Connect for SAP® (Classic)

Demo Guide

1 Structure of Connect for SAP$^®$ Demo Applications

All demo applications for Connect for SAP$^®$ are located in the following directory:

```
SAPx\Demo
```

The next table lists functionality covered by the Connect for SAP$^®$ demo applications:

Area	Location
Client demos. Describes Connect for SAP$^®$ working as client.	SAPx\Demo\Client\*
Server demos. Describes Connect for SAP$^®$ working as external SAP server.	SAPx\Demo\Server\*

2 Connect for SAP® Client Demos

2.1 Simple Connect to a SAP Server and Call of RFC_PING

This demo allows connecting to a SAP server and calling a remote function RFC_PING.

The connection can be established using one of the next ways:

1. Use an alias file with connection parameters. For additional details regarding the connection aliases, see the topic 5.1 “Creating and Maintaining aliases” of the “Getting Started” (see Locations of Connect for SAP® Documents).
2. Specify the connection parameters manually.

Location

`SAPx\Demo\Client\01_ConnectAndPing`

Application area

The functionality of the demo is implemented mainly by two VCL components: TSAPxRFCvClientConnectionGS and TSAPxRFCvFunctionGS.

The main properties of TSAPxRFCvClientConnectionGS are shown in the next table:

Property	Value	Description
AliasName	<An alias name>	Define an alias to be used to initialize the connection.
Active	True	Establish a connection.
LoginPrompt	True	Display a login dialog on the stage of establishing a connection.

The main properties of TSAPxRFCvFunctionGS are shown in the next table:

Property	Value	Description
Connection	FCConnection	Define the connection object to be used by the function.
ObjName	RFC_PING	Define the function calls RFC_PING on a SAP server.

Execution

Depending on the chosen way of specifying connection parameters, the user sets further either a certain alias or connection parameters and presses the Connect button. After that, the user can call RFC_PING by pressing the corresponding button on the demo form.

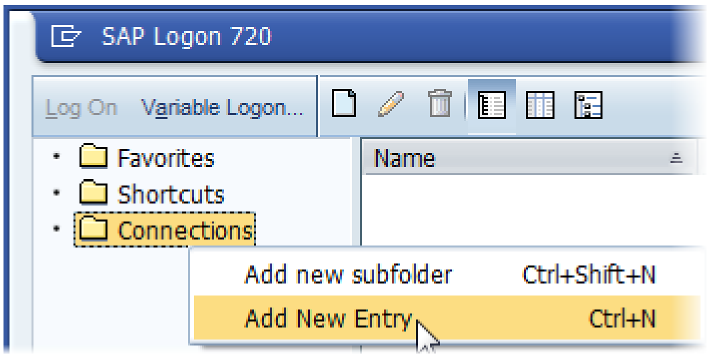
2.2 Test a RFC Function via SAP GUI

Any RFC functions stored on a SAP server can also be executed via the SAP GUI application. As an example, we show how to call an RFC function BAPI_PO_GETDETAIL (Purchase Order Details). This is the entrance point to describe the later example using Connect for SAP to call the same RFCs.

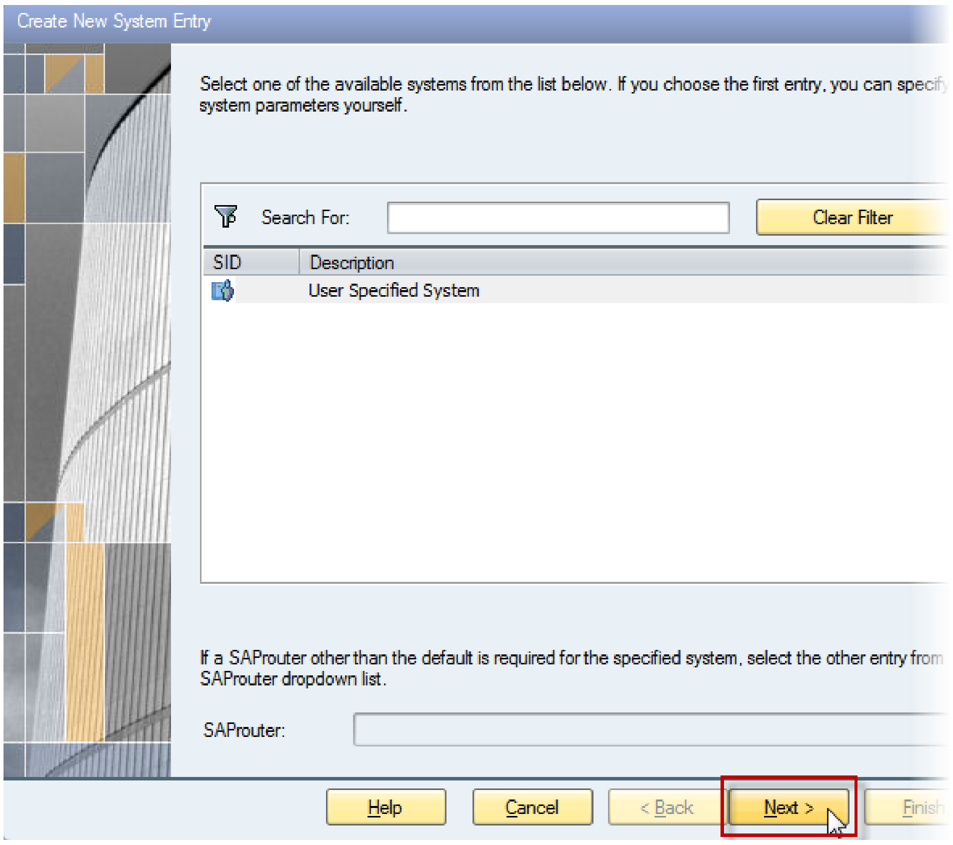
Execution

To prepare the SAP server side and execute the RFC module the following steps should be done

1. Launch the SAP GUI application (normally it's located in `%Program Files (x86)%\SAP\FrontEnd\SAPgui\saplogon.exe`).
2. Create a new connection to the SAP server by clicking Add New Entry under Connections.



3. Click Next on the shown wizard to navigate to a connection configuration page.


Create New System Entry

Select one of the available systems from the list below. If you choose the first entry, you can specify system parameters yourself.

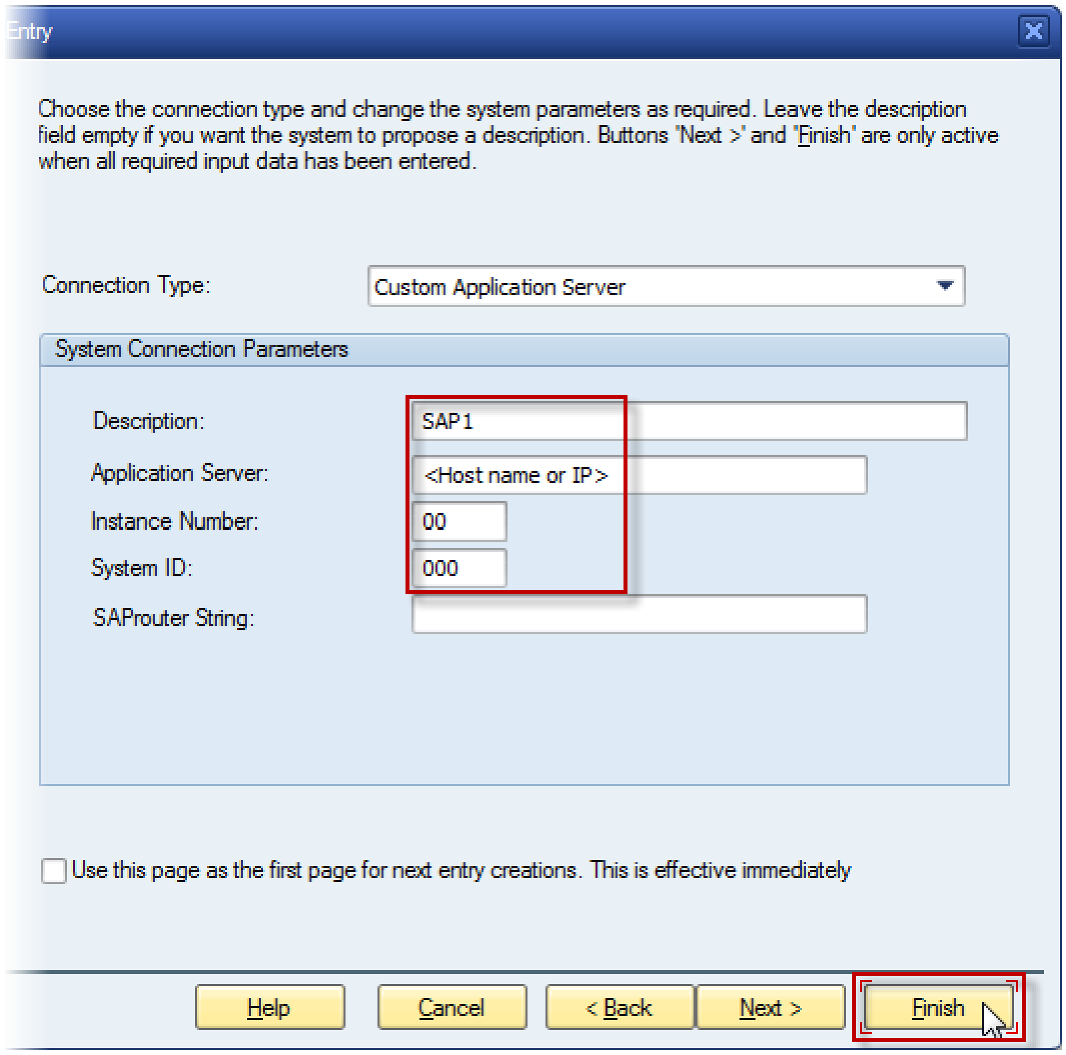
Search For:

SID	Description
	User Specified System

If a SAProuter other than the default is required for the specified system, select the other entry from SAProuter dropdown list.

SAProuter:

4. On the connection configuration page, set parameters as shown on the next screenshot. And click the Finish.



Entry

Choose the connection type and change the system parameters as required. Leave the description field empty if you want the system to propose a description. Buttons 'Next >' and 'Finish' are only active when all required input data has been entered.

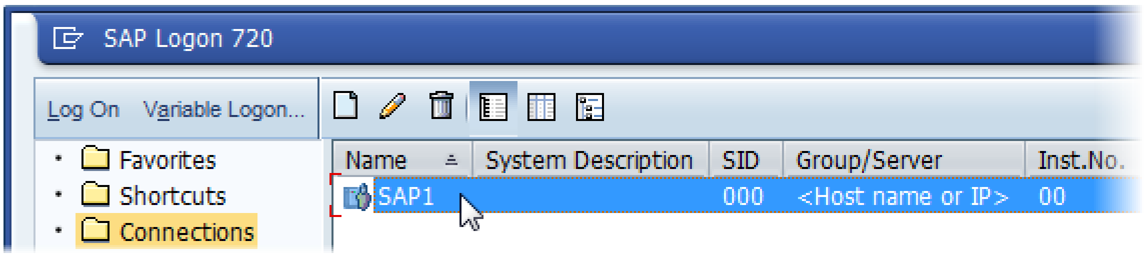
Connection Type: Custom Application Server

System Connection Parameters

Description:	SAP1
Application Server:	<Host name or IP>
Instance Number:	00
System ID:	000
SAProuter String:	

Use this page as the first page for next entry creations. This is effective immediately

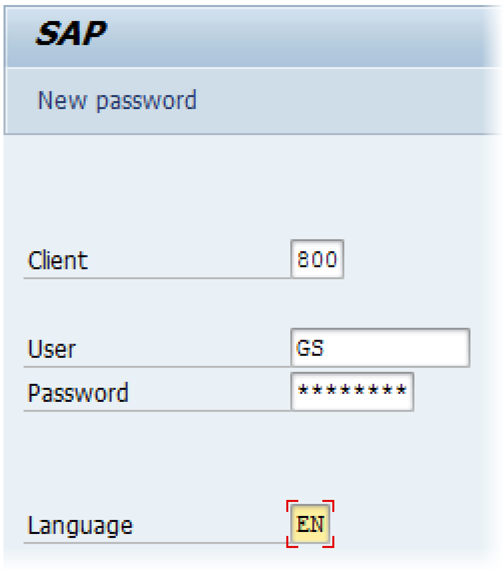
5. Log on the SAP server using the created connection.



SAP Logon 720

Name	System Description	SID	Group/Server	Inst.No.
SAP1		000	<Host name or IP>	00

6. Input your client parameters on the logon window appeared (an example is shown on the next screenshot) and press Enter.

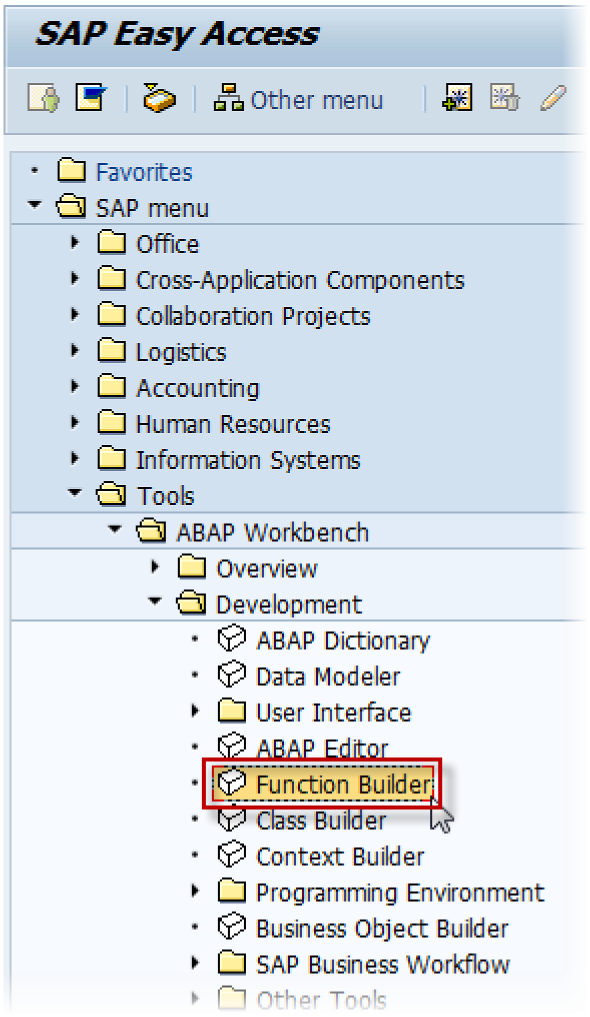


SAP

New password

Client	800
User	GS
Password	********
Language	EN

7. To search for BAPI_PO_GETDETAIL on the SAP server, navigate to the item Function Builder of the SAP Menu and double click on the item to run it.

