utilities* → *API Methods* → *Add method*.
2. In the next dialog box enter the name of the function module, for example, *BAPI_COMPANYCODE_GETDETAIL*, and choose *Continue*.
3. In the next dialog box specify the following information for the method to be defined:

- Method

A default name for the method is provided, based on the name of the function module. You may have to modify the suggested name:

Example: If the name of the function module is *SALESORDER_GETSTATUS*, the suggested method name might be *BapiSalesorderGetstatus*. You should edit this so that the resulting name is *GetStatus*.

- Texts

Enter meaningful descriptions for your BAPI.

- Radio buttons *Dialog*, *Synchronous*

Enter relevant details for your BAPI. Make sure that a BAPI is not dialog orientated. BAPIs are usually implemented synchronously.

4. Choose *Next Step*.

A list of parameters and default names is displayed which you need to edit as required. Modify the parameter names as follows:

- Each new word in the parameter name must start with a capital letter, for example, *CompanyCodeDetail*.
- Make sure that the parameter names of the method in the BOR are identical to the parameter names in the function module except for the upper/lower case letters.
- The import and export behavior of the table parameters must be correctly defined in the BOR.

Reason : In contrast to the function module, in the BOR you can differentiate between import and export for tables also. You should therefore only select the standard option *Import/export*, if the table is actually going to be imported **and** exported.

- The return parameter is always defined as an *export* parameter.

5. Choose *Next Step*.

To create the method choose *Yes* in the next dialog box.

Result

After the program has been generated and executed, check that all the definitions have been made correctly by the BOR/BAPI Wizard. To do this, look at the newly created method of the business object type or interface type.



The BOR/BAPI Wizard is used only to create new BAPIs for the first time. It is not used to make changes to existing BAPIs.

If you make changes to the underlying function module after you have created the BAPI in the BOR, for example, if you make compatible interface enhancements or modify short texts, such changes do **not** automatically take effect in the BOR. You have to make these changes manually in the BOR. For information about creating and modifying business object types

.....
.....

Copied from <http://help.sap.com/>

.....
.....
This BAPI reads system status for a production order from table JEST and system status text from table TJ02T

Name	ZGetOrderStatus
Function group	ZBAPISTATUS
Function module:	Z_BAPI_GET_ORDER_STATUS
Import parameters:	ORDER_STATUS_IMPORT type ZBAPI_ORDER_STATUS_IMPORT: <ul style="list-style-type: none">• AUFNR Order number (Keyfield)• SPRAS Language• Excludelinactive - Checkbox - Exclude inactive status
Tables	T_BAPISTAT type ZBAPISTAT: <ul style="list-style-type: none">• OBJNR like JEST-OBJNR• STAT like JEST-STAT• INACT like JEST-INACT• TXT04 like TJ02T-TXT04• TXT30 like TJ02T-TXT30
Export parameters	RETURN like BAPIRETURN

One thing you have to remember while creating a BAPI is don't assign it to any temporary package because at last we have to release the BAPI

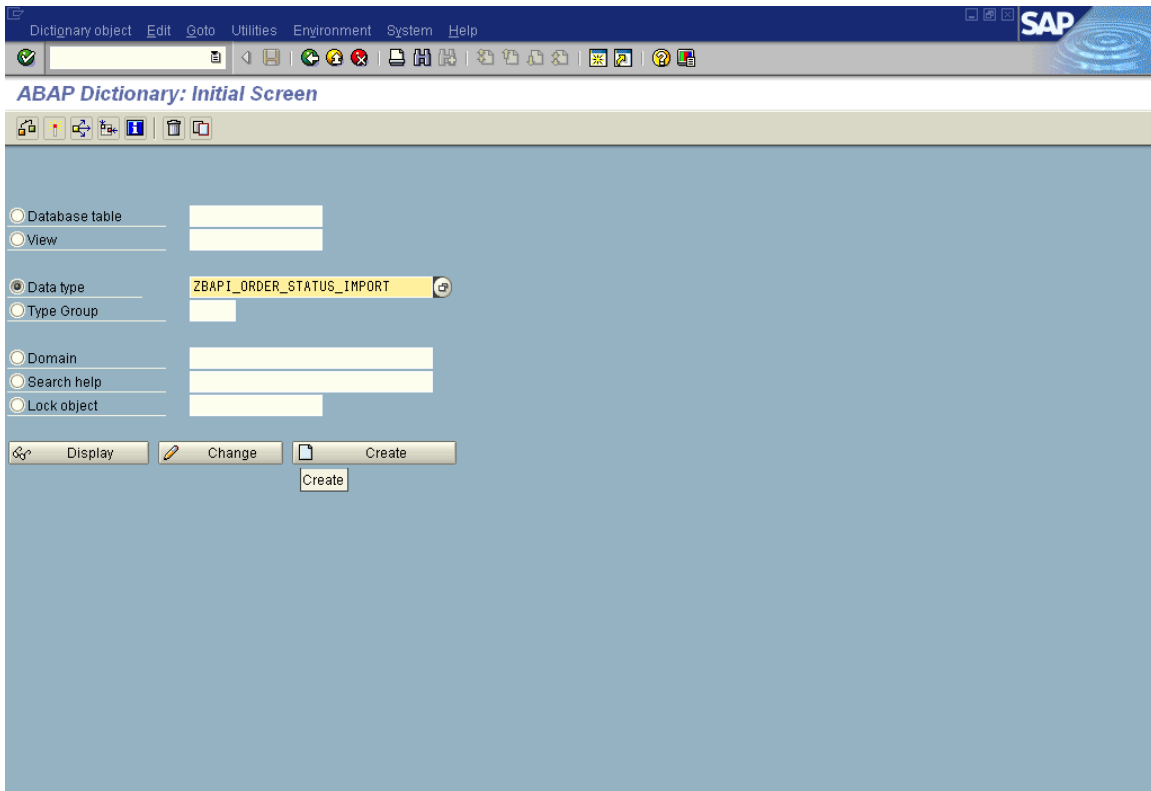
Import and Export parameters of an BAPI should be defined under a structure only it should not refer to an Table

Define a structures for the BAPI

Create new structure for our BAPI

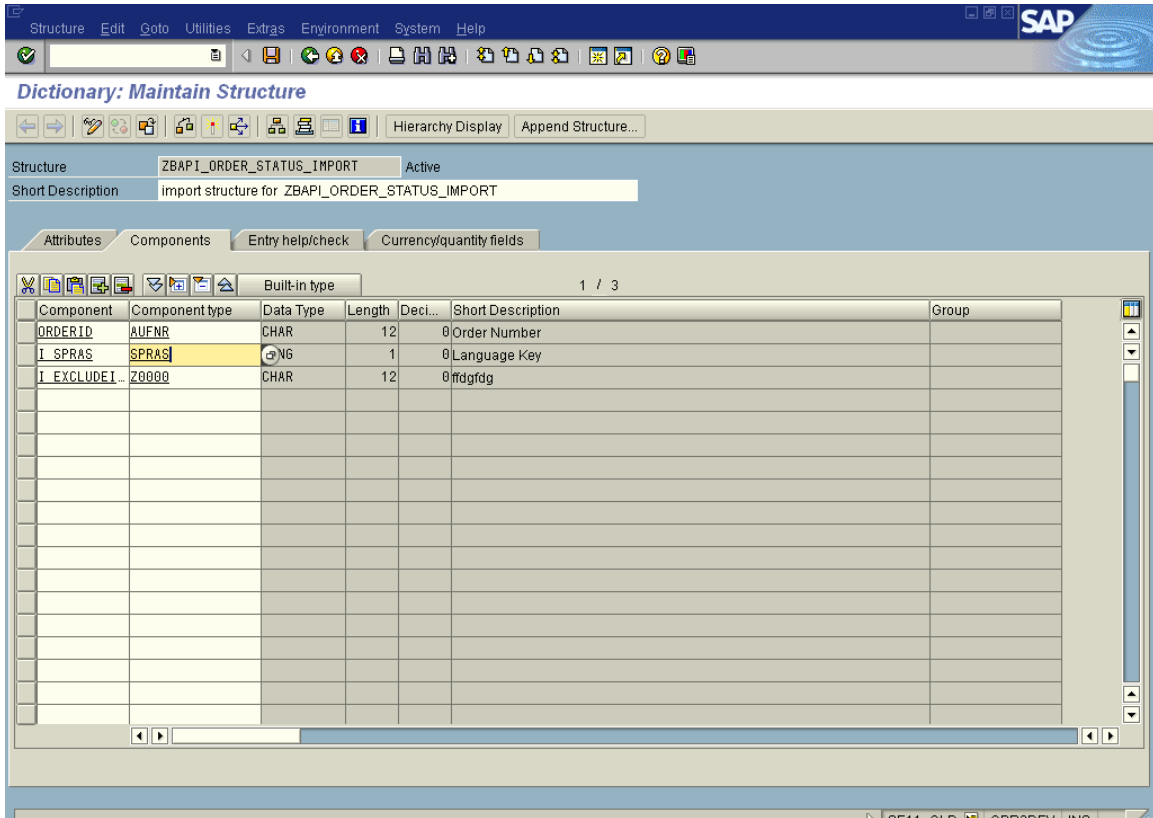
Goto Tcode SE11 select the radio button Data Type