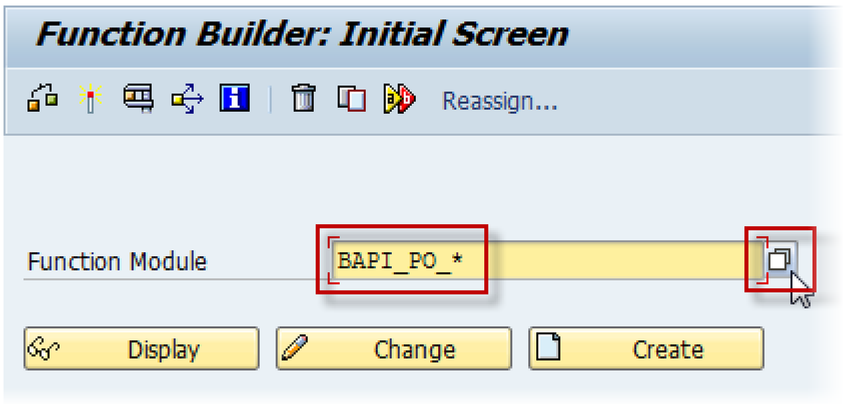


SAP Easy Access

Other menu

- Favorites
- SAP menu
  - Office
  - Cross-Application Components
  - Collaboration Projects
  - Logistics
  - Accounting
  - Human Resources
  - Information Systems
  - Tools
    - ABAP Workbench
      - Overview
      - Development
        - ABAP Dictionary
        - Data Modeler
        - User Interface
        - ABAP Editor
        - Function Builder
        - Class Builder
        - Context Builder
        - Programming Environment
        - Business Object Builder
        - SAP Business Workflow
        - Other Tools

8. On the Initial Screen of Function Builder, set a search mask for required function. For example, for the function BAPI_PO_GETDETAIL, it could be "BAPI_PO_*". Then press the search button as shown on the next screenshot.

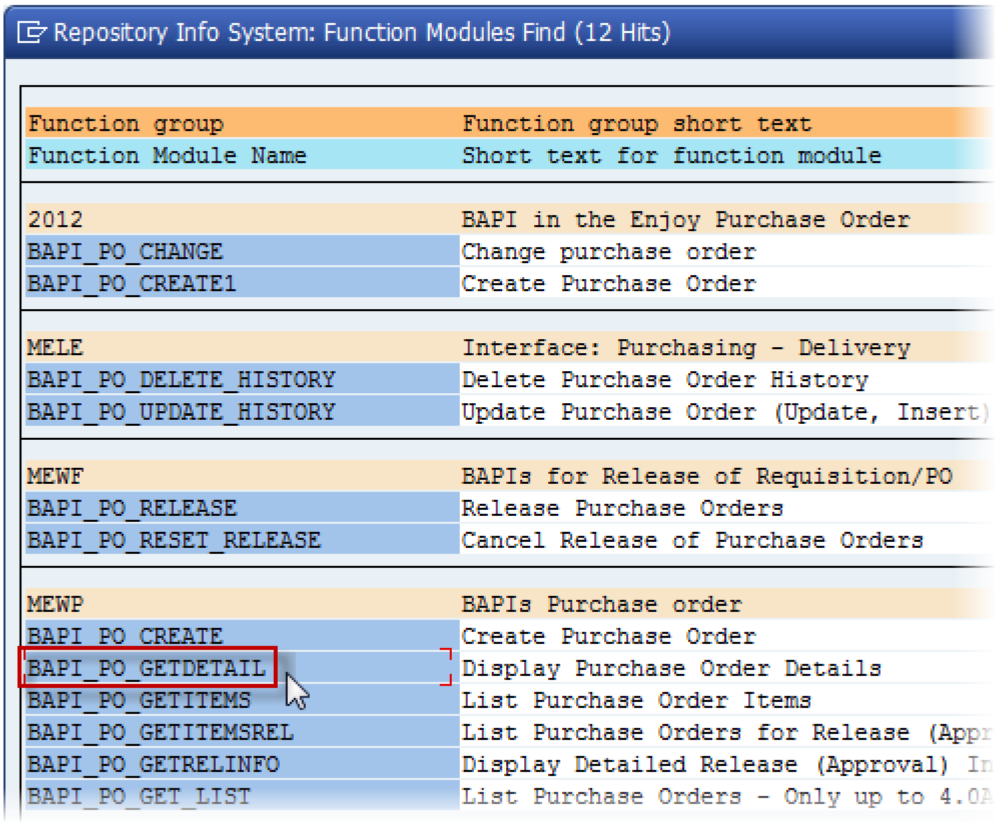


Function Builder: Initial Screen

Reassign...

Function Module: BAPI_PO_*

9. Find the required function in the displayed list and double click to insert the function in Function Builder.

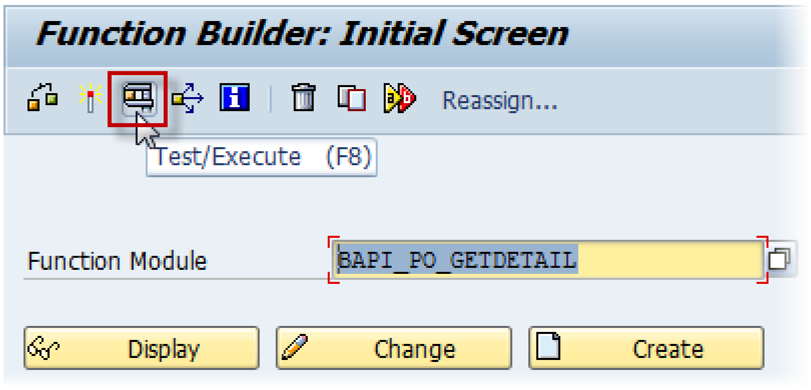


Repository Info System: Function Modules Find (12 Hits)

Function group	Function group short text
Function Module Name	Short text for function module
2012	BAPI in the Enjoy Purchase Order
BAPI_PO_CHANGE	Change purchase order
BAPI_PO_CREATE1	Create Purchase Order
MELE	Interface: Purchasing - Delivery
BAPI_PO_DELETE_HISTORY	Delete Purchase Order History
BAPI_PO_UPDATE_HISTORY	Update Purchase Order (Update, Insert)
MEWF	BAPIs for Release of Requisition/PO
BAPI_PO_RELEASE	Release Purchase Orders
BAPI_PO_RESET_RELEASE	Cancel Release of Purchase Orders
MEWP	BAPIs Purchase order
BAPI_PO_CREATE	Create Purchase Order
BAPI_PO_GETDETAIL	Display Purchase Order Details
BAPI_PO_GETITEMS	List Purchase Order Items
BAPI_PO_GETITEMSREL	List Purchase Orders for Release (Appr
BAPI_PO_GETRELINFO	Display Detailed Release (Approval) In
BAPI_PO_GET_LIST	List Purchase Orders - Only up to 4.0A

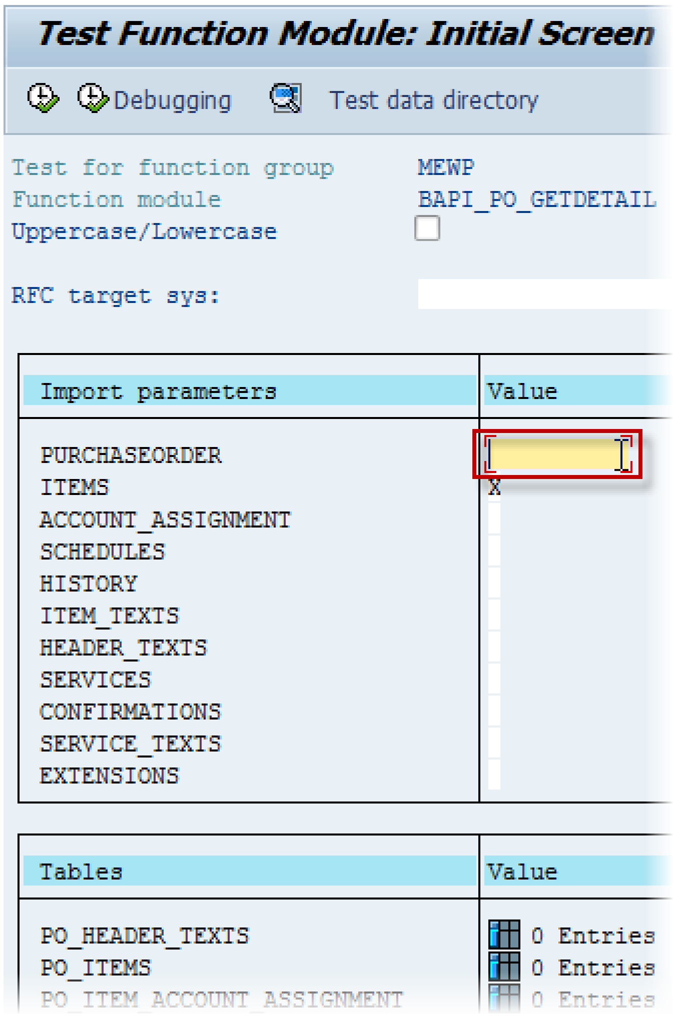
10. Call of BAPI_PO_GETDETAIL.

11. After the function is found and specified in the Function Builder - press Test/Execute to activate Test Function Module.



Function Module BAPI_PO_GETDETAIL

12. The window shows a list of import parameters and tables for the function. The function BAPI_PO_GETDETAIL has just one non-optional parameter PURCHASEORDER. The next step describes how to find any Purchase Orders stored on the SAP server.

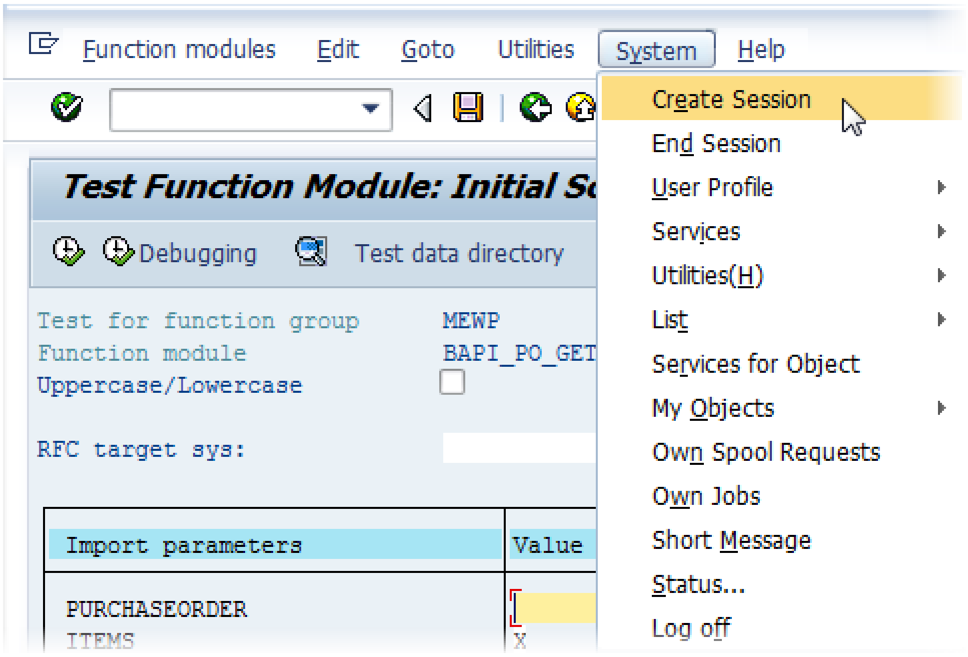


```
Test for function group     MEWP
Function module             BAPI_PO_GETDETAIL
Uppercase/Lowercase         ☐

RFC target sys:
```

Import parameters	Value
PURCHASEORDER	
ITEMS	X
ACCOUNT_ASSIGNMENT	
SCHEDULES	
HISTORY	
ITEM_TEXTS	
HEADER_TEXTS	
SERVICES	
CONFIRMATIONS	
SERVICE_TEXTS	
EXTENSIONS	

Tables	Value
PO_HEADER_TEXTS	0 Entries
PO_ITEMS	0 Entries
PO_ITEM_ACCOUNT_ASSIGNMENT	0 Entries

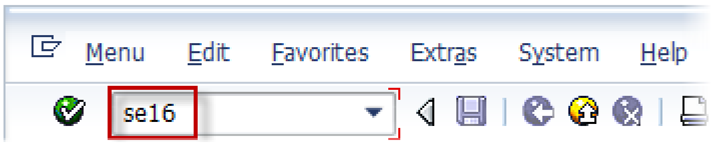
13. To get a list of Purchase Orders, create a new session as shown on the next screenshot.



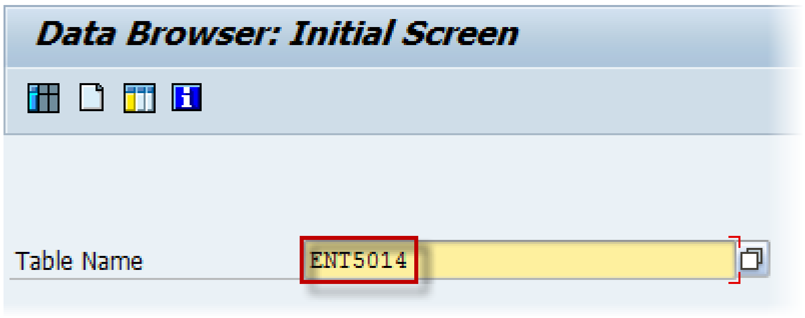
Test Function Module: Initial Sc

```
Test for function group    MEWP
Function module            BAPI_PO_GET
Uppercase/Lowercase

RFC target sys:

Import parameters                    Value
PURCHASEORDER
ITEMS                                X
```

14. To browse data on the SAP server, input the short code SE16 (which associated with Data Browser) into the search box in the new session window. And then press Enter.



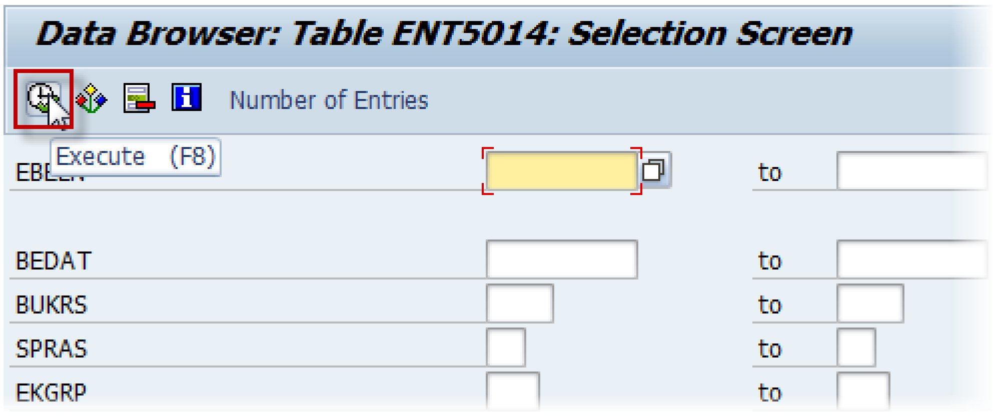
15. Specify ENT5014 as a value of Table Name in the appeared Data Browser and press Enter.



Data Browser: Initial Screen

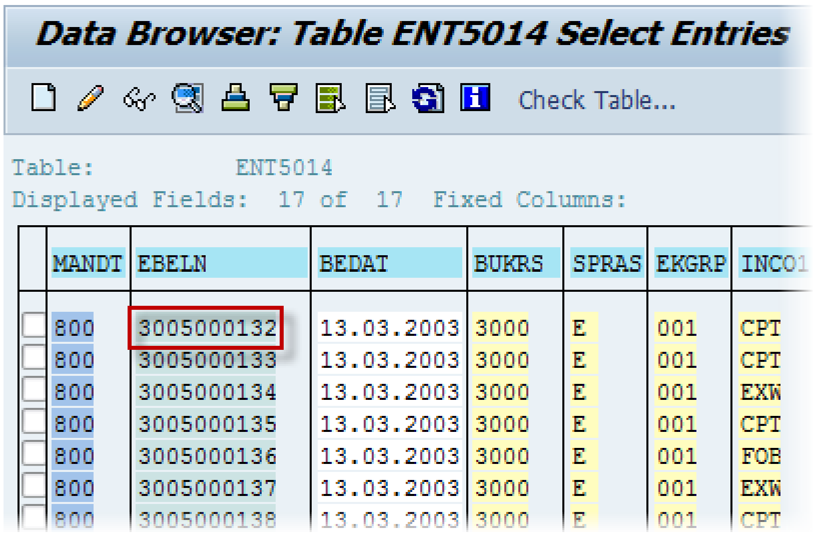
Table Name ENT5014

16. Press Execute in the appeared Selection Screen of the table. No filters required to see all entries of the table.



EBELN		to	
BEDAT		to	
BUKRS		to	
SPRAS		to	
EKGRP		to	

17. The window shows the entries of the table ENT5014. For example, EBELN of the first row will be chosen as a parameter for BAPI_PO_GETDETAIL. Just remember the number for a usage in the next step.

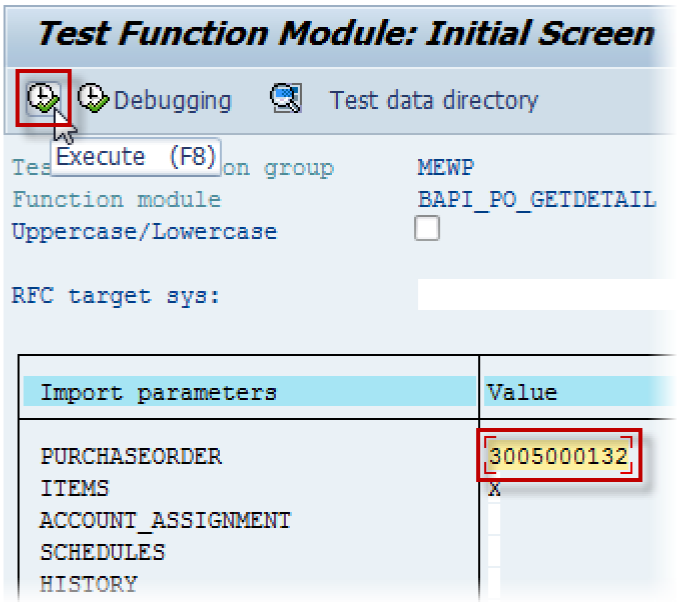


```
Table:          ENT5014
Displayed Fields:  17 of  17  Fixed Columns:
```

MANDT	EBELN	BEDAT	BUKRS	SPRAS	EKGRP	INCO1
800	3005000132	13.03.2003	3000	E	001	CPT
800	3005000133	13.03.2003	3000	E	001	CPT
800	3005000134	13.03.2003	3000	E	001	EXW
800	3005000135	13.03.2003	3000	E	001	CPT
800	3005000136	13.03.2003	3000	E	001	FOB
800	3005000137	13.03.2003	3000	E	001	EXW
800	3005000138	13.03.2003	3000	E	001	CPT

18. Switch to the window with the first session (where the actions were performing before the step **13**)

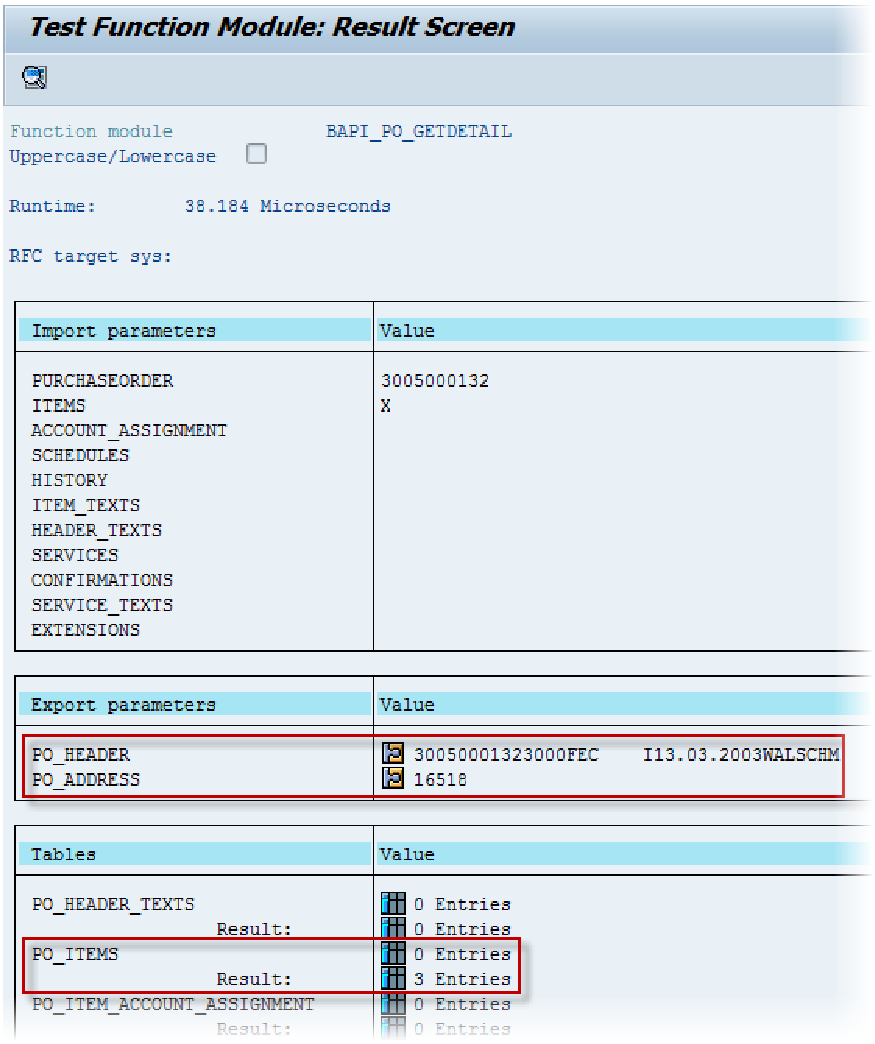
19. Input the chosen EBELN number into the PURCHASEORDER edit box and press Execute.



Test function group	MEWP
Function module	BAPI_PO_GETDETAIL
Uppercase/Lowercase	☐
RFC target sys:	

Import parameters	Value
PURCHASEORDER	3005000132
ITEMS	X
ACCOUNT_ASSIGNMENT	
SCHEDULES	
HISTORY	

20. The appeared Result Screen shows results of the execution: export parameters and tables.



Test Function Module: Result Screen

```
Function module          BAPI_PO_GETDETAIL
Uppercase/Lowercase      ☐

Runtime:        38.184 Microseconds

RFC target sys:
```

Import parameters	Value
PURCHASEORDER	3005000132
ITEMS	X
ACCOUNT_ASSIGNMENT	
SCHEDULES	
HISTORY	
ITEM_TEXTS	
HEADER_TEXTS	
SERVICES	
CONFIRMATIONS	
SERVICE_TEXTS	
EXTENSIONS	

Export parameters	Value
PO_HEADER	30050001323000FEC I13.03.2003WALSCHM
PO_ADDRESS	16518

Tables	Value
PO_HEADER_TEXTS	0 Entries
Result:	0 Entries
PO_ITEMS	0 Entries
Result:	3 Entries
PO_ITEM_ACCOUNT_ASSIGNMENT	0 Entries
Result:	0 Entries

2.3 Call BAPI_PO_GETDETAIL

This demo shows connecting to a SAP server and calling BAPI_PO_GETDETAIL with an initialization of parameters from the code (without usage of the Connect for SAP® VCL components). The results of the execution (parameters and tables) are shown in a log. The main goal of the demo is to perform the same actions and to get the same results as described by the steps **19** and **20** of the topic **2.2**.

Location

`SAPx\Demo\Client\02_RfcCall`

Application area

The functionality of the demo requires the next parameters (TSAPxRFCParameterGS) and tables (TSAPxRFCTableGS) are initialized before the call BAPI_PO_GETDETAIL.

The main settings for input parameters are shown in the next table:

Property	Value	Description
Name	PURCHASEORDER	Defines the parameter represents Purchase Order for the function.
AsString	3005000132	Identifier of Purchase Order represented by a string.
ParameterType	ptImportGS	Defines the parameter is used as an input.

The main settings of output parameters are shown in the next table:

Property	Value	Description
PO_HEADER parameter		
Name	PO_HEADER	Defines the parameter represents a Header of the returned details.
ParameterType	ptExportGS	Defines the parameter is used as an output.
DataType	dtStructureGS	Defines the parameters has structure-like data (has subfields)
StructName	BAPIEKKOL	Defines the parameter data has structure as the specified SAP structure.
PO_ADDRESS parameter		
Name	PO_ADDRESS	Defines the parameter represents an Address of the returned details.
ParameterType	ptExportGS	Defines the parameter is used as an output.
DataType	dtStructureGS	Defines the parameters has structure-like data (has subfields)
StructName	BAPIADDRESS	Defines the parameter data has structure as the specified SAP structure.

The main settings for tables are shown in the next table:

Property	Value	Description
Name	PO_ITEMS	Defines the table represents items returned for the specified PURCHASEORDER.
StructName	BAPIEKPO	Defines the table has the same fields as the specified SAP structure.

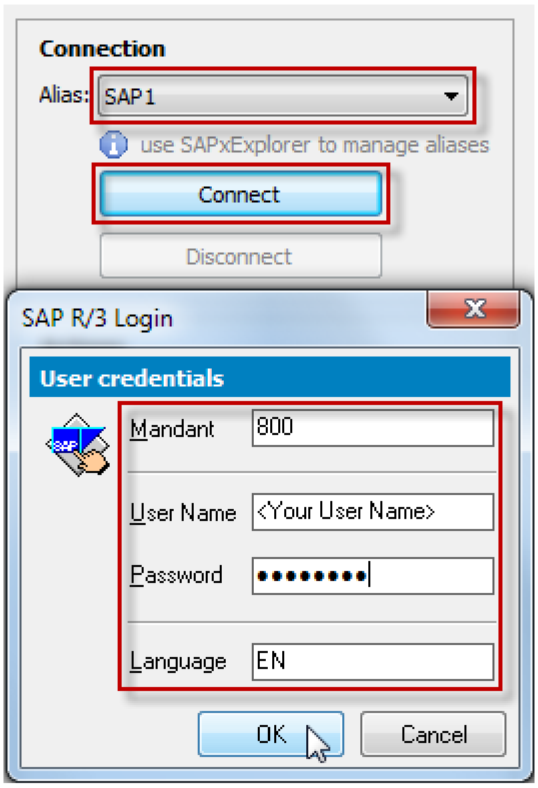
Visualization of the parameters and the tables is performed by writing values of their SubFields to the Output log as shown on the next listing:

Listing 1: Source code of writing fields to Output

```
procedure TfrmMain.WriteFields(AFields: TSAPxRFCFieldsListGS; AOutput: TStrings;
AIndent: Integer);
const
  C_DateToStrTemplate: string = 'yyyy.mm.dd';
  C_TimeToStrTemplate: string = 'hh:nn:ss:zzz';
var
  i: Integer;
  oField: TSAPxRFCFieldGS;
  sValue, sIndent: string;
begin
  sIndent := Indent(AIndent);
  if AFields.Count = 0 then begin
    AOutput.Add(sIndent + 'No fields');
    Exit;
  end;

  for i := 0 to AFields.Count - 1 do begin
    oField := AFields[i];
    case oField.DataType of
      dtDateGS: sValue := FormatDateTime(C_DateToStrTemplate, oField.AsDate);
      dtTimeGS: sValue := FormatDateTime(C_TimeToStrTemplate, oField.AsTime)
      else      sValue := oField.AsString;
    end;
    AOutput.Add(sIndent + Format(C_NameValueTemplateEx, [oField.Name, sValue]));
  end;
end;
```

Execution

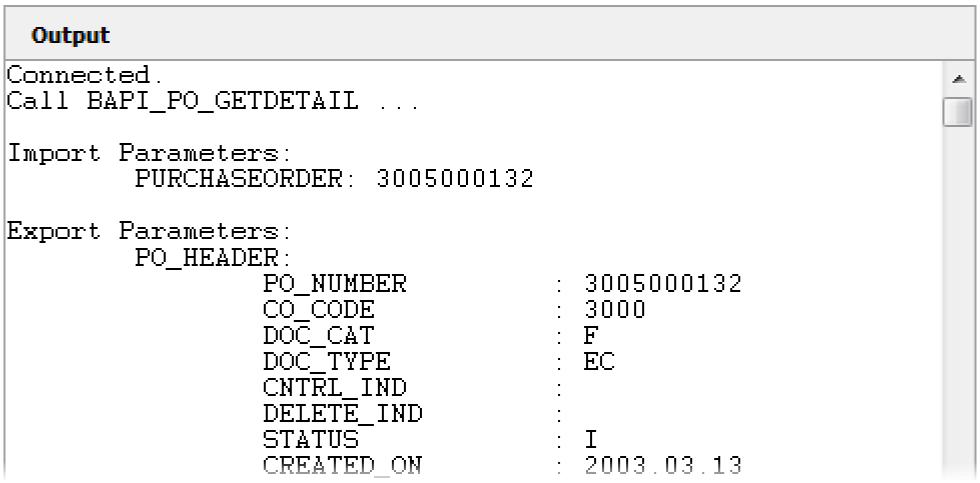
The user establishes a connection to a SAP server by choosing a desired alias for the connection, pressing Connect and specifying the user credentials in the appeared login dialog.



Mandant: 800
User Name: <Your User Name>
Password: ••••••••
Language: EN

After the connection established, the user presses Execute on the main form.

The processes of connecting, executing as well as import/export parameters and tables are written to the Output.



```
Connected.
Call BAPI_PO_GETDETAIL ...

Import Parameters:
        PURCHASEORDER: 3005000132

Export Parameters:
        PO_HEADER:
                PO_NUMBER          : 3005000132
                CO_CODE            : 3000
                DOC_CAT            : F
                DOC_TYPE           : EC
                CNTRL_IND          :
                DELETE_IND         :
                STATUS             : I
                CREATED_ON         : 2003.03.13
```

2.4 Call BAPI_PO_GETDETAIL with a generated Function Wrappers

This demo shows calling of BAPI_PO_GETDETAIL via a wrapper generated by the Connect for SAP® Explorer tool. The results of the execution (parameters and tables) are shown in a log.

Location

SAPx\Demo\Client\03_RfcWrapper

Application area

For implementation of the functionality, the demo includes a file containing wrappers for the function, its parameters and tables. These wrappers are generated by the Connect for SAP® Explorer tool as described in the topic 5.4 "Generating wrapping code for RFC function module of the "Getting Started" (see Locations of Connect for SAP® Documents).

The main goal of wrappers is to provide a static binding (which requires less time) of Client data types with ones defined on SAP server.