Data Type → ZBAPI_ORDER_STATUS_IMPORT



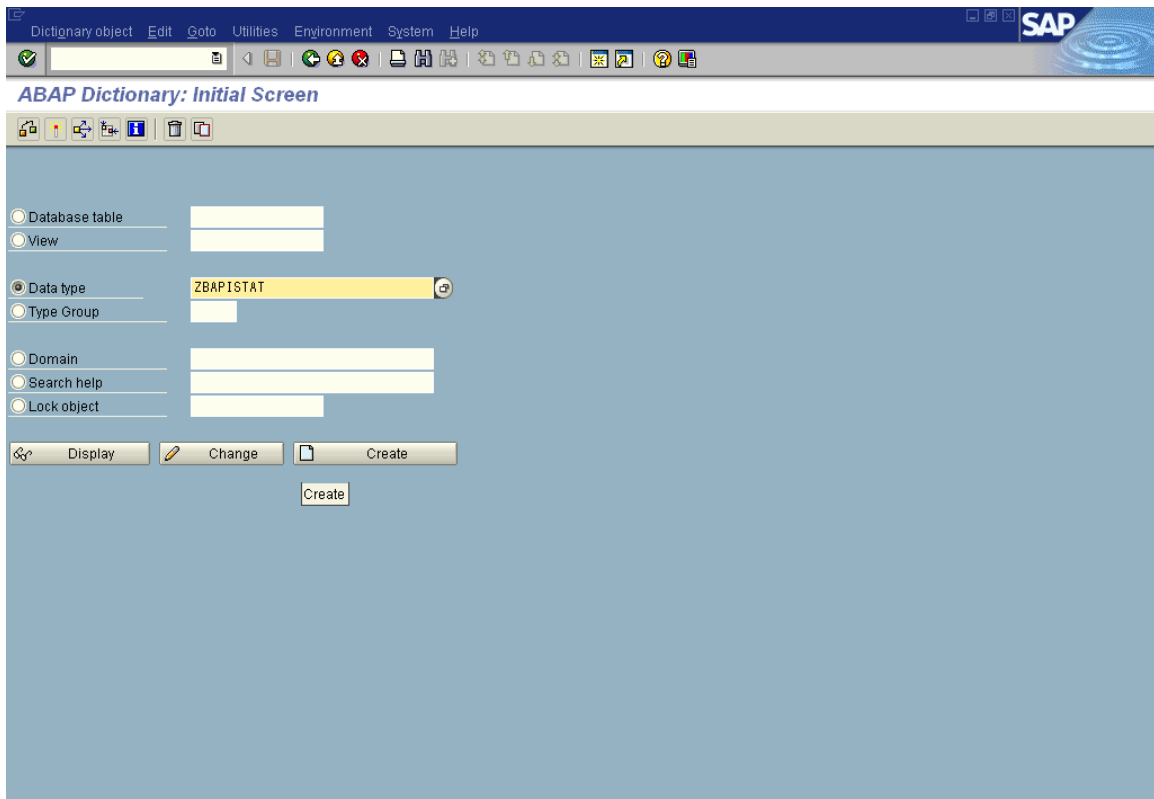
Define the following fields in the structure

- **ORDERID** Order number (Keyfield)
- **I_SPRAS** Language
- **I_EXCLUDEINACTIVE** - Checkbox - Exclude inactive status



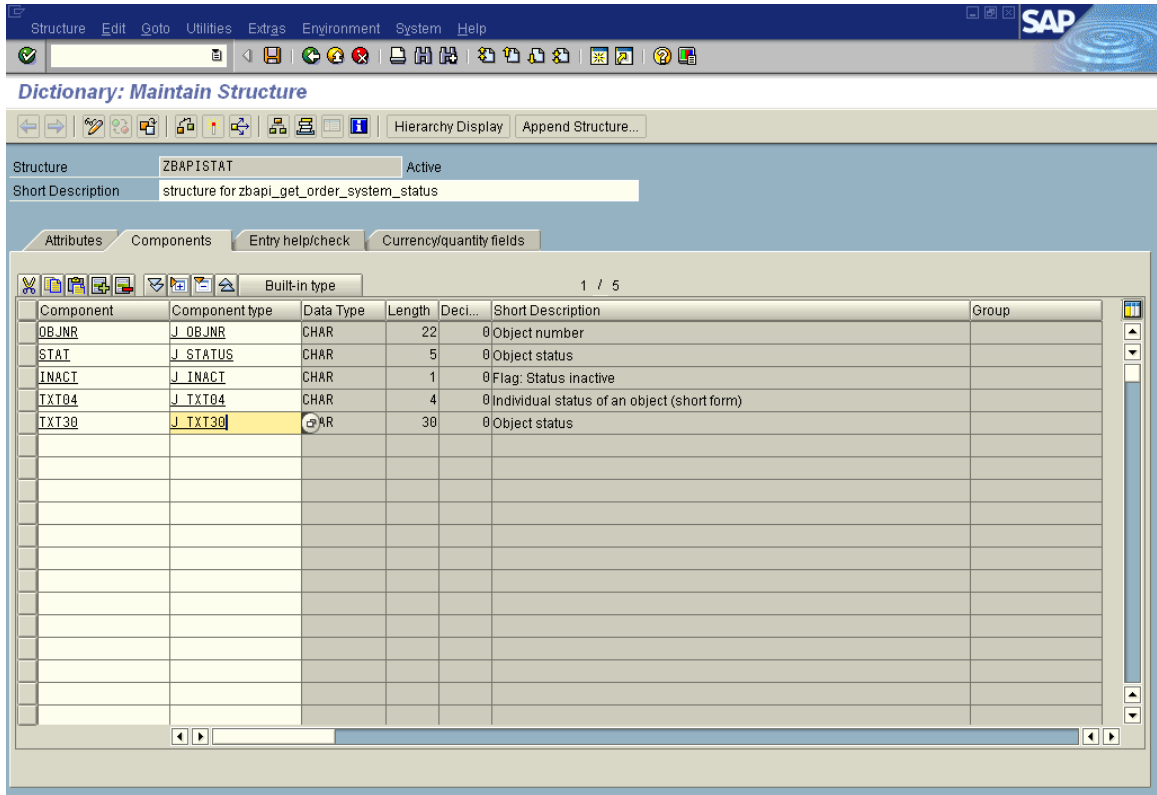
Save it & activate it.....

Now create another structure ZBAPISTAT



Declare these following fields

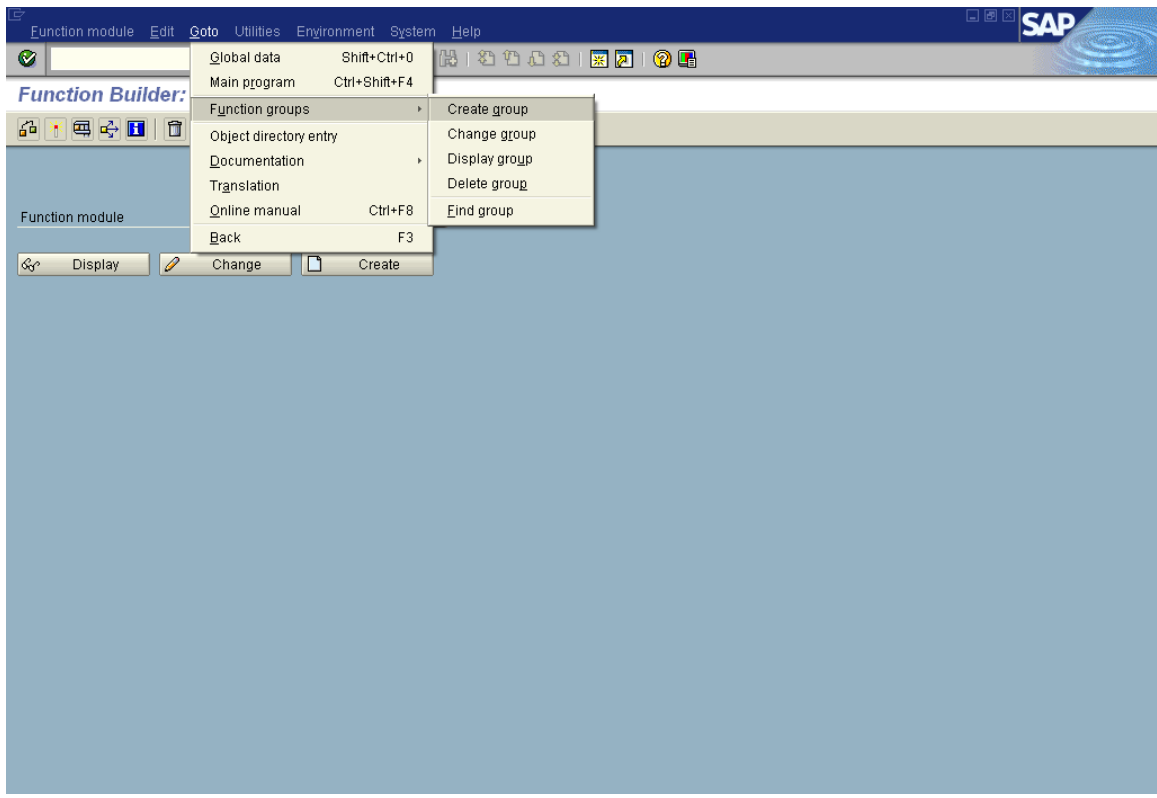
- **OBJNR**
- **STAT**
- **INACT**
- **TXT04**
- **TXT30**