The main work is focused around the function wrapper TSAPxRFCBAPI_PO_GETDETAILFuncGS (properties PURCHASEORDER and ITEMS are generated by the wrapper):

The main input properties of TSAPxRFCBAPI_PO_GETDETAILFuncGS are shown in the next table:

Property	Value	Description
Connection	FCConnection.RfcConnection	Sets a connection for the function.
PURCHASEORDER	3005000132	Sets the identifier of Purchase Order.
ITEMS	X	Defines that items related with the Purchase Order will be added to the table PO_ITEMS. Note: to prevent filling of the table – set the property to ' ' (Space) or '' (Empty).

After the execution, the Demo shows the next updated properties of TSAPxRFCBAPI_PO_GETDETAILFuncGS into the log:

The main output properties of TSAPxRFCBAPI_PO_GETDETAILFuncGS are shown in the next table:

Property	Description
PO_HEADER	A wrapper structure representing a Header.
PO_ADDRESS	A wrapper structure representing an Address.
PO_ITEMS	A wrapper table representing Items related with the Purchase Order.

Execution

The user establishes a connection to a SAP server by choosing a desired alias for the connection, pressing Connect and specifying the user credentials in the appeared login dialog.

After the connection established, the user presses Execute on the main form.

The processes of connecting, executing as well as import/export parameters and tables are written to the Output.

2.5 Create an own RFC

Before, it was shown in the topic **2.2** how to call existing functions on a SAP server. However, the user can create an own RFC functions as well. The process is step by step described in the topic **4.3**.

2.6 Multithread Calls

This sample demonstrates how several functions being called from different threads can share the same client connection.

Location

`SAPx\Demo\Client\04_Multithread`

SAP system area

We need to create only one custom function module on a SAP server side. This module Z_SAPX_CALL_SLEEP has neither import nor export parameters; all what it does is to "sleep" for 10 seconds. You may find an ABAP source code of the module in `SAPx\Demo\Client\04_Multithread\fMain.pas` unit. The information on creating an ABAP functional module is in the topic How to define and execute an RFC Function Module [SE37]. Another functional module being used by the demo is the standard module RFC_PING that normally already exists in SAP systems.

Application area

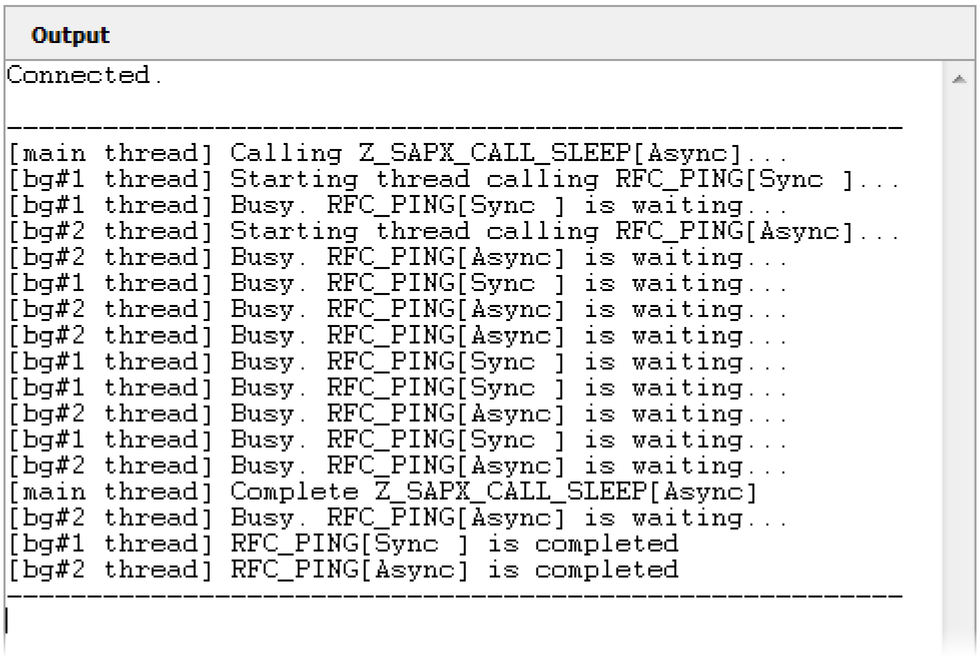
In the application there are three function components of TSAPxRFCvFunctionGS type, which are connected with the same client connection component TSAPxRFCvClientConnectionGS. The next table specifies values of key properties of the components with names FCFunctionSleepAsync, FCFunctionPingSync and FCFunctionPingAsync.

Property	Value	Description
FCFunctionSleepAsync		
Async	True	To be called asynchronously in the main thread
ObjName	Z_SAPX_CALL_SLEEP	
FCFunctionPingSync		
Async	False	To be called asynchronously in the background thread
ObjName	RFC_PING	
FCFunctionPingAsync		
Async	True	To be called synchronously in the background thread
ObjName	RFC_PING	

The scenario supposes that FCFunctionPingSync and FCFunctionPingAsync are called in each own background thread. And FCFunctionSleepAsync is called from the main thread. The RFC Client connection can handle only one function call at the moment. It means that the next call can be performed within a single client connection as soon as the previous one is completed.

Execution

Run the demo and connect to the target SAP system. After pressing Execute, Output log reflects a sequence of calls (see below).


Output

```
Connected.

--------------------------------------------------------
[main thread] Calling Z_SAPX_CALL_SLEEP[Async]...
[bg#1 thread] Starting thread calling RFC_PING[Sync ]...
[bg#1 thread] Busy. RFC_PING[Sync ] is waiting...
[bg#2 thread] Starting thread calling RFC_PING[Async]...
[bg#2 thread] Busy. RFC_PING[Async] is waiting...
[bg#1 thread] Busy. RFC_PING[Sync ] is waiting...
[bg#2 thread] Busy. RFC_PING[Async] is waiting...
[bg#2 thread] Busy. RFC_PING[Async] is waiting...
[bg#1 thread] Busy. RFC_PING[Sync ] is waiting...
[bg#1 thread] Busy. RFC_PING[Sync ] is waiting...
[bg#2 thread] Busy. RFC_PING[Async] is waiting...
[bg#1 thread] Busy. RFC_PING[Sync ] is waiting...
[bg#2 thread] Busy. RFC_PING[Async] is waiting...
[main thread] Complete Z_SAPX_CALL_SLEEP[Async]
[bg#2 thread] Busy. RFC_PING[Async] is waiting...
[bg#1 thread] RFC_PING[Sync ] is completed
[bg#2 thread] RFC_PING[Async] is completed
--------------------------------------------------------
```

2.7 Reading of Table Data

The demo shows how to read a data from a SAP table with VCL components. There is a possibility to specify parameters of a query for a SAP server

Location

`SAPx\Demo\Client\05_ReadTableData`

Application area

The following main components are used in the demo:

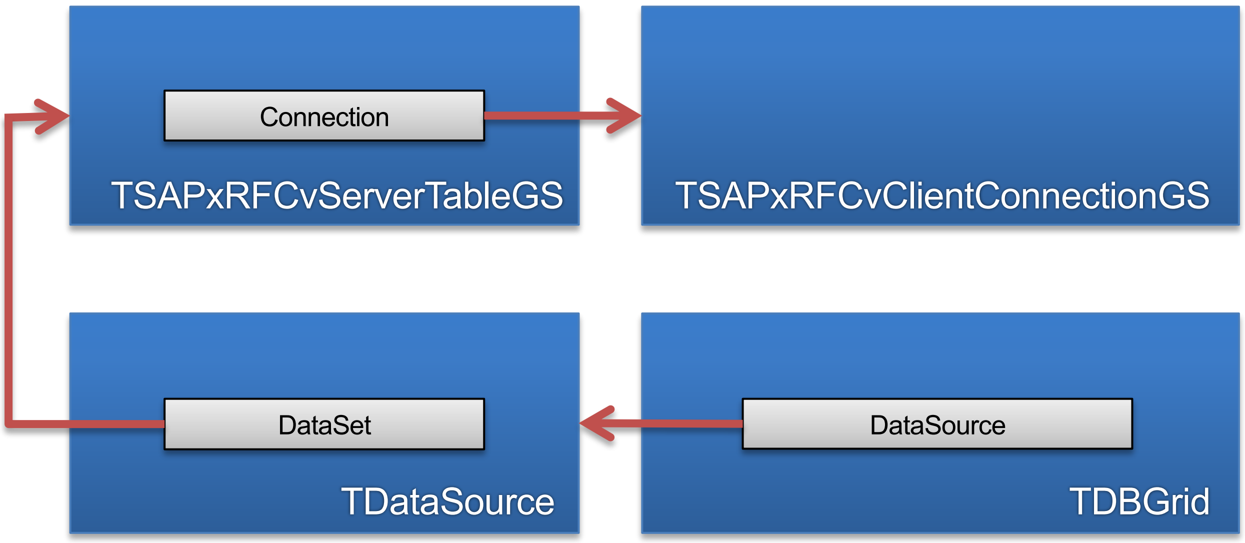
Class	Component name
TSAPxRFCvServerTableGS	FSTable
TDataSource	dsTable
TDBGrid	grdDBGrid
TSAPxRFCvClientConnectionGS	FCConnection

Server table component (`TSAPxRFCvServerTableGS`) is configured as shown below:

Property	Description
TableName	Defines a name of table to be read.
Select	Defines a set of fields to be read.
Where	Defines a set of optional conditions to additionally detail the entries to be read.

After setting the properties, the table's method `Open` is called to start the reading process.

After the execution, the output control `grdDBGrid` is automatically filled by the data. The binding of the mentioned components has the standard VCL approach as shown in the next figure:



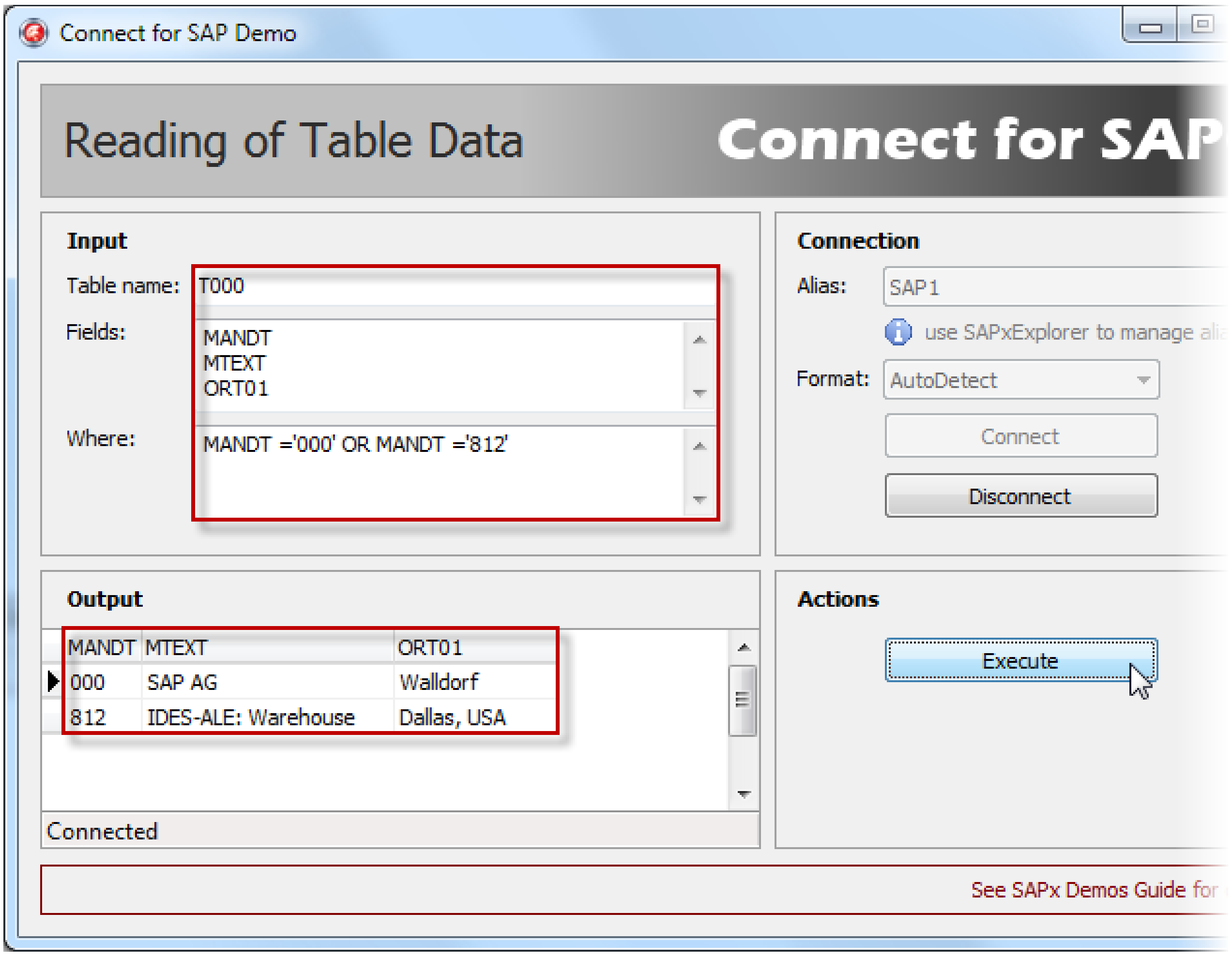
Execution

The user establishes a connection to a SAP server by choosing a desired connection alias. As well the connection format should be specified before connecting.

After pressing `Connect` and specifying the user credentials in the appeared login dialog the connection is established. The user specifies a table name to read the data from. Optionally, the required fields of the table and other conditions can be specified in the `Fields` and the `Where` memos.

At last after the user presses Execute to get result data, which are shown in the Output data grid.

The next picture shows an example of reading data from the standard SAP table T000 with additional parameters of the query:


Connect for SAP Demo

Reading of Table Data

Connect for SAP

Input

Table name: T000

Fields:
MANDT
MTEXT
ORT01

Where: MANDT ='000' OR MANDT ='812'

Connection

Alias: SAP1

use SAPxExplorer to manage ali

Format: AutoDetect

Connect

Disconnect

Output

MANDT	MTEXT	ORT01
000	SAP AG	Walldorf
812	IDES-ALE: Warehouse	Dallas, USA

Connected

Actions

Execute

See SAPx Demos Guide for

2.8 Working with different Data Types

This demo shows in details how different data type specified in the “Appendix A – Data type and mapping” of the “Getting Started” (see Locations of Connect for SAP® Documents) are physically used to represent data on the ABAP and Delphi sides. The Z_SAPX_TEST_PARAMTYPES functional module contains all supported data types among its import and export parameters. The module just transfers all input data to its output. After the execution, the demo compares the input parameters against the output parameters and writes the results into the Output log.

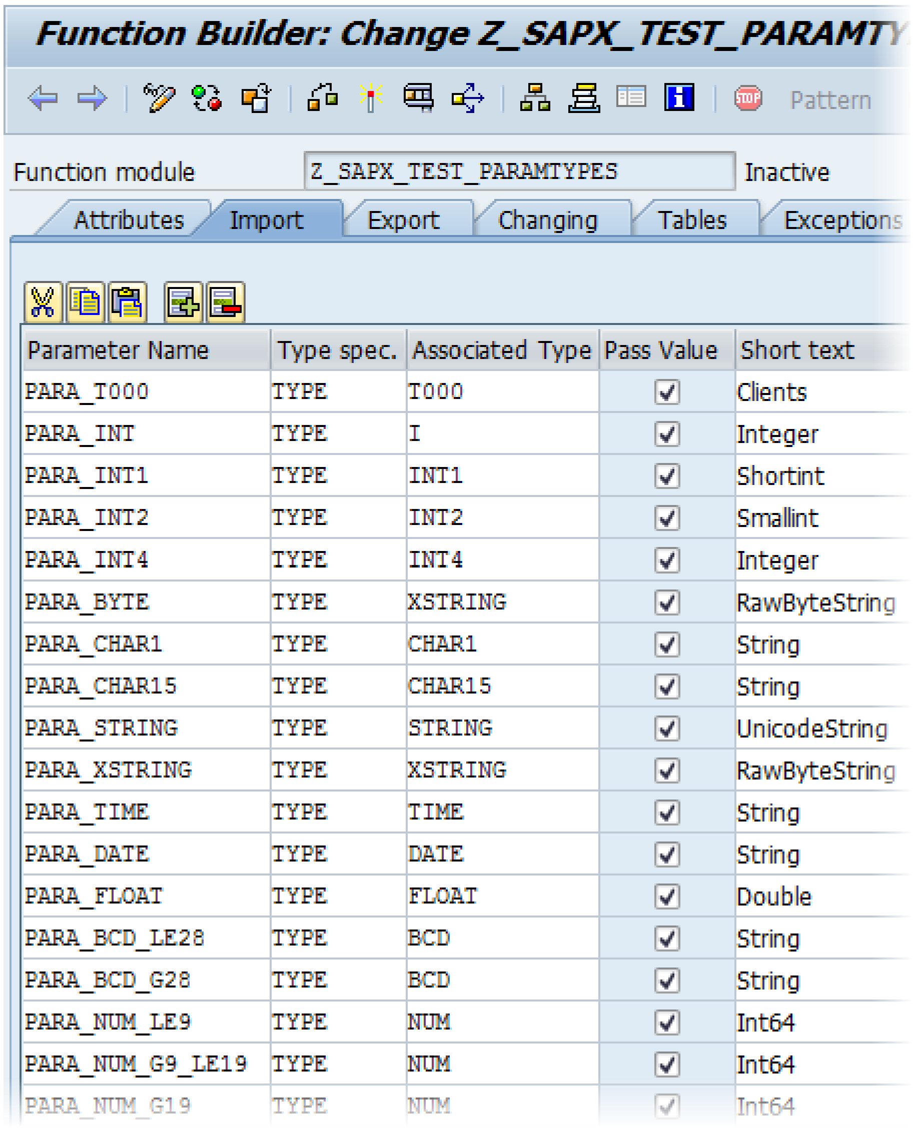
Location

SAPx\Demo\Client\99_DataTypes

SAP system area

The demo expects the function Z_SAPX_TEST_PARAMTYPES has already been created on a SAP server. The process of creation of ABAP functions is described by the topic How to define and execute an RFC Function Module [SE37]. To create the function Z_SAPX_TEST_PARAMTYPES you use the next parameters and source code.

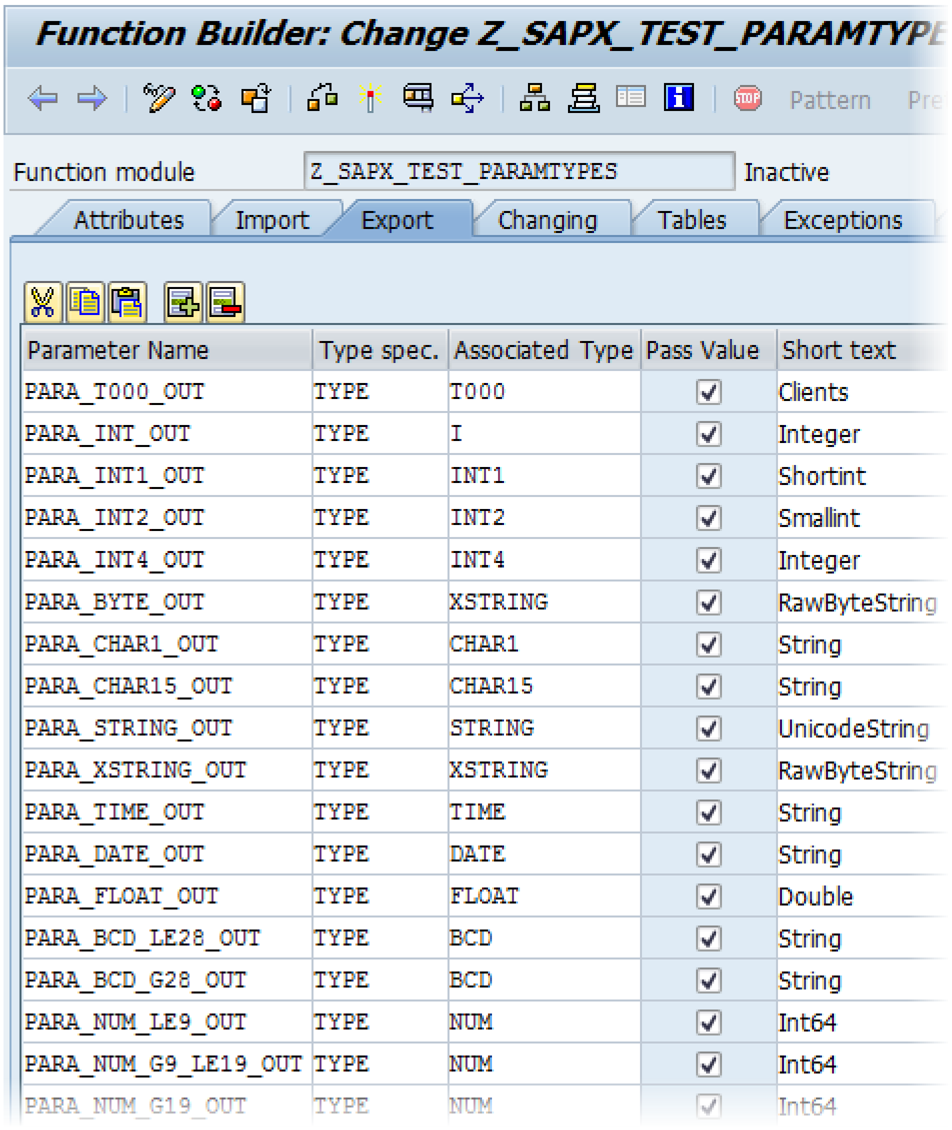
Import parameters:



Function module Z_SAPX_TEST_PARAMTYPES Inactive

Parameter Name	Type spec.	Associated Type	Pass Value	Short text
PARA_T000	TYPE	T000	✓	Clients
PARA_INT	TYPE	I	✓	Integer
PARA_INT1	TYPE	INT1	✓	Shortint
PARA_INT2	TYPE	INT2	✓	Smallint
PARA_INT4	TYPE	INT4	✓	Integer
PARA_BYTE	TYPE	XSTRING	✓	RawByteString
PARA_CHAR1	TYPE	CHAR1	✓	String
PARA_CHAR15	TYPE	CHAR15	✓	String
PARA_STRING	TYPE	STRING	✓	UnicodeString
PARA_XSTRING	TYPE	XSTRING	✓	RawByteString
PARA_TIME	TYPE	TIME	✓	String
PARA_DATE	TYPE	DATE	✓	String
PARA_FLOAT	TYPE	FLOAT	✓	Double
PARA_BCD_LE28	TYPE	BCD	✓	String
PARA_BCD_G28	TYPE	BCD	✓	String
PARA_NUM_LE9	TYPE	NUM	✓	Int64
PARA_NUM_G9_LE19	TYPE	NUM	✓	Int64
PARA_NUM_G19	TYPE	NUM	✓	Int64

Export parameters:



Function Builder: Change Z_SAPX_TEST_PARAMTYPE

Function module Z_SAPX_TEST_PARAMTYPES Inactive

Parameter Name	Type spec.	Associated Type	Pass Value	Short text
PARA_T000_OUT	TYPE	T000	✓	Clients
PARA_INT_OUT	TYPE	I	✓	Integer
PARA_INT1_OUT	TYPE	INT1	✓	Shortint
PARA_INT2_OUT	TYPE	INT2	✓	Smallint
PARA_INT4_OUT	TYPE	INT4	✓	Integer
PARA_BYTE_OUT	TYPE	XSTRING	✓	RawByteString
PARA_CHAR1_OUT	TYPE	CHAR1	✓	String
PARA_CHAR15_OUT	TYPE	CHAR15	✓	String
PARA_STRING_OUT	TYPE	STRING	✓	UnicodeString
PARA_XSTRING_OUT	TYPE	XSTRING	✓	RawByteString
PARA_TIME_OUT	TYPE	TIME	✓	String
PARA_DATE_OUT	TYPE	DATE	✓	String
PARA_FLOAT_OUT	TYPE	FLOAT	✓	Double
PARA_BCD_LE28_OUT	TYPE	BCD	✓	String
PARA_BCD_G28_OUT	TYPE	BCD	✓	String
PARA_NUM_LE9_OUT	TYPE	NUM	✓	Int64
PARA_NUM_G9_LE19_OUT	TYPE	NUM	✓	Int64
PARA_NUM_G19_OUT	TYPE	NUM	✓	Int64

You may find ABAP source code of the module in `SAPx\Demo\Client\99_DataTypes\fMain.pas` unit.

Application area

To access the function and its parameters, the demo uses wrappers generated by the Connect for SAP® Explorer. During the generation process, the Connect for SAP® Explorer is mapping RFC types (referred by the function parameters) onto Delphi types.

The next table shows the mapping for each of these parameters:

Import	Export	RFC type	Delphi type	Description
PARA_T000	PARA_T000_OUT	T000	TSAPxRFCT000StrGS	structure of table T000
PARA_INT	PARA_INT_OUT	I	Integer	4-byte Integer
PARA_INT1	PARA_INT1_OUT	INT1	Shortint	1-byte Integer
PARA_INT2	PARA_INT2_OUT	INT2	Smallint	2-byte Integer
PARA_INT4	PARA_INT4_OUT	INT4	Integer	4-byte Integer
PARA_BYTE	PARA_BYTE_OUT	XSTRING	RawByteString	single byte
PARA_CHAR1	PARA_CHAR1_OUT	CHAR1	String	1-symbol string
PARA_CHAR15	PARA_CHAR15_OUT	CHAR15	String	15-symbol string
PARA_STRING	PARA_STRING_OUT	STRING	UnicodeString	string
PARA_XSTRING	PARA_XSTRING_OUT	XSTRING	RawByteString	array of bytes
PARA_TIME	PARA_TIME_OUT	TIME	String	time as a 6-symbol string
PARA_DATE	PARA_DATE_OUT	DATE	String	date as a 8-symbol string
PARA_FLOAT	PARA_FLOAT_OUT	FLOAT	Double	float
PARA_BCD_LE28	PARA_BCD_LE28_OUT	BCD	String	BCD value containing up to 28 digits
PARA_BCD_G28	PARA_BCD_G28_OUT	BCD	String	BCD value containing more than 28 digits
PARA_NUM_LE9	PARA_NUM_LE9_OUT	NUM	Int64	number containing up to 9 digits
PARA_NUM_G9_LE19	PARA_NUM_G9_LE19_OUT	NUM	Int64	number containing from 10 to 19 digits
PARA_NUM_G19	PARA_NUM_G19_OUT	NUM	Int64	number containing more than 19 digits

Using the wrapper, the demo initializes these import parameters, executes the function and then compares each of the import parameter against the corresponding export parameter.

Execution

The user establishes a connection to a SAP server by choosing a desired alias for the connection, pressing Connect and specifying the user credentials in the appeared login dialog.

After the connection established, the user presses Execute on the main form.

The processes of connecting, executing and comparison of the import/export parameters are written to the Output.

3 Connect for SAP® Server Demos

This section contains demo applications showing how Connect for SAP® works in a role of an external SAP server.

3.1 Simple external SAP Server

The demo shows a simple use case when there is a single server connection executing just one function.

Location

`SAPx\Demo\Server\01_Simple_Server`

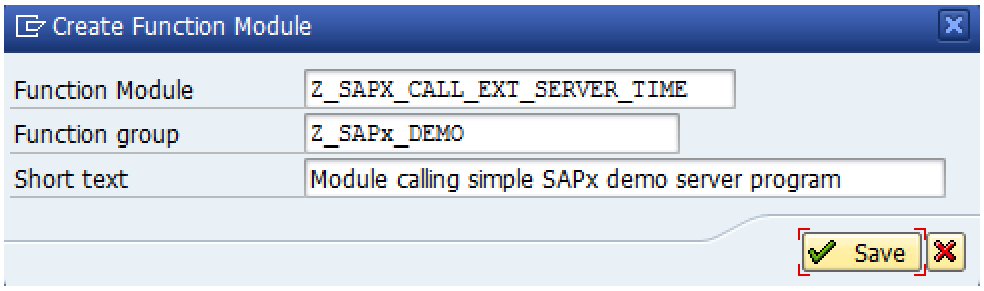
SAP system area

The scenario of the demo assumes that there is an ABAP function module calling an external function that is registered and executed on the Connect for SAP® server implemented within the demo application. To run the demo it is need to properly setup environment on a SAP sever side. The following steps should be done to setup the SAP server environment before running the application:

1. Create a RFC destination Z_SAPX_SERVER_DEST with parameters specified in the next table (see also How to define a Server Destination [SM59]).