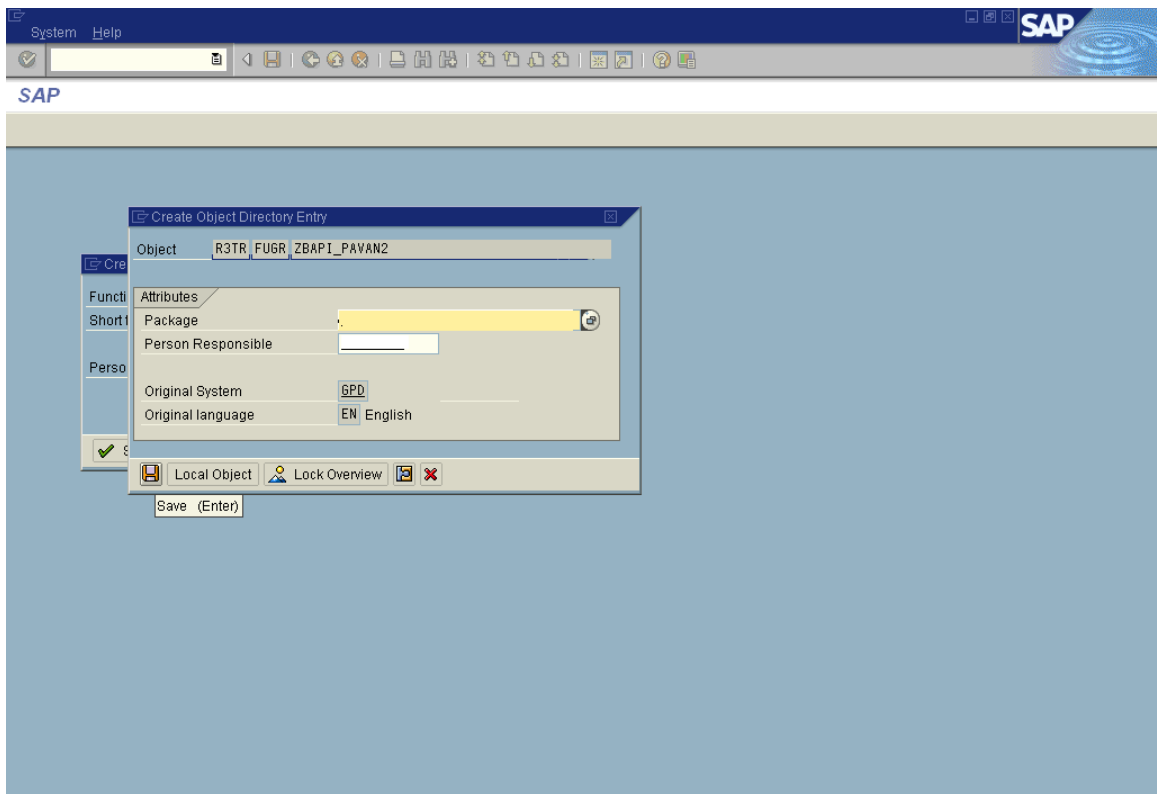
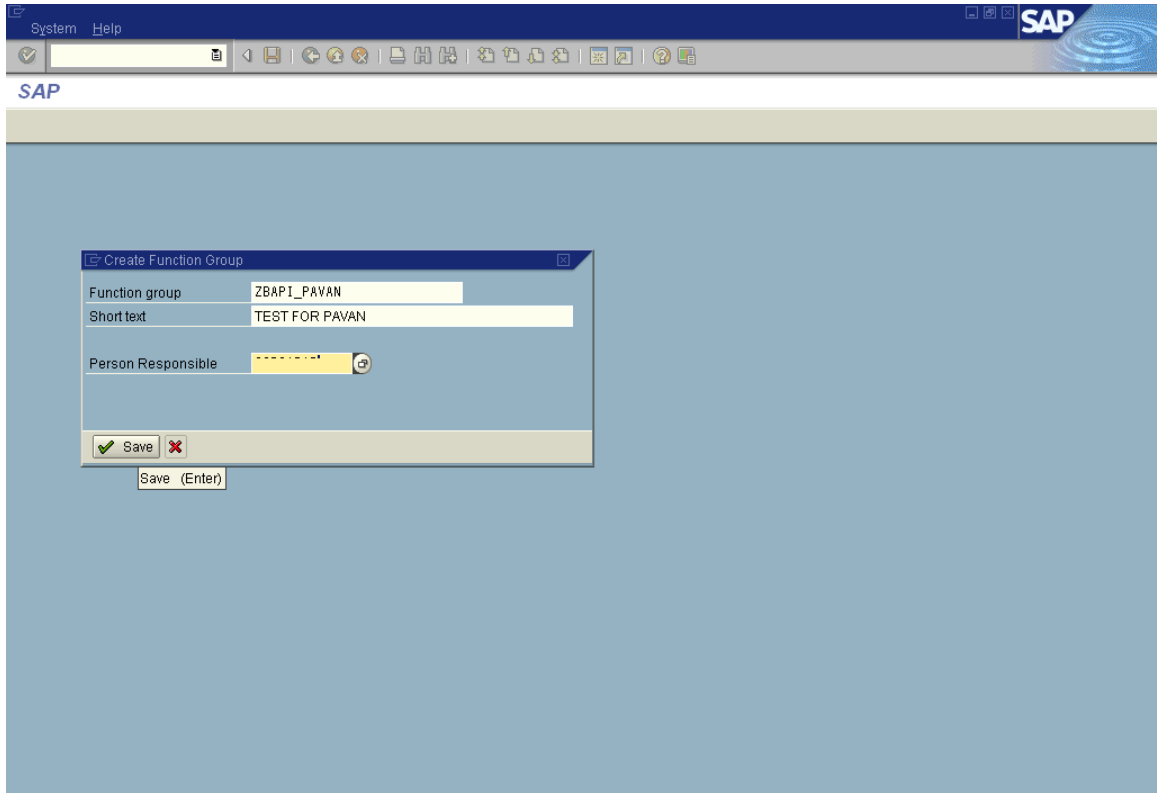


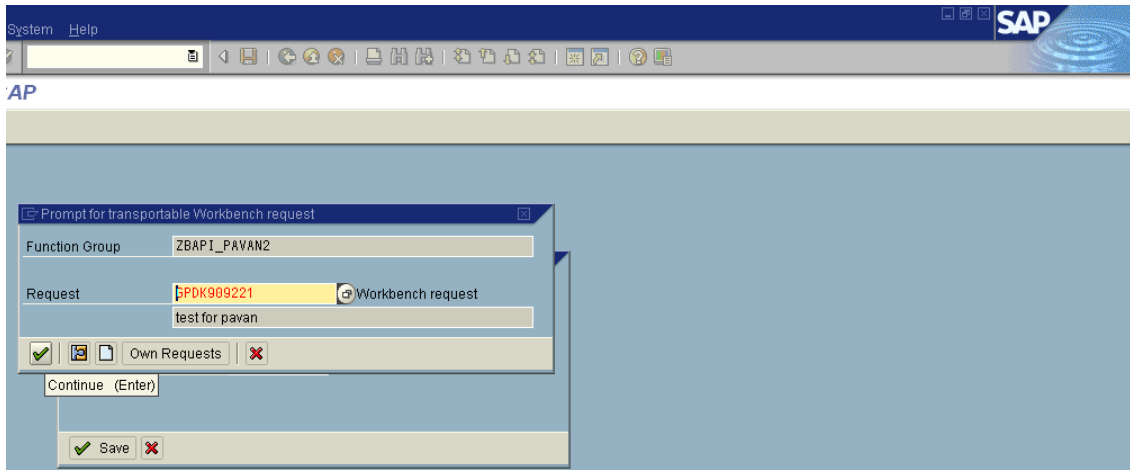
Now the structures has been created for our BAPI next goto Tcode SE37.

For every BAPI or Function module there will be a separate function group which we cant use for another BAPI or Function module so create your own function group for each BAPI you create

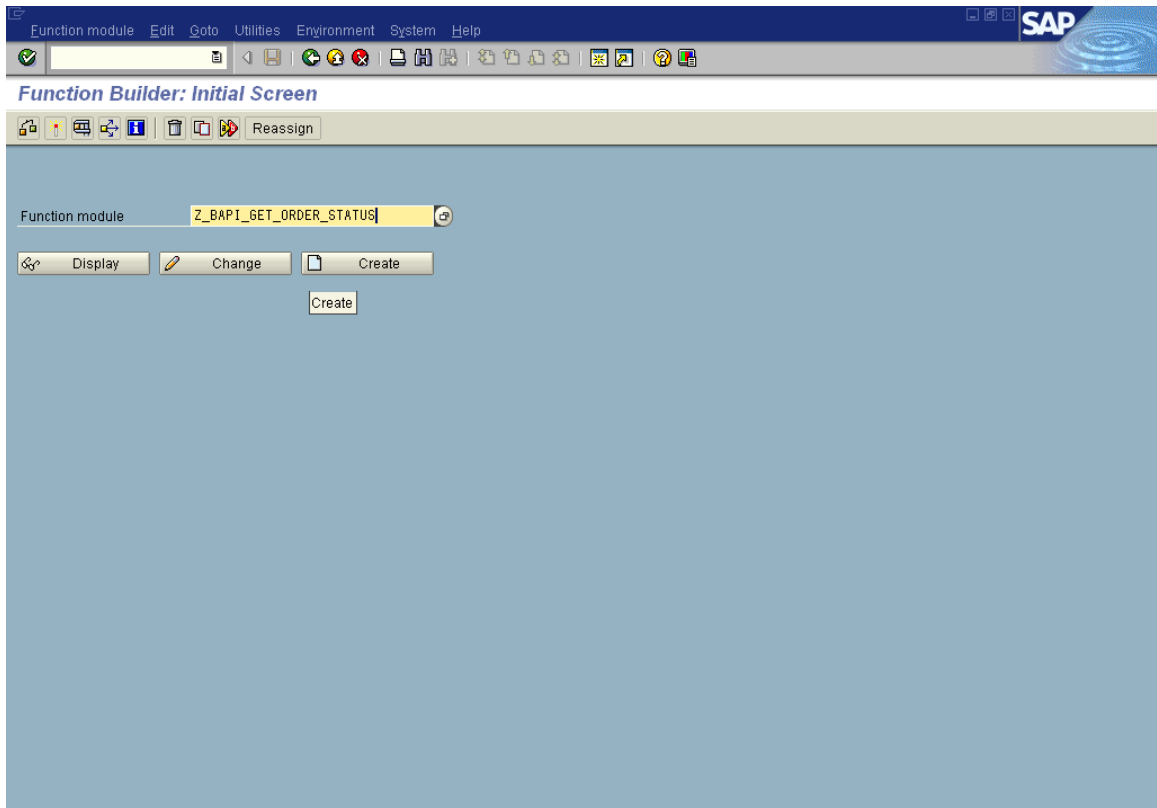
Create Function Group:



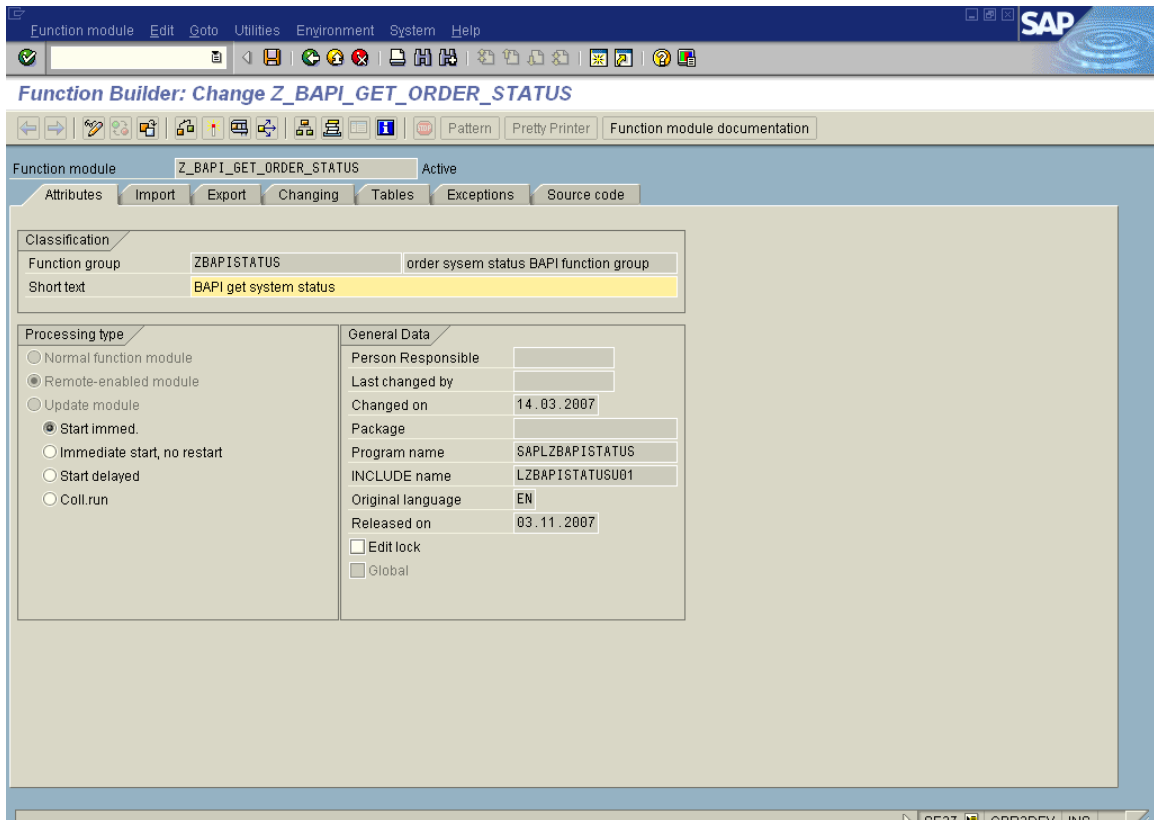




Create new BAPI:



In the attributes fill all necessary parameters as shown below



- Under the attributes tab remember to select Processing Type Remote Enabled module, otherwise the function module cannot be invoked via RFC and used as a BAPI
- Import/Export parameters can only be BY VALUE for an RFC enabled function module
- We are only creating one BAPI in this example, but you can create related BAPIs in the same function pool, so they will be able to share global data.

Export Parameters:

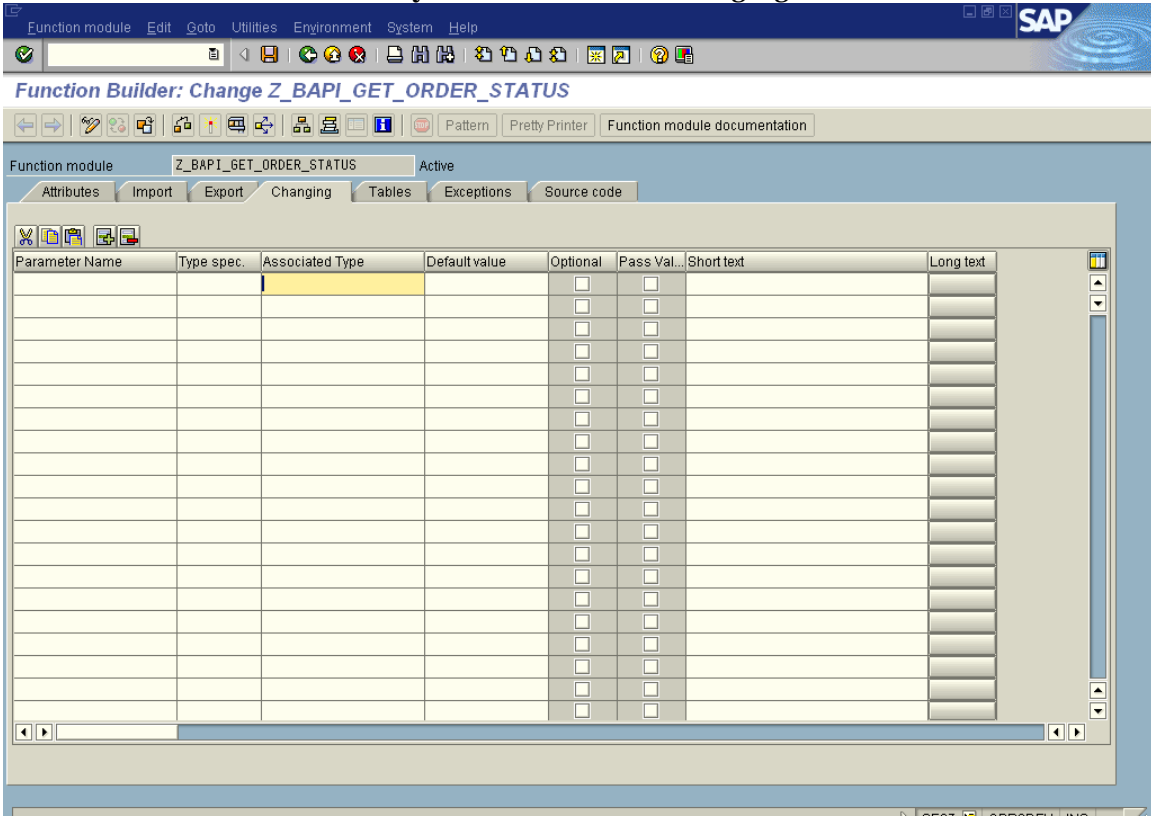
As I have said earlier in the output parameters there will be only some constant parameters should be passed

BAPIRETURN
BAPIRETURN1
BAPIRET1
BAPIRET2

BAPI Return Structure Type:

- **Type Message type**
 - **Blank or "S"=Success**
 - **"E"=Error**
 - **"W"=Warning**
 - **"I"=Information**
 - **"A"=Abort**
- **Message** **Message text**
- **Log_No** **Application Log Number**
- **Log_Msg_No** **Application Log Message Serial Number**
- **Message_V1 -V4** **Message variables**

There is no need to declare any values in the tab *changing*



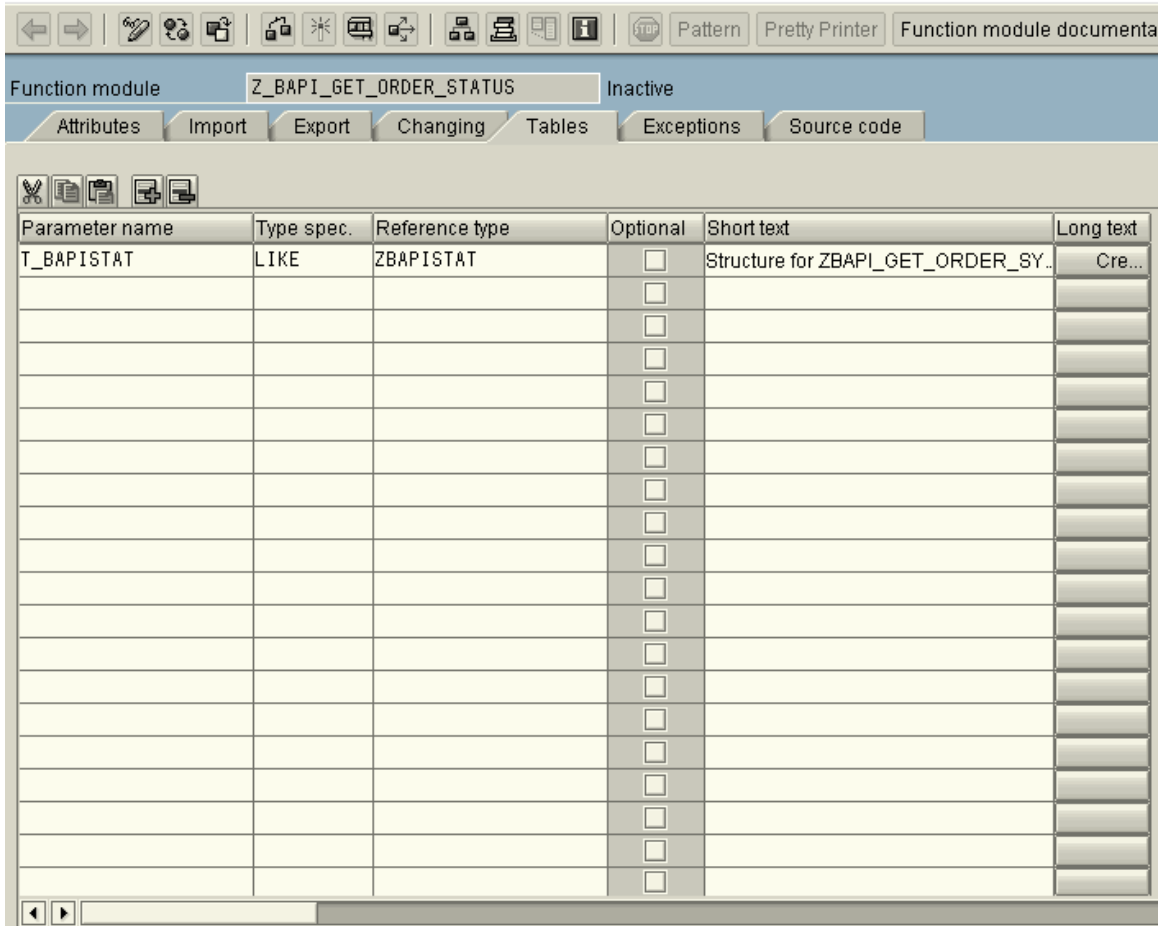
Tables:

Parameter Name : *T_BAPISTAT*

Type Specification : *LIKE*

Associated Type : *ZBAPISTAT*

Function Builder: Change Z_BAPI_GET_ORDER_STATUS



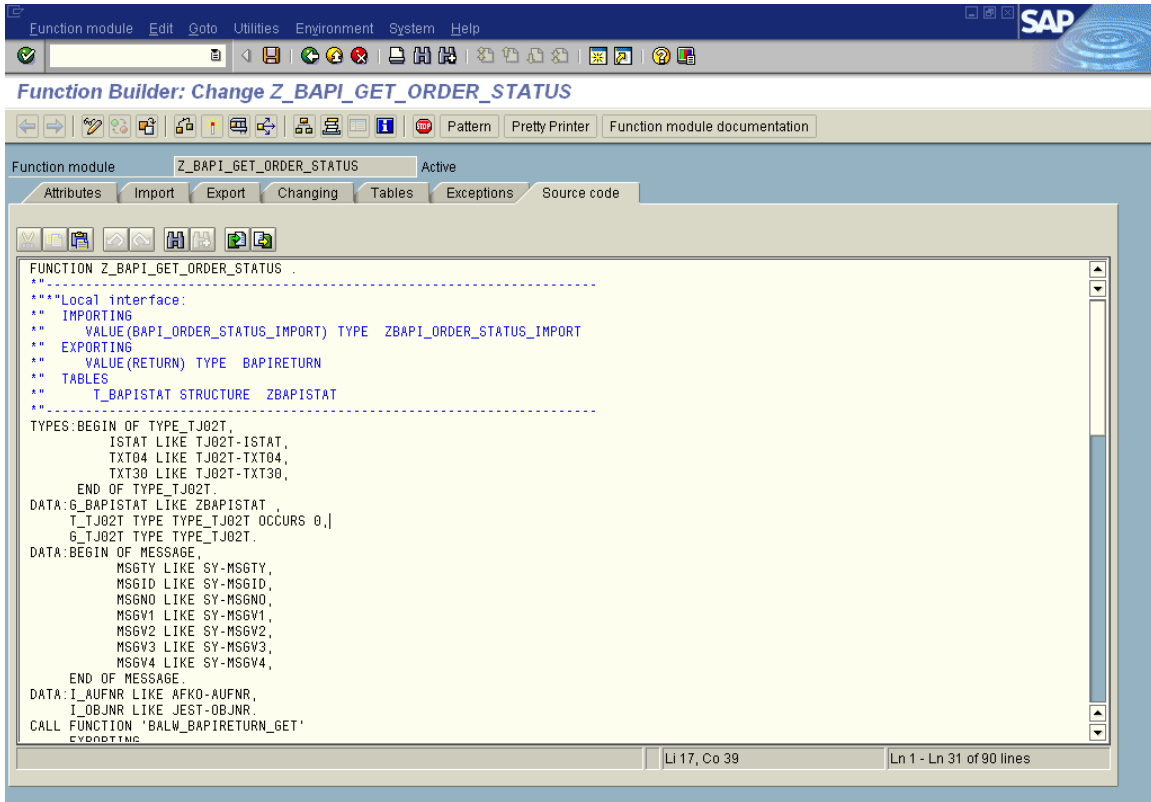
The screenshot shows the SAP Function Builder interface for the function module Z_BAPI_GET_ORDER_STATUS. The 'Tables' tab is selected, displaying a table with the following columns: Parameter name, Type spec., Reference type, Optional, Short text, and Long text. The first row contains the parameter T_BAPISTAT, which is of type LIKE and references the table ZBAPISTAT. The 'Optional' column for this parameter has a checkbox that is currently unchecked. There are 17 rows in total, with the first row containing data and the rest being empty.

Parameter name	Type spec.	Reference type	Optional	Short text	Long text
T_BAPISTAT	LIKE	ZBAPISTAT	<input type="checkbox"/>	Structure for ZBAPI_GET_ORDER_SY..	Cre...
			<input type="checkbox"/>		
			<input type="checkbox"/>		
			<input type="checkbox"/>		
			<input type="checkbox"/>		
			<input type="checkbox"/>		
			<input type="checkbox"/>		
			<input type="checkbox"/>		
			<input type="checkbox"/>		
			<input type="checkbox"/>		
			<input type="checkbox"/>		
			<input type="checkbox"/>		
			<input type="checkbox"/>		
			<input type="checkbox"/>		
			<input type="checkbox"/>		
			<input type="checkbox"/>		
			<input type="checkbox"/>		
			<input type="checkbox"/>		

Code

Notes:

- The subroutine SET_RETURN_MESSAGE is a standard routine used for BAPIs that use the BAPIRETURN structure
- In form Z_BAPI_GET_ORDER_SYSTEM_STATUS there is a test *IF I = 2*. If the test is true a message is displayed. The condition will obviously never be true, and we will never want to display a message in a BAPI. The reason why it is included is, that it create a reference for the message, so that the WHERE USED functionality can be used for the message. This is the SAP standard way to handle it, copied from the Company Code GetList BAPI.



The screenshot displays the SAP Function Builder interface for the function module `Z_BAPI_GET_ORDER_STATUS`. The window title is "Function Builder: Change Z_BAPI_GET_ORDER_STATUS". The interface includes a menu bar (Function module, Edit, Goto, Utilities, Environment, System, Help) and a toolbar with various icons. Below the menu bar, there are tabs for "Attributes", "Import", "Export", "Changing", "Tables", "Exceptions", and "Source code". The "Source code" tab is active, showing the following ABAP code:

```
FUNCTION Z_BAPI_GET_ORDER_STATUS .
-----
**Local interface:
** IMPORTING
**   VALUE(BAPI_ORDER_STATUS_IMPORT) TYPE ZBAPI_ORDER_STATUS_IMPORT
** EXPORTING
**   VALUE(RETURN) TYPE BAPIRETURN
** TABLES
**   T_BAPISTAT STRUCTURE ZBAPISTAT
-----
TYPES:BEGIN OF TYPE_TJ02T,
       ISTAT LIKE TJ02T-ISTAT,
       TXT04 LIKE TJ02T-TXT04,
       TXT30 LIKE TJ02T-TXT30,
     END OF TYPE_TJ02T.
DATA:G_BAPISTAT LIKE ZBAPISTAT,
      T_TJ02T TYPE TYPE_TJ02T OCCURS 0,
      G_TJ02T TYPE TYPE_TJ02T.
DATA:BEGIN OF MESSAGE,
       MSGTY LIKE SY-MSGTY,
       MSGID LIKE SY-MSGID,
       MSGNO LIKE SY-MSGNO,
       MSGV1 LIKE SY-MSGV1,
       MSGV2 LIKE SY-MSGV2,
       MSGV3 LIKE SY-MSGV3,
       MSGV4 LIKE SY-MSGV4,
     END OF MESSAGE.
DATA:I_AUFNR LIKE AFKO-AUFNR,
      I_OBJNR LIKE JEST-OBJNR.
CALL FUNCTION 'BALW_BAPIRETURN_GET'
  EXPORTING
    EXPORTING
```

The status bar at the bottom indicates "Ln 17, Co 39" and "Ln 1 - Ln 31 of 90 lines".