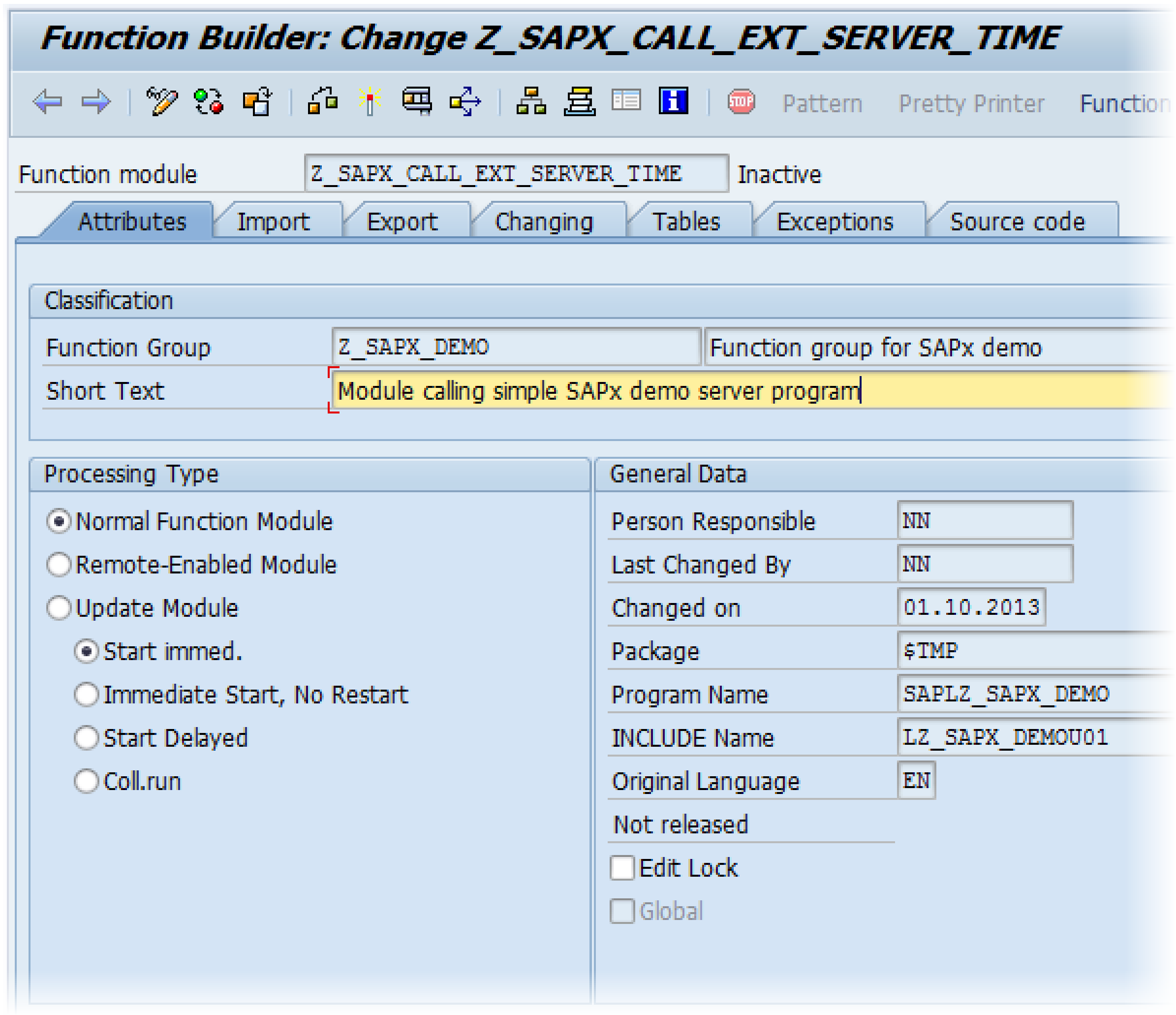
Parameter	Value
RFC destination	Z_SAPX_SERVER_DEST
Connection type	T
Activation type	Registered Server Program
Program ID	sapx_server_prog_id_demo

2. Create a function group Z_SAPx_DEMO if the group does not exist (see How to define a Function Group)
3. Create a function module Z_SAPX_CALL_EXT_SERVER_TIME (see How to define and execute an RFC Function Module [SE37]) in function group Z_SAPx_DEMO. The function module is an ABAP wrapper that calls the external server function.


Create Function Module

Function Module	Z_SAPX_CALL_EXT_SERVER_TIME
Function group	Z_SAPx_DEMO
Short text	Module calling simple SAPx demo server program

4. Press Save button and the main page with attributes appears.



Function Builder: Change Z_SAPX_CALL_EXT_SERVER_TIME

Function module: Z_SAPX_CALL_EXT_SERVER_TIME Inactive

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

Classification

Function Group	Z_SAPX_DEMO	Function group for SAPx demo
Short Text	Module calling simple SAPx demo server program	

Processing Type

- ● Normal Function Module
- ○ Remote-Enabled Module
- ○ Update Module
  - ● Start immed.
  - ○ Immediate Start, No Restart
  - ○ Start Delayed
  - ○ Coll.run

General Data

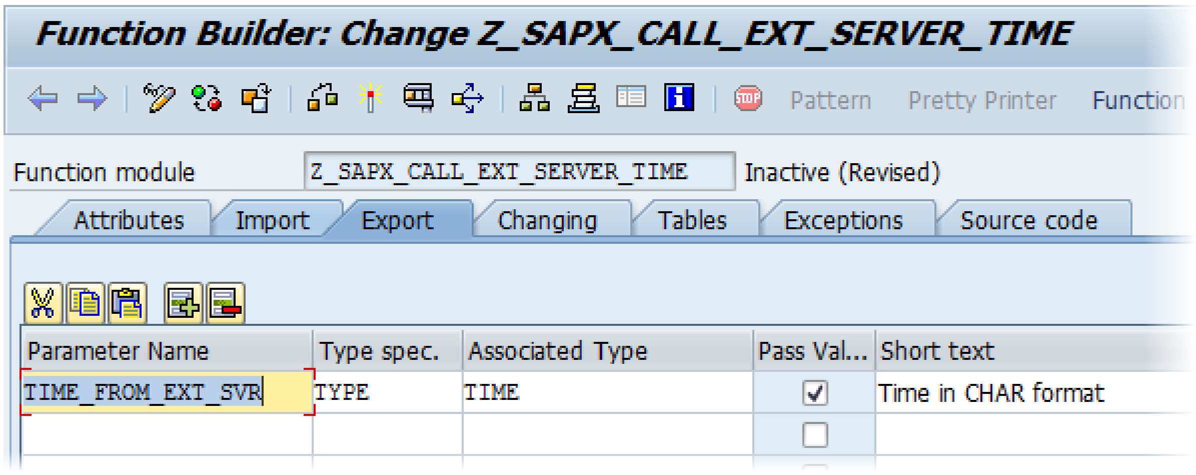
Person Responsible	NN
Last Changed By	NN
Changed on	01.10.2013
Package	$TMP
Program Name	SAPLZ_SAPX_DEMO
INCLUDE Name	LZ_SAPX_DEMOU01
Original Language	EN

Not released

☐ Edit Lock

☐ Global

5. Import page should be empty because the function we are going to work with has no input parameters.
6. Switch to Export page and add a parameter TIME_FROM_EXT_SVR as shown below and press Save button to save changes.



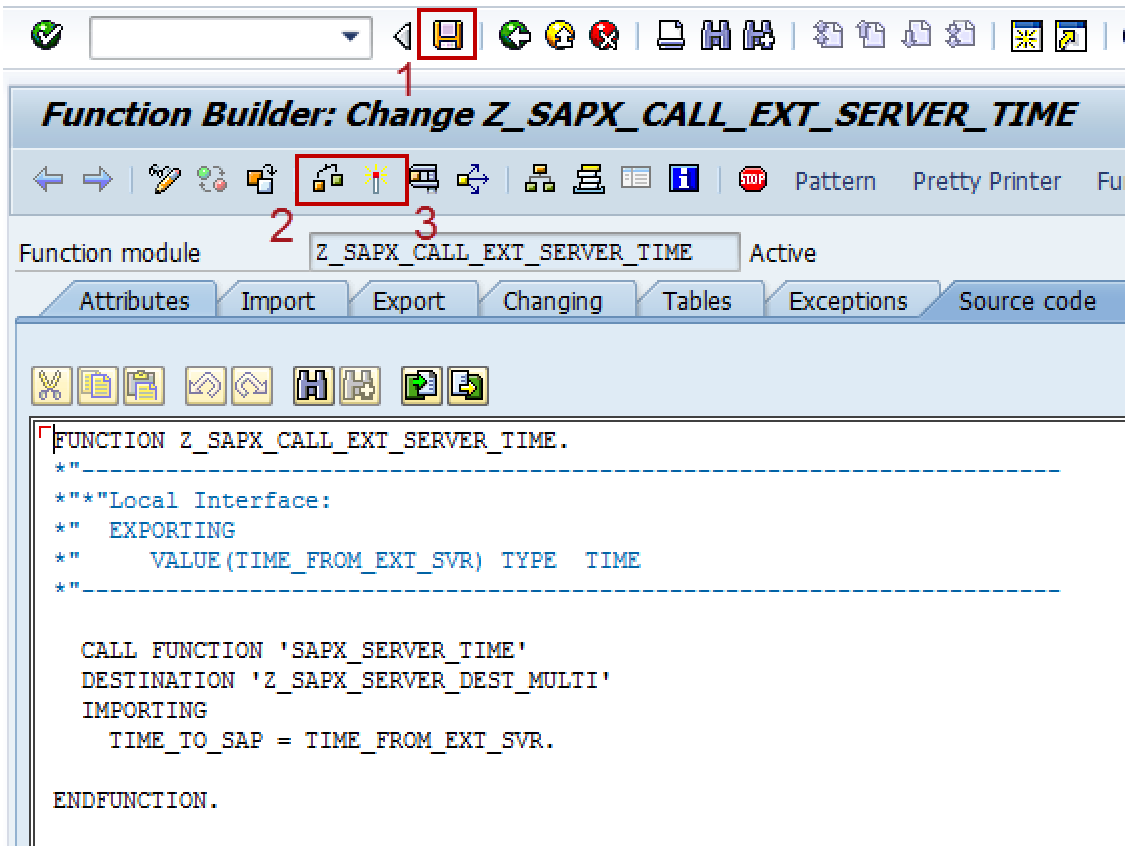
Function Builder: Change Z_SAPX_CALL_EXT_SERVER_TIME

Function module: Z_SAPX_CALL_EXT_SERVER_TIME Inactive (Revised)

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

Parameter Name	Type spec.	Associated Type	Pass Val...	Short text
TIME_FROM_EXT_SVR	TYPE	TIME	✓	Time in CHAR format
			☐	

7. Switch to the Source code page, copy the source code of the functional module from `SAPx\Demo\Server\01_Simple_Server \fMain.pas` unit and press "Save" button to save the changes.
8. Press consequently Save, Check and Activate buttons to save and make the functional module properly installed and activated on the SAP server side.



```
FUNCTION Z_SAPX_CALL_EXT_SERVER_TIME.
*"----------------------------------------------------------------------
*"*"Local Interface:
*"  EXPORTING
*"     VALUE(TIME_FROM_EXT_SVR) TYPE  TIME
*"----------------------------------------------------------------------

  CALL FUNCTION 'SAPX_SERVER_TIME'
  DESTINATION 'Z_SAPX_SERVER_DEST_MULTI'
  IMPORTING
    TIME_TO_SAP = TIME_FROM_EXT_SVR.

ENDFUNCTION.
```

9. As soon as the steps have been done the server environment are ready.

Application area

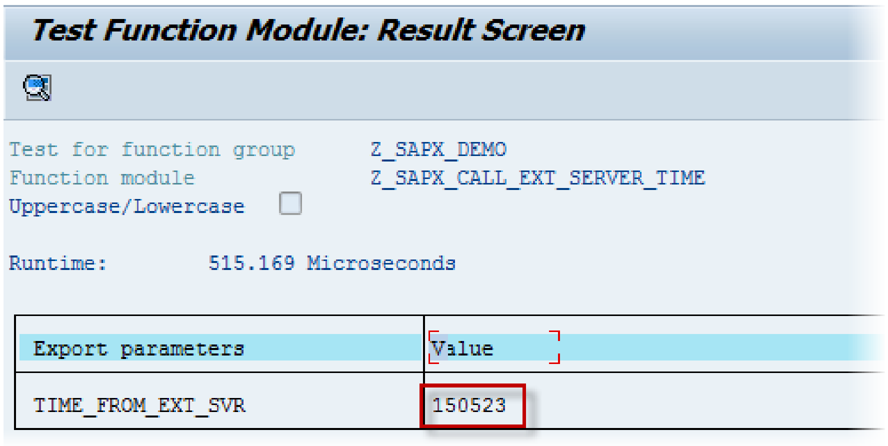
The functionality of the demo is based on two components TSAPxRFCvServerConnectionGS and TSAPxRFCvServerFunctionGS.

The following tables specify main properties of these components:

Property	Value	Description
TSAPxRFCvServerConnectionGS		
CommandLine.PROGID	sapx_server_prog_id_demo	Specify id that is the same as we defined in Program ID for the RFC destination (see the step **above**)
OnError		Handling RFC errors
TSAPxRFCvServerFunctionGS		
Connection	FServerConnection	Reference to a server connection component
ObjName	SAPX_SERVER_TIME	Name of function that is executed . It is the same as we defined in ABAP source code as CALL FUNCTION 'SAPX_SERVER_TIME' (see the step above)

After running, it's need to specify in GWHost box a host name or IP address of the SAP gateway where the external server will be registered. As well specify GWHost if the server port differs from *3300*. Then press Connect button.

To test how the external server responds it is need to return back to the SAP GUI and call the function module by pressing Test/Execute button.



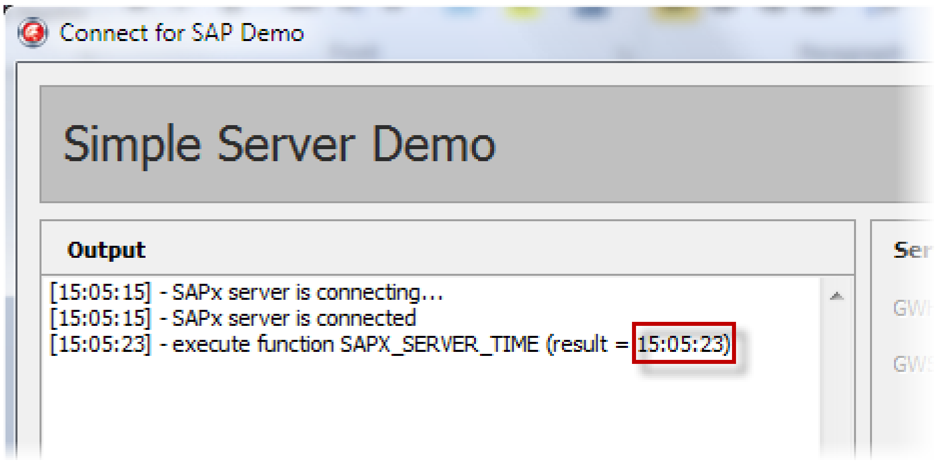
Test Function Module: Result Screen

```
Test for function group     Z_SAPX_DEMO
Function module             Z_SAPX_CALL_EXT_SERVER_TIME
Uppercase/Lowercase

Runtime:        515.169 Microseconds
```

Export parameters	Value
TIME_FROM_EXT_SVR	150523

As a result the export parameter TIME_FROM_EXT_SVR is displayed.

At the same time in the demo the output log contains entry that the function has been called.



```
[15:05:15] - SAPx server is connecting...
[15:05:15] - SAPx server is connected
[15:05:23] - execute function SAPX_SERVER_TIME (result = 15:05:23)
```

3.2 Multiple Connection Server

This application demonstrates more complex case when there are two servers. Each of the servers contains two functions which process requests from different SAP systems. Here is we need as well prepare both environments on two SAP servers and on the application level too.

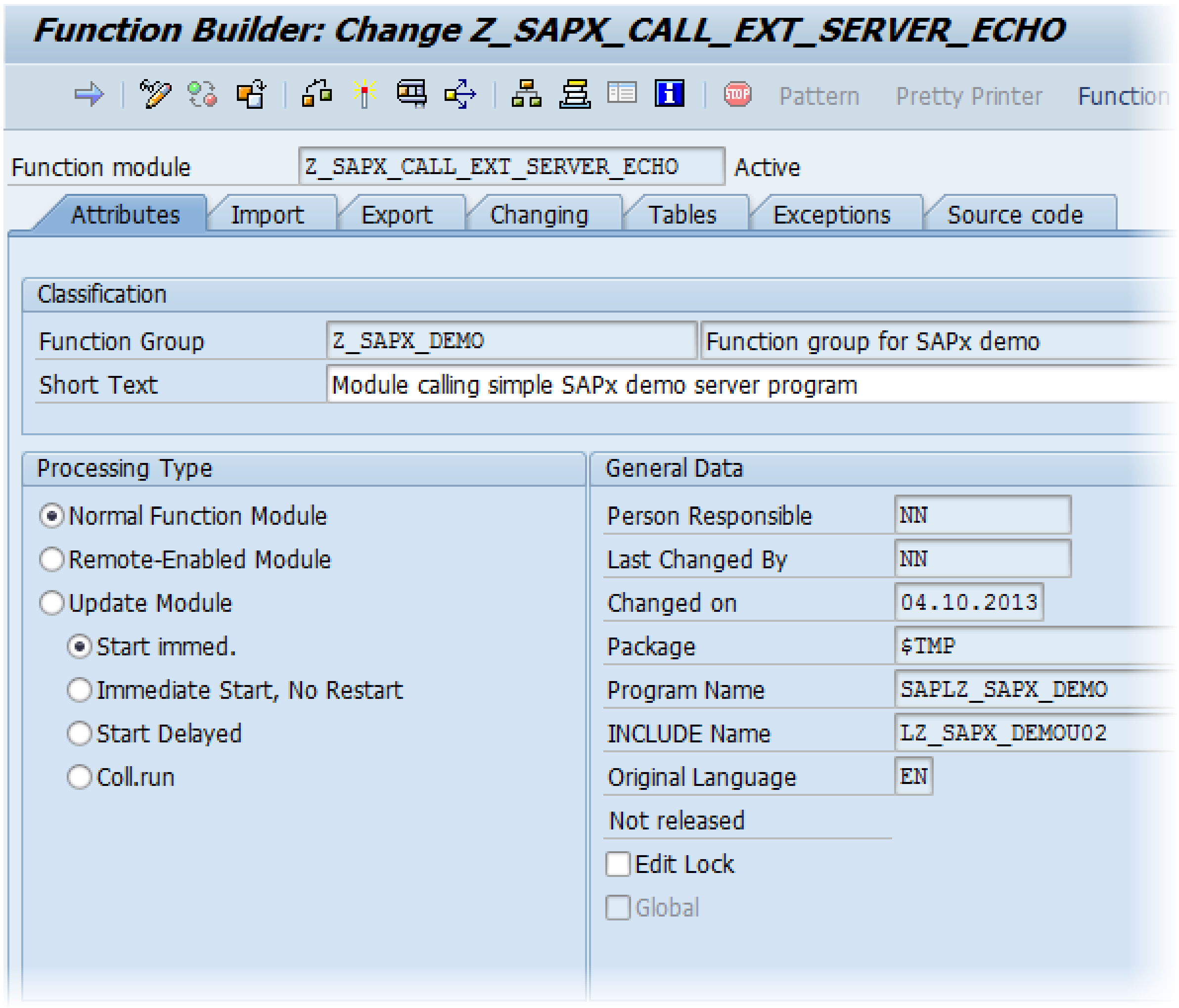
Location

`SAPx\Demo\Server\02_MultiConnections`

SAP system area

The scenario of the demo assumes that there are two SAP systems, which we need to work with. To set the first server we just enhance already existing environment (see SAP system area in the topic Simple external SAP Server) by adding a new RFC functional module. This function module is created similarly as Z_SAPX_CALL_EXT_SERVER_TIME (see **above**).

1. Create a new function module Z_SAPX_CALL_EXT_SERVER_ECHO with SE37(see How to define and execute an RFC Function Module [SE37]).


Function Builder: Change Z_SAPX_CALL_EXT_SERVER_ECHO

Function module Z_SAPX_CALL_EXT_SERVER_ECHO Active

Classification

Function Group	Z_SAPX_DEMO	Function group for SAPx demo
Short Text	Module calling simple SAPx demo server program	

Processing Type

- ● Normal Function Module
- ○ Remote-Enabled Module
- ○ Update Module
  - ● Start immed.
  - ○ Immediate Start, No Restart
  - ○ Start Delayed
  - ○ Coll.run

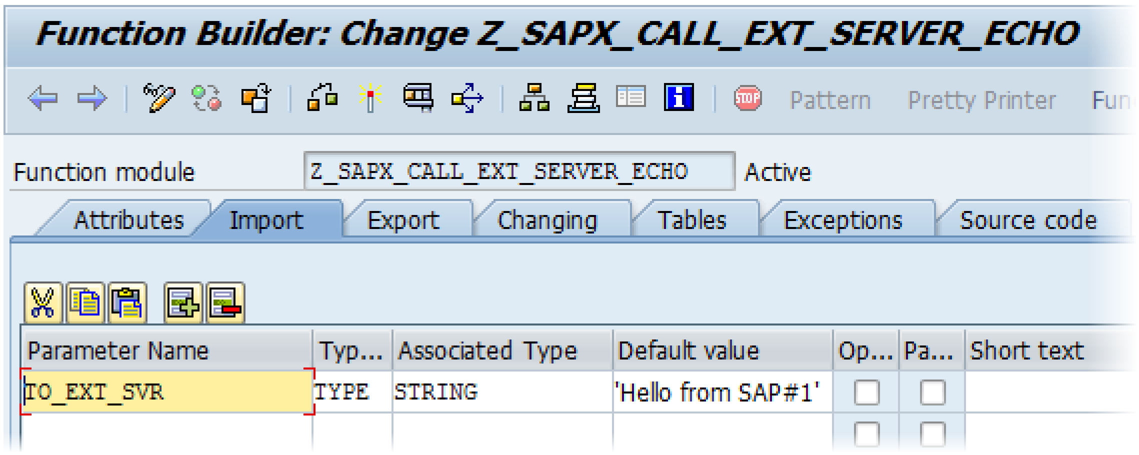
General Data

Person Responsible	NN
Last Changed By	NN
Changed on	04.10.2013
Package	$TMP
Program Name	SAPLZ_SAPX_DEMO
INCLUDE Name	LZ_SAPX_DEMOU02
Original Language	EN

Not released

☐ Edit Lock
☐ Global

2. Add Import parameter TO_EXT_SVR.

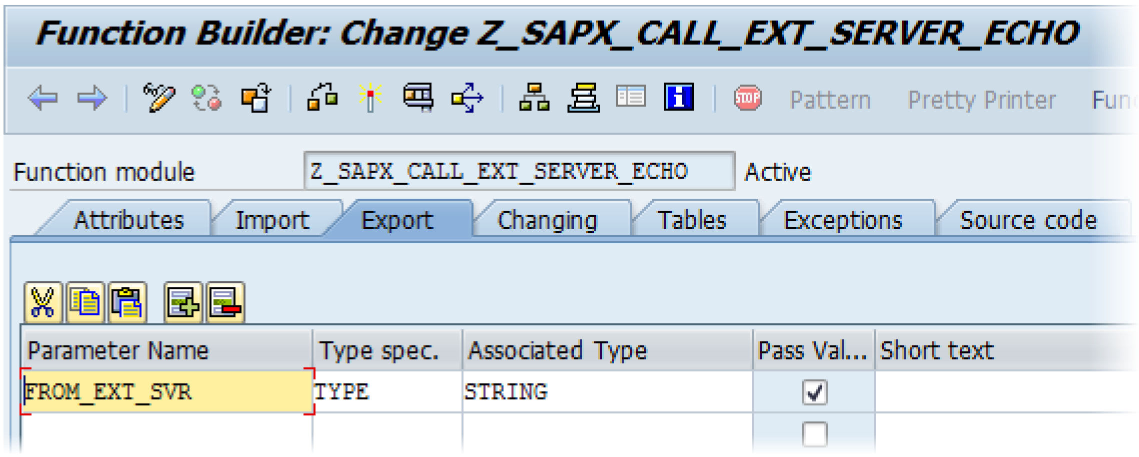


Function Builder: Change Z_SAPX_CALL_EXT_SERVER_ECHO

Function module Z_SAPX_CALL_EXT_SERVER_ECHO Active

Parameter Name	Typ...	Associated Type	Default value	Op...	Pa...	Short text
TO_EXT_SVR	TYPE	STRING	'Hello from SAP#1'	☐	☐	

3. Add export parameter FROM_EXT_SVR.



Function Builder: Change Z_SAPX_CALL_EXT_SERVER_ECHO

Function module Z_SAPX_CALL_EXT_SERVER_ECHO Active

Parameter Name	Type spec.	Associated Type	Pass Val...	Short text
FROM_EXT_SVR	TYPE	STRING	✓	

4. Copy ABAP source code from `SAPx\Demo\ Server\02_MultiConnections\fMain.pas` unit.
5. Press Save, Check and Activate buttons.

At the stage SAP server, name it simply SAP#1, has been prepared for working. The set of steps for setting another server SAP#2 is the same. The only difference is that on SAP#2 server default value of the import parameter TO_EXT_SVR in Z_SAPX_CALL_EXT_SERVER_ECHO functional module is *'Hello from SAP#2'* instead of *'Hello from SAP#1'*.

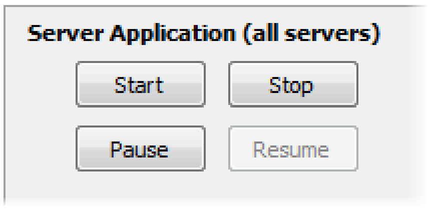
Application area

In the application there are two servers implemented in standalone class TSAPxInfoServer in the following unit:

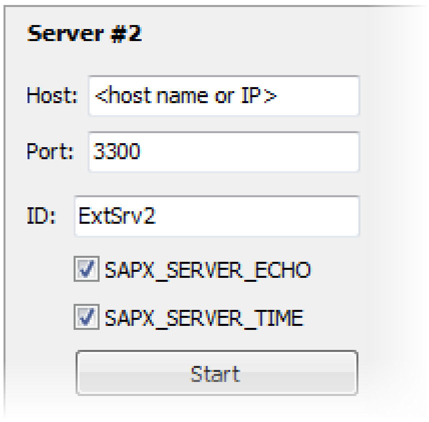
`SAPx\Demo\Server\02_MultiConnections\uSAPxTestServer.pas`

All tuning of such server objects are performed on fly. At runtime it remains only to specify Host and Port values of correspondent SAP gateways where the servers should be registered.

There are two ways to control the servers. The first way is starting or stopping all servers together in section Server Application (all servers).In this way suspending and resuming for all server connections are possible.



The second option is to handle each server independently within the panel of the specific server.



Server #2

Host: <host name or IP>

Port: 3300

ID: ExtSrv2

☑ SAPX_SERVER_ECHO

☑ SAPX_SERVER_TIME

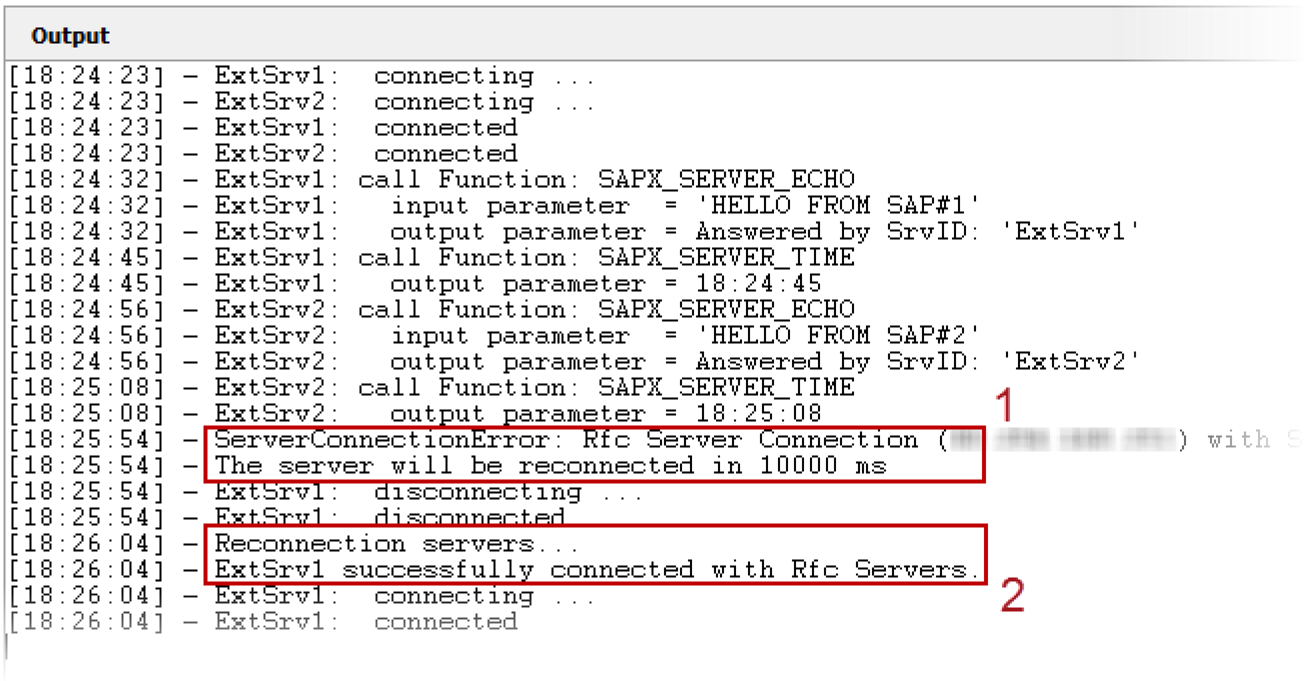
As well in the demo there is an example of restoring activities of the servers on errors that can lead to falling down a server connection. For this goal a timer is used to properly restore working. Its interval is 10 seconds and as soon as a connection is failed in the interval the server will try to reconnect with the SAP server.

Execution

Run the demo application; specify Host and Port values for both servers. And press Start buttons for the server(s) you want to work with. The Output memo displays a status of the server connections.

Then switch to the SAP GUI and log in SAP#1 and SAP#2 servers. Create additional sessions for each server (see menu System -> Create Session). There should be two sessions per each SAP server. Press Execute button in all four windows one by one.