Complete coding in BAPI :

```
FUNCTION Z_BAPI_GET_ORDER_STATUS .
*-----
*""Local interface:
*  IMPORTING
*    VALUE(BAPI_ORDER_STATUS_IMPORT) TYPE ZBAPI_ORDER_STATUS_IMPORT
*  EXPORTING
*    VALUE(RETURN) TYPE BAPIRETURN
*  TABLES
*    T_BAPISTAT STRUCTURE ZBAPISTAT
*-----
TYPES:BEGIN OF TYPE_TJ02T,
       ISTAT LIKE TJ02T-ISTAT,
       TXT04 LIKE TJ02T-TXT04,
       TXT30 LIKE TJ02T-TXT30,
     END OF TYPE_TJ02T.
DATA:G_BAPISTAT LIKE ZBAPISTAT ,
     T_TJ02T TYPE TYPE_TJ02T OCCURS 0,
     G_TJ02T TYPE TYPE_TJ02T.
DATA:BEGIN OF MESSAGE,
       MSGTY LIKE SY-MSGTY,
       MSGID LIKE SY-MSGID,
       MSGNO LIKE SY-MSGNO,
       MSGV1 LIKE SY-MSGV1,
       MSGV2 LIKE SY-MSGV2,
       MSGV3 LIKE SY-MSGV3,
       MSGV4 LIKE SY-MSGV4,
     END OF MESSAGE.
DATA:I_AUFNR LIKE AFKO-AUFNR,
     I_OBJNR LIKE JEST-OBJNR.
CALL FUNCTION 'BALW_BAPIRETURN_GET'
  EXPORTING
    TYPE      = MESSAGE-MSGTY
    CL        = MESSAGE-MSGID
    NUMBER    = MESSAGE-MSGNO
    PAR1      = MESSAGE-MSGV1
    PAR2      = MESSAGE-MSGV2
    PAR3      = MESSAGE-MSGV3
    PAR4      = MESSAGE-MSGV4
  IMPORTING
    BAPIRETURN = RETURN
  EXCEPTIONS
    OTHERS     = 1.

SELECT SINGLE AUFNR FROM AFKO INTO I_AUFNR
  WHERE AUFNR = BAPI_ORDER_STATUS_IMPORT-ORDERID.
IF SY-SUBRC NE 0.
  CLEAR MESSAGE.
  MESSAGE-MSGTY = 'E'.
  MESSAGE-MSGID = 'Z3'.
  MESSAGE-MSGNO = '000'.
  MESSAGE-MSGV1 = BAPI_ORDER_STATUS_IMPORT-ORDERID.
  IF 1 = 2.
    MESSAGE E000(Z3).
  ENDIF.
ENDIF.
CHECK RETURN IS INITIAL.
CONCATENATE 'OR' BAPI_ORDER_STATUS_IMPORT-ORDERID INTO I_OBJNR.
IF BAPI_ORDER_STATUS_IMPORT-I_EXCLUDEINACTIVE = 'X'.
  SELECT OBJNR STAT INACT FROM JEST INTO TABLE T_BAPISTAT
```

```
WHERE OBJNR = I_OBJNR AND INACT <> 'X'.  
ELSE.  
  SELECT OBJNR STAT INACT FROM JEST INTO TABLE T_BAPISTAT  
    WHERE OBJNR = I_OBJNR .  
ENDIF.  
IF SY-SUBRC <> 0.  
  CLEAR MESSAGE.  
  MESSAGE-MSGTY = 'E'.  
  MESSAGE-MSGID = 'Z3'.  
  MESSAGE-MSGNO = '001'.  
  MESSAGE-MSGV1 = BAPI_ORDER_STATUS_IMPORT-ORDERID.  
IF 1 = 2.  
  MESSAGE E001(Z3).  
ENDIF.  
ENDIF.  
CHECK RETURN IS INITIAL.  
SELECT ISTAT TXT04 TXT30 FROM TJ02T INTO TABLE T_TJ02T FOR ALL ENTRIES  
IN T_BAPISTAT  
  WHERE ISTAT = T_BAPISTAT-STAT AND  
  SPRAS = BAPI_ORDER_STATUS_IMPORT-I_SPRAS.  
SORT T_TJ02T BY ISTAT.  
LOOP AT T_BAPISTAT INTO G_BAPISTAT.  
READ TABLE T_TJ02T WITH KEY ISTAT = G_BAPISTAT-STAT  
  BINARY SEARCH INTO G_TJ02T.  
IF SY-SUBRC = 0.  
  MOVE:G_TJ02T-TXT04 TO G_BAPISTAT-TXT04,  
    G_TJ02T-TXT30 TO G_BAPISTAT-TXT30.  
MODIFY T_BAPISTAT FROM G_BAPISTAT TRANSPORTING TXT04 TXT30.  
ENDIF.  
ENDLOOP.  
ENDFUNCTION.
```

Create a program in SE38

LZBAPISTATUSF01

LZBAPISTATUSTOP

When you try to create a program name starts with L then it gives an message as

“Program names L... are reserved for function group includes”

ignore that and press enter then you can be able to create a new program

Coding for Include program LZBAPISTATUSF01:

```
*&-----*
*& Include          LZBAPISTATUSF01
*&-----*

*&-----*
*& Form SET_RETURN_MESSAGE
*&-----*
* This routine is used for setting the BAPI return message.
* The routine is a standard routine for BAPIs that handles the message
* structure for the BAPIRETURN structure. It has been copied from the
* BAPI Company Code Getlist
*-----*
* -->P_MESSAGE text
* <--P_RETURN text
*-----*

form SET_RETURN_MESSAGE USING VALUE(P_MESSAGE) LIKE MESSAGE
                           CHANGING P_RETURN LIKE BAPIRETURN.
  CHECK NOT MESSAGE IS INITIAL.
  CALL FUNCTION 'BALW_BAPIRETURN_GET'
    EXPORTING
      TYPE           = P_MESSAGE-MSGTY
      CL             = P_MESSAGE-MSGID
      NUMBER        = P_MESSAGE-MSGNO
      PAR1          = P_MESSAGE-MSGV1
      PAR2          = P_MESSAGE-MSGV2
      PAR3          = P_MESSAGE-MSGV3
      PAR4          = P_MESSAGE-MSGV4
      LOG_NO        = ' '
      LOG_MSG_NO    = ' '
    IMPORTING
      BAPIRETURN    = P_RETURN
  EXCEPTIONS
    OTHERS         = 1.

IF SY-SUBRC <> 0.
  * MESSAGE ID SY-MSGID TYPE SY-MSGTY NUMBER SY-MSGNO
  * WITH SY-MSGV1 SY-MSGV2 SY-MSGV3 SY-MSGV4.
ENDIF.

endform.                " SET_RETURN_MESSAGE
```

Save it and activate if it shows any warnings ignore them no problem but do remember that all u create should be under any package only because at last we have to release the BAPI

Coding for LZBAPISTATUSTOP:

```
*FUNCTION-POOL ZBAPISTATUS.                                "MESSAGE-ID ..
FUNCTION-POOL ZBAPISTATUS.                                "MESSAGE-ID Z3
Types:
  begin of Type_tj02t,
    istat like tj02t-istat,
    txt04 like tj02t-txt04,
    txt30 like tj02t-txt30,
  end of type_tj02t.
DATA:
* Declarations for TABLE parameter
  T_BAPISTAT like ZBAPISTAT occurs 0,
  G_BAPISTAT like ZBAPISTAT,
* Table for object texts
  t_tj02t    type type_tj02t occurs 0,
  g_tj02t    type type_tj02t.
* Structure for return messages
DATA:
  BEGIN OF MESSAGE,
    MSGTY LIKE SY-MSGTY,
    MSGID LIKE SY-MSGID,
    MSGNO LIKE SY-MSGNO,
    MSGV1 LIKE SY-MSGV1,
    MSGV2 LIKE SY-MSGV2,
    MSGV3 LIKE SY-MSGV3,
    MSGV4 LIKE SY-MSGV4,
  END OF MESSAGE.

INCLUDE LZBAPISTATUSF01.
* - Subroutines
```

Save it and activate if it shows any warnings ignore them. Don't bother about them.....

Create the API Method Using the BAPI WIZARD:

The BAPI wizard is used to expose the remote function module as a BAPI. The wizard will generate some additional code, so the function module is a valid method of the BOR. This allows the BAPI to be called as a workflow method in addition to be called by an outside program.

Note:

Each function module corresponds to a method in the BOR

Go to the Business Object Builder SWO1.

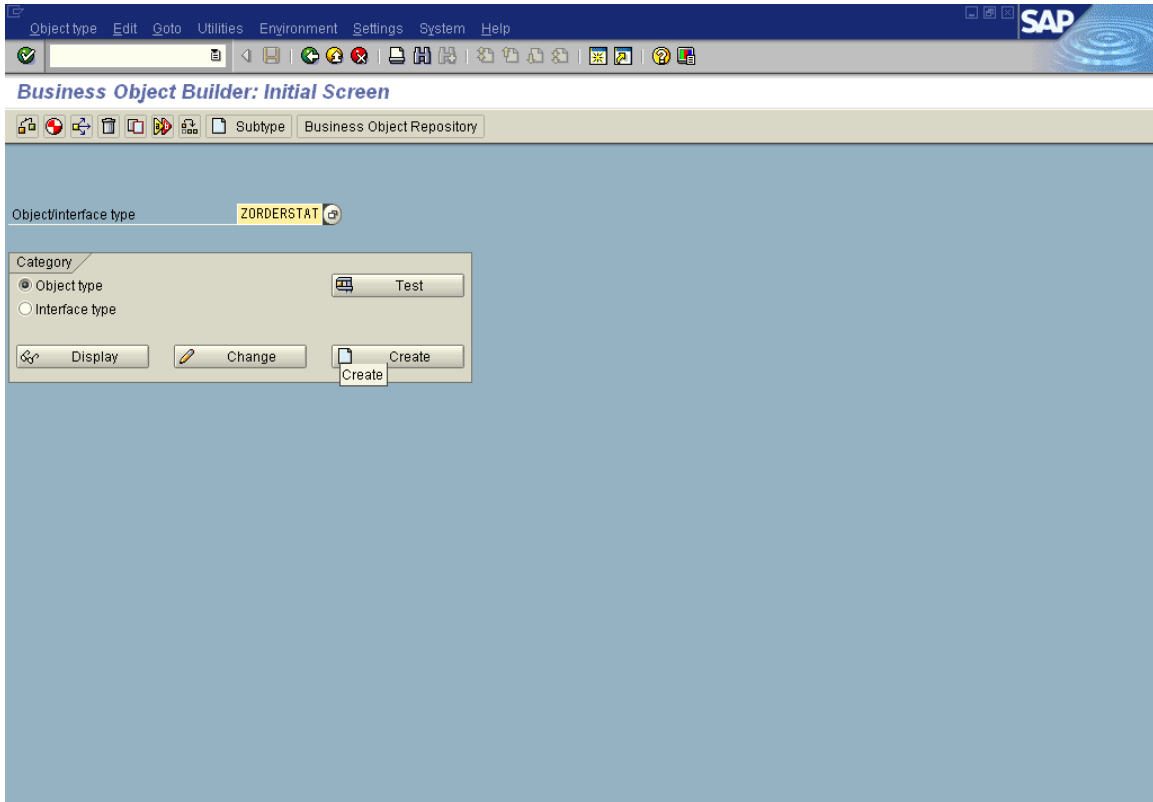
You can either create the new Object type as a subtype of an existing business object or create a new business object from scratch. In this example it would be obvious to create the Object type as a subtype of BUS2005 Production order. However, to illustrate how to create a new Object type from scratch, we will do this.

In the *Object/Interface type* field write the name of the new Business Object: ZORDERSTAT. Press enter and fill in the additional fields necessary to create the object type.

Supertype: Not relevant because we are creating our object from scratch

Program. This is the name of the program where the wizard generates code for the Object type, NOT the function module we created earlier. The program name must not be the name of an existing program.

Goto the Tcode *SWO1* in the Tcode its “O” Not zero. To create new Business Object give the Object name *ZORDERSTAT* and then press F5 or press on the button create



and fill with the following values

Supertype → not necessary for our present requirement

Object Type → *ZORDERSTAT*

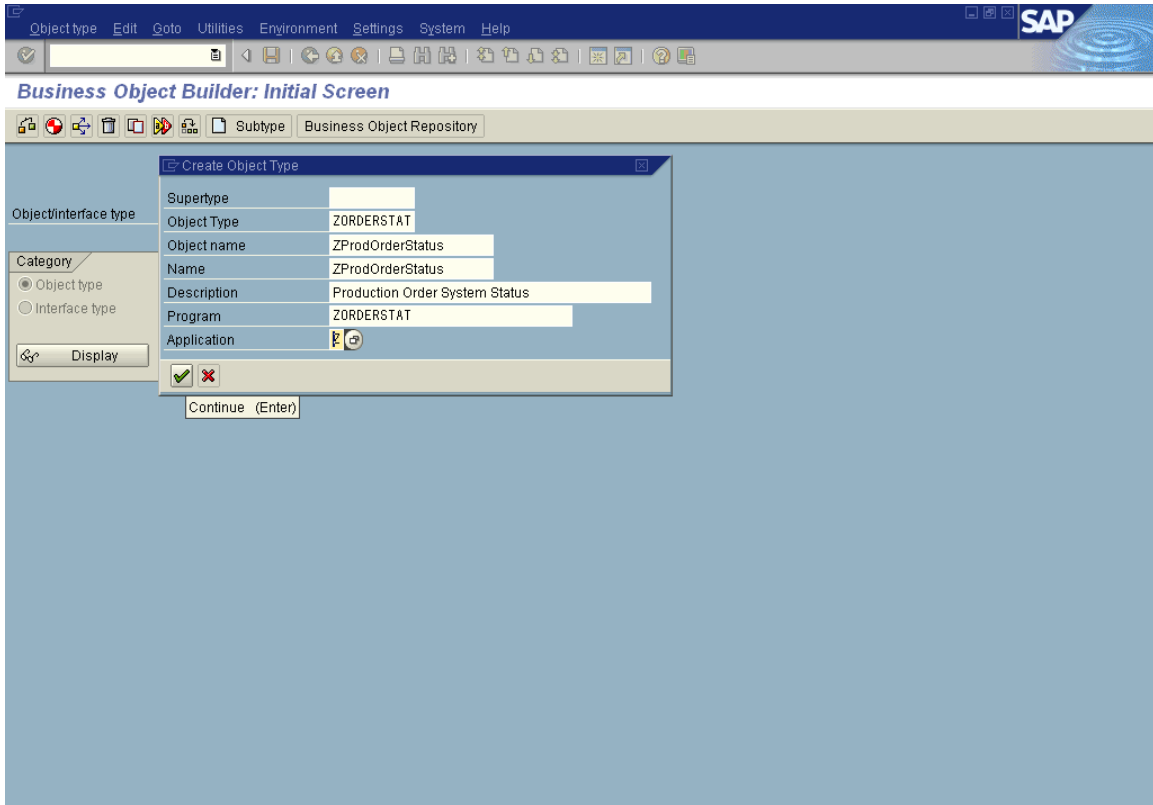
Object name → *ZProdOrderStatus*

Name → *ZProdOrderStatus*

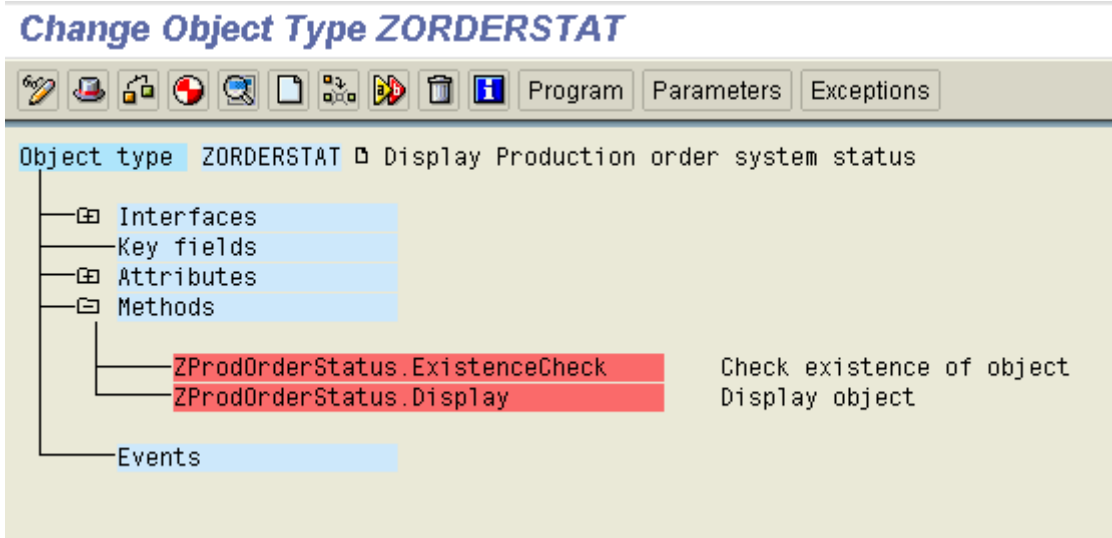
Description → *Production Order System Status*

Program → *ZORDERSTAT*


Application → *Z*

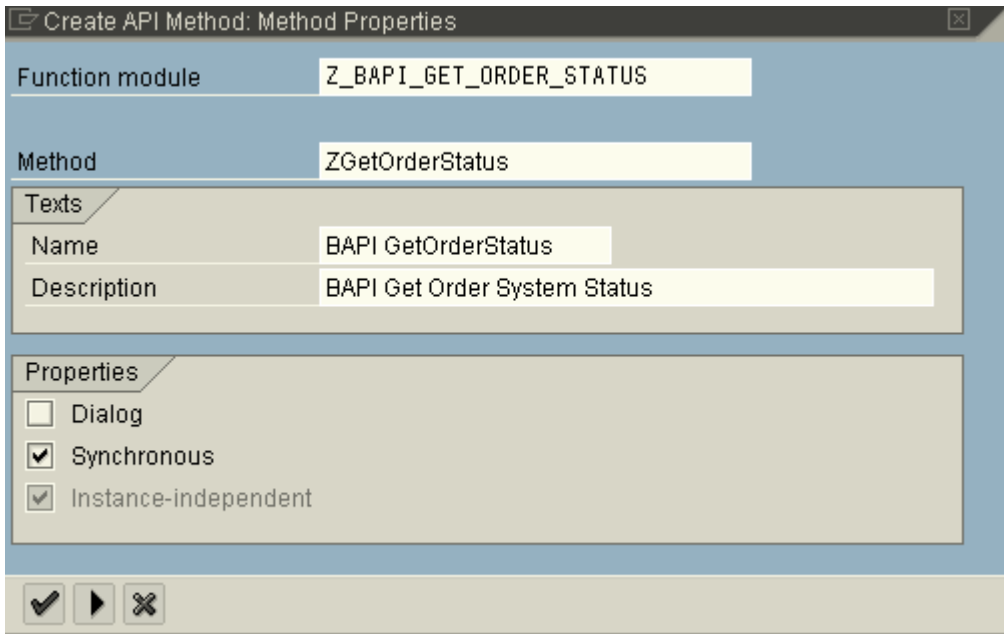


Press enter and create the new business object. Note that when you create the business object a standard interface, an attribute `ObjectType` and the methods `ExistenceCheck` and `Display` are automatically generated. These cannot be changed. So Assign a package while creating only not temporary(\$tmp)



The next step is to add the **Z_BAPI_GET_ORDER_STATUS** method to the business object.

Select **Utilities -> API methods -> Add method** and write the name of the function module in the dialog box. Next the dialog box shown below will be shown. This is the start screen of the BAPI wizard. Proceed with wizard by pressing the  (Next) button. But don't press Enter button



Create API Method: Method Properties

Function module

Method

Texts

Name

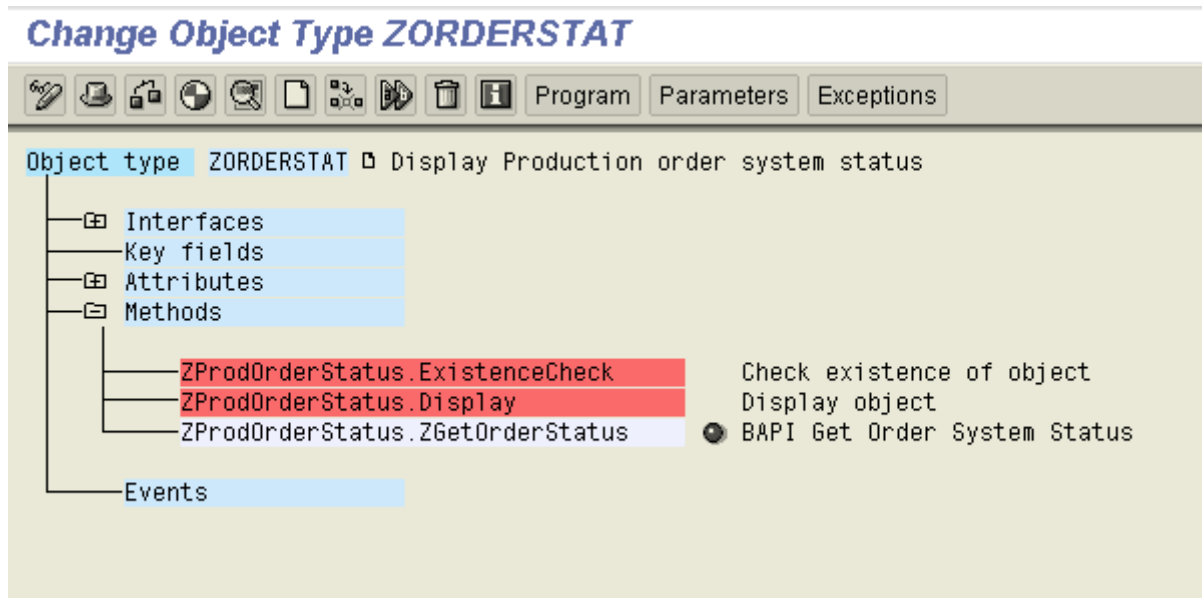
Description

Properties

Dialog

Synchronous

Instance-independent



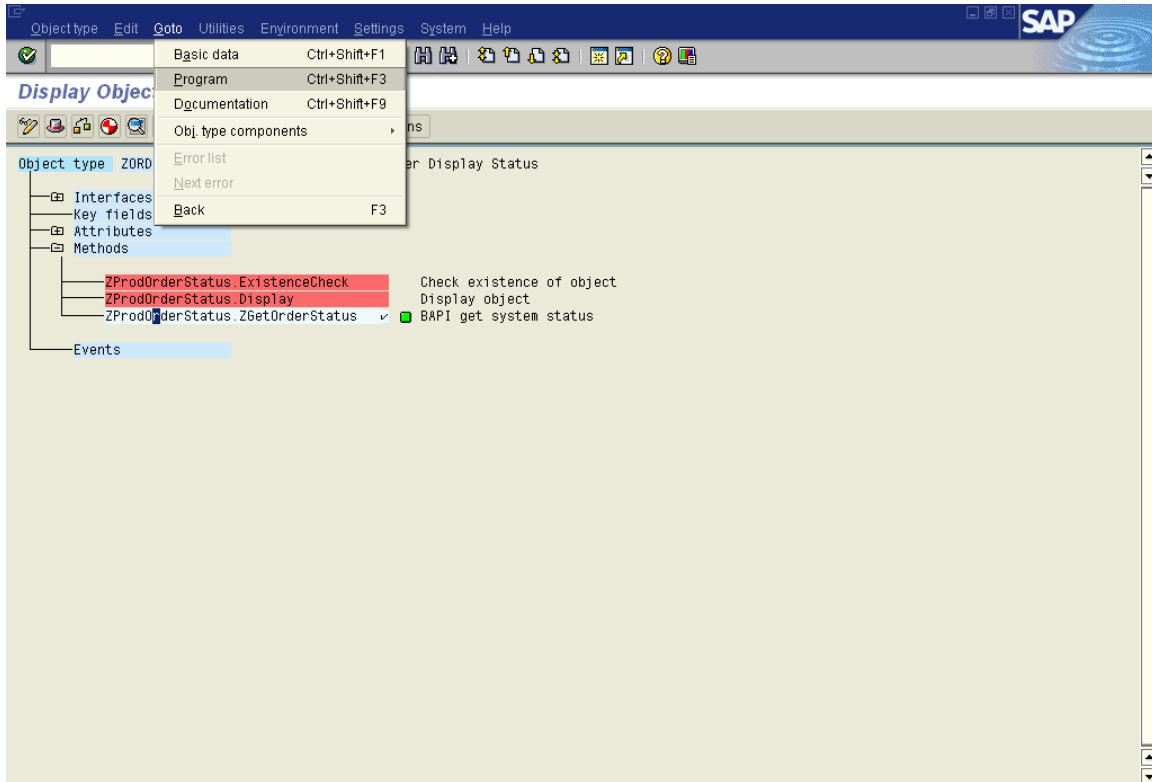
You can double-click on the method to see its properties. To use the business object you must change the Object type status to Implemented.

Use menu *Edit->Change releases status->Object type->To implemented.*

Now you can test the object (Press F8).

Note that the BAPI wizard has added a wrapper class for the function module so it can be used as method in the business object.

Choose menu *Goto->Program* to display the program:



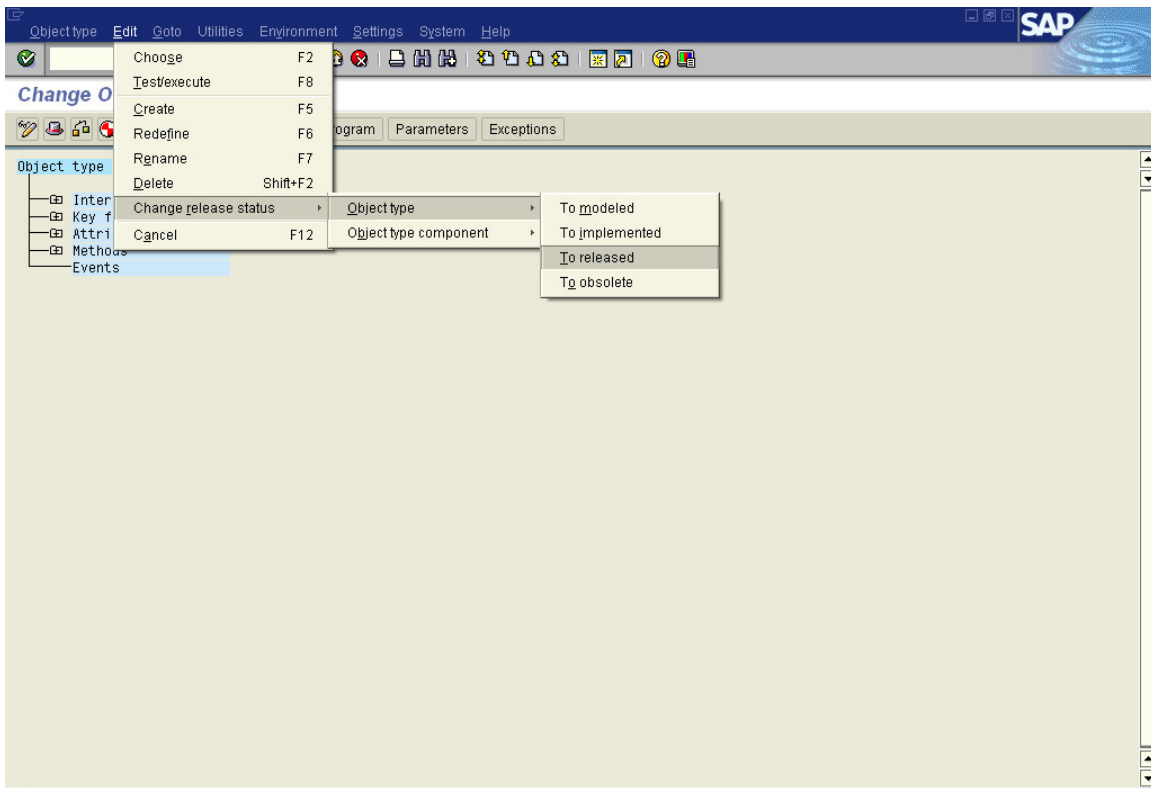
This is automatically generated code there is no need to perform any modifications in that coding

```
*****      Implementation of object type ZORDERSTAT      *****
INCLUDE <OBJECT>.
BEGIN_DATA OBJECT. " Do not change.. DATA is generated
* only private members may be inserted into structure private
DATA:
" begin of private,
" to declare private attributes remove comments and
" insert private attributes here ...
" end of private,
  KEY LIKE SWOTOBJID-OBJKEY.
END_DATA OBJECT. " Do not change.. DATA is generated

BEGIN_METHOD ZGETORDERSTATUS CHANGING CONTAINER.
DATA:
  BAPIORDERSTATUSIMPORT LIKE ZBAPI_ORDER_STATUS_IMPORT,
  RETURN LIKE BAPIRETURN,
  TBAPISTAT LIKE ZBAPISTAT OCCURS 0.
SWC_GET_ELEMENT CONTAINER 'BapiOrderStatusImport'
```

```
BAPIORDERSTATUSIMPORT.  
SWC_GET_TABLE CONTAINER 'TBapistat' TBAPISTAT.  
CALL FUNCTION 'Z_BAPI_GET_ORDER_STATUS'  
EXPORTING  
  BAPI_ORDER_STATUS_IMPORT = BAPIORDERSTATUSIMPORT  
IMPORTING  
  RETURN = RETURN  
TABLES  
  T_BAPISTAT = TBAPISTAT  
EXCEPTIONS  
  OTHERS = 01.  
CASE SY-SUBRC.  
  WHEN 0.      " OK  
  WHEN OTHERS. " to be implemented  
ENDCASE.  
SWC_SET_ELEMENT CONTAINER 'Return' RETURN.  
SWC_SET_TABLE CONTAINER 'TBapistat' TBAPISTAT.  
END_METHOD.
```

Now release the BAPI



When the Business object has been checked and the documentation created, the following steps must be carried out:

*** Release the BAPI function module (in the Function Builder).**

*** Release the business object type**

(in the BOR ObjectType -> Change release status to -> Implemented).

*** Release the BAPI as a method in the BOR (Release the methods you has created
- Set the cursor on the method then**

Edit -> Change release status -> Object type component -> To released)

- **For potential write BAPIs: Release the IDoc and its segments**

Goto the Tcode *BAPI*