The application output looks like the following:



Output

```
[18:24:23] – ExtSrv1:  connecting ...
[18:24:23] – ExtSrv2:  connecting ...
[18:24:23] – ExtSrv1:  connected
[18:24:23] – ExtSrv2:  connected
[18:24:32] – ExtSrv1: call Function: SAPX_SERVER_ECHO
[18:24:32] – ExtSrv1:   input parameter  = 'HELLO FROM SAP#1'
[18:24:32] – ExtSrv1:   output parameter = Answered by SrvID: 'ExtSrv1'
[18:24:45] – ExtSrv1: call Function: SAPX_SERVER_TIME
[18:24:45] – ExtSrv1:   output parameter = 18:24:45
[18:24:56] – ExtSrv2: call Function: SAPX_SERVER_ECHO
[18:24:56] – ExtSrv2:   input parameter  = 'HELLO FROM SAP#2'
[18:24:56] – ExtSrv2:   output parameter = Answered by SrvID: 'ExtSrv2'
[18:25:08] – ExtSrv2: call Function: SAPX_SERVER_TIME
[18:25:08] – ExtSrv2:   output parameter = 18:25:08
[18:25:54] – ServerConnectionError: Rfc Server Connection ([redacted]) with S
[18:25:54] – The server will be reconnected in 10000 ms
[18:25:54] – ExtSrv1:  disconnecting ...
[18:25:54] – ExtSrv1:  disconnected
[18:26:04] – Reconnection servers...
[18:26:04] – ExtSrv1 successfully connected with Rfc Servers.
[18:26:04] – ExtSrv1:  connecting ...
[18:26:04] – ExtSrv1:  connected
```

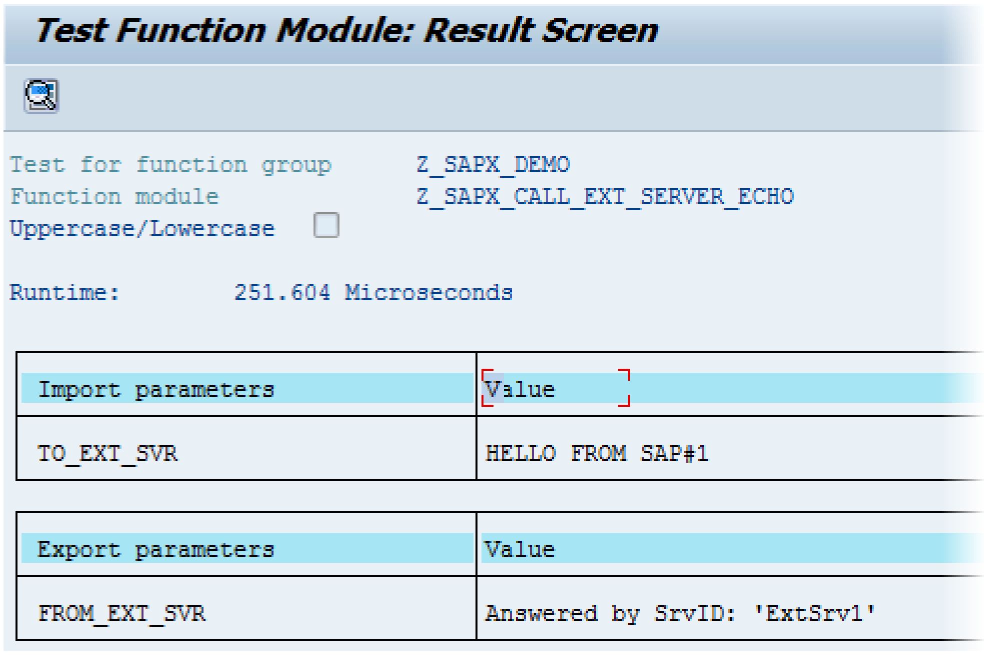
The output log shows that the connection with SAP#1 server is lost and then was successfully reconnected.

Hint: You have to know that event handlers for a server connection's events (e.g. BeforeConnect, AfterConnect, BeforeDisconnect, AfterDisconnect , OnError and etc.) are called within the context of the server connection's thread but not within main thread's one. It means that all VCL calls have to be **synchronized** inside the handlers' code.

The application synchronizes such code with default Synchronize/CheckSyncronize approach. The details are shown in the listing below. The DoLog method puts all requests in the thread's queue. Later during CheckSynchronize executing the calls FOnLog(sMsg)are executed in the main thread and further interaction with VCL controls like memOutput.Lines.Add(…) correctly goes.

```
{ --------------------------------------------------------------------------- }
procedure TSAPxInfoServer.DoLog(AMsg: string);
var
  sMsg: string;
begin
  if Assigned(FOnLog) then begin
    sMsg := ID + ': ' + AMsg;
    TThread.Queue(nil,
      procedure
      begin
        FOnLog(sMsg);
      end
    );
  end;
end;

{ --------------------------------------------------------------------------- }
procedure TSAPxInfoServer.DoBeforeConnect(ASender: TObject);
begin
  DoLog(' connecting ... ');
end;

...

{ --------------------------------------------------------------------------- }
procedure TfrmMain.DoLogServer(AMsg: string);
begin
  LogMessage(AMsg);
end;

{ --------------------------------------------------------------------------- }
procedure TfrmMain.LogMessage(const AMessage: string);
begin
  memOutput.Lines.Add(Format('[%s] - %s', [FormatDateTime('hh:mm:ss', Time), AMessage]))
end;
```

SAP#1 server output:

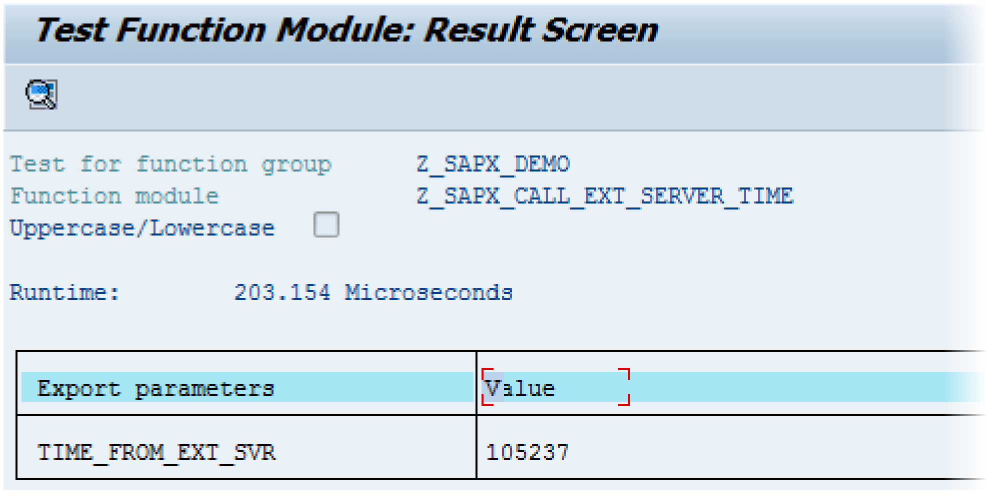


Test Function Module: Result Screen

```
Test for function group       Z_SAPX_DEMO
Function module               Z_SAPX_CALL_EXT_SERVER_ECHO
Uppercase/Lowercase      ☐

Runtime:         251.604 Microseconds
```

Import parameters	Value
TO_EXT_SVR	HELLO FROM SAP#1

Export parameters	Value
FROM_EXT_SVR	Answered by SrvID: 'ExtSrv1'

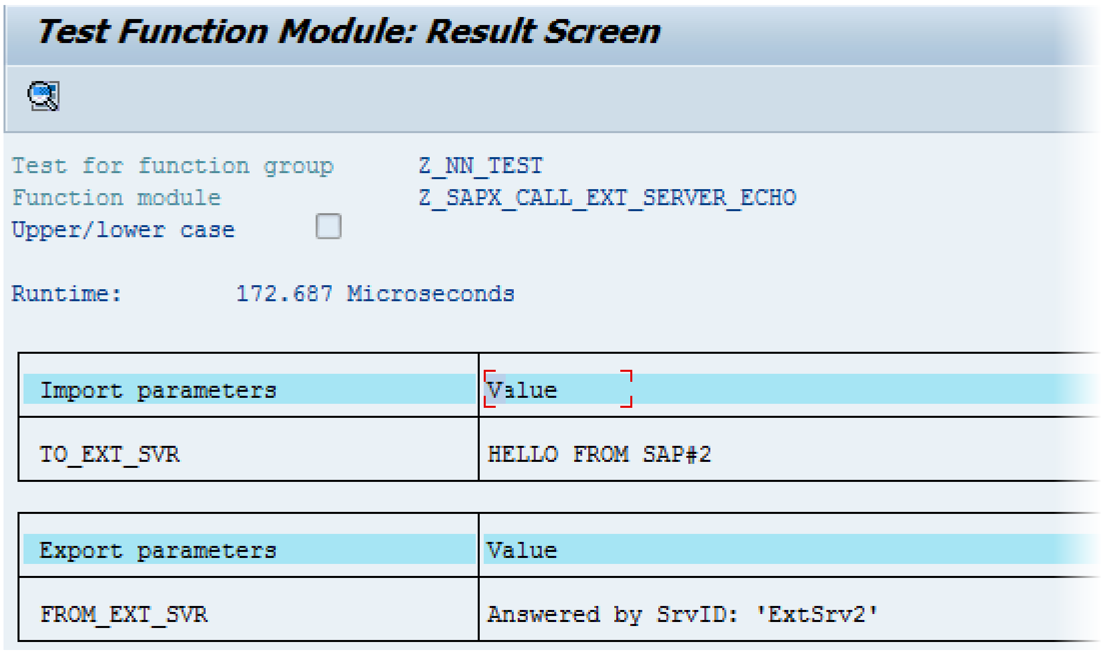


Test Function Module: Result Screen

```
Test for function group       Z_SAPX_DEMO
Function module               Z_SAPX_CALL_EXT_SERVER_TIME
Uppercase/Lowercase      ☐

Runtime:         203.154 Microseconds
```

Export parameters	Value
TIME_FROM_EXT_SVR	105237

SAP#2 server output:

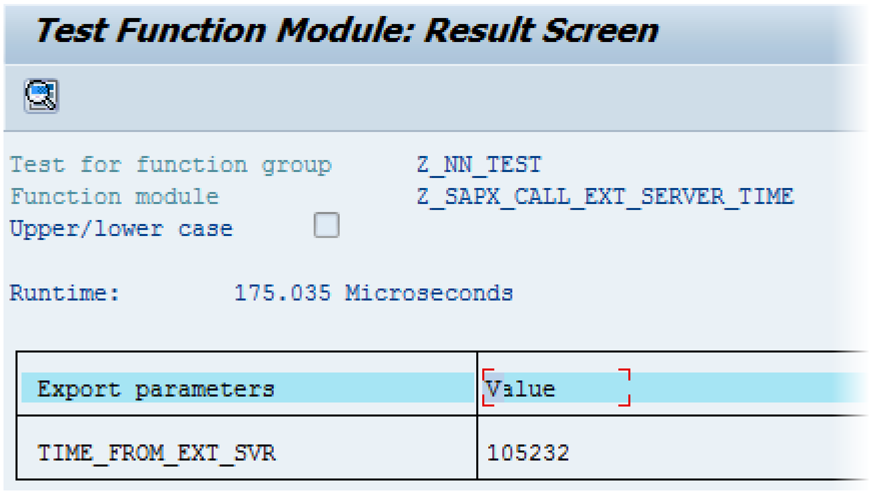


Test Function Module: Result Screen

```
Test for function group      Z_NN_TEST
Function module              Z_SAPX_CALL_EXT_SERVER_ECHO
Upper/lower case        ☐

Runtime:        172.687 Microseconds
```

Import parameters	Value
TO_EXT_SVR	HELLO FROM SAP#2

Export parameters	Value
FROM_EXT_SVR	Answered by SrvID: 'ExtSrv2'



Test Function Module: Result Screen

```
Test for function group      Z_NN_TEST
Function module              Z_SAPX_CALL_EXT_SERVER_TIME
Upper/lower case        ☐

Runtime:        175.035 Microseconds
```

Export parameters	Value
TIME_FROM_EXT_SVR	105232

3.3 Transactional Server

The application demonstrates a simple implementation of an external server working with tRFC protocol (see details of transactional mode in the **Appendix D – Transaction management in Connect for SAP® server application** of the “Getting Started”). The server contains a single function, which processes requests from a SAP system. To properly working it’s needed to prepare environments on both the SAP system’s side and the application’s one.

Location

`SAPx\Demo\Server\03_tRFCServer`

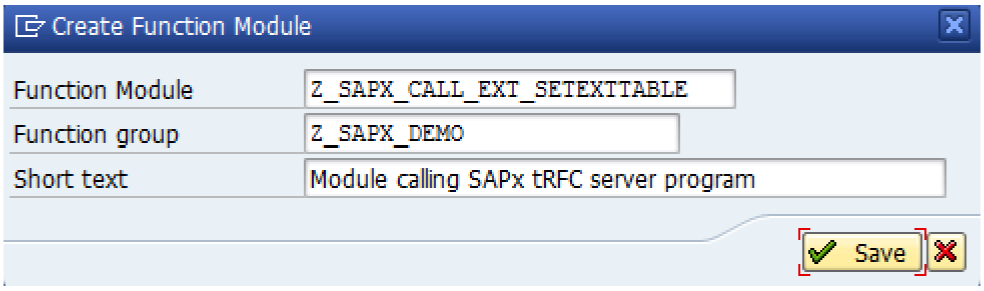
SAP system area

The scenario of the demo assumes that there is a SAP system, which we need to work with. The functional module Z_SAPX_CALL_EXT_SETEXTTABLE calls in background mode function on our demo server. The following steps should be done to setup the SAP server environment before running the application:

1. Create an RFC destination Z_SAPX_SERVER_DEST_TRFC with parameters specified in the next table (see also How to define a Server Destination [SM59]).

Parameter	Value
RFC destination	Z_SAPX_SERVER_DEST_TRFC
Connection type	T
Activation type	Registered Server Program
Program ID	sapx_server_prog_id_demo_trfc

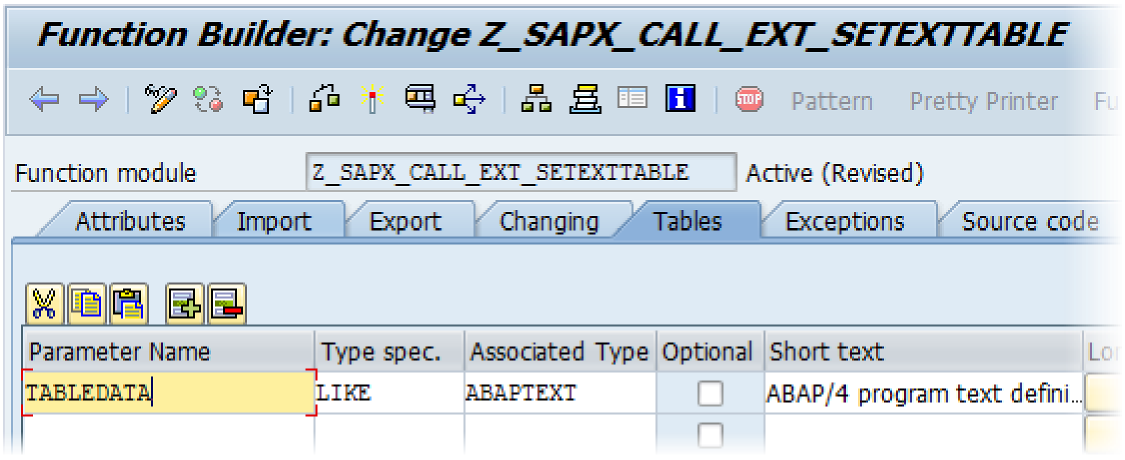
2. Create a function group Z_SAPx_DEMO if the group does not exist (see How to define a Function Group).
3. Create a function module Z_SAPX_CALL_EXT_SETEXTTABLE (see How to define and execute an RFC Function Module [SE37]) in function group Z_SAPx_DEMO.



Function Module	Z_SAPX_CALL_EXT_SETEXTTABLE
Function group	Z_SAPX_DEMO
Short text	Module calling SAPx tRFC server program

4. Press Save button and main page with attributes appears.
5. Import and Export pages should be empty because the function we are going to work with has no input parameters.

6. Switch to Tables page and add a table TABLEDATA as shown below and press Save button to save changes.


Function Builder: Change Z_SAPX_CALL_EXT_SETEXTTABLE

Function module Z_SAPX_CALL_EXT_SETEXTTABLE Active (Revised)

Parameter Name	Type spec.	Associated Type	Optional	Short text
TABLEDATA	LIKE	ABAPTEXT	☐	ABAP/4 program text defini…
			☐	

7. Switch to Source code page, copy ABAP source code of the functional module from `SAPx\Demo\Client\ Server\03_tRFCServer\fMain.pas` and press “Save” button to save changes.
8. Press consequently Save, Check and Activate buttons to save and make the functional module properly installed and activated on the SAP server side.
9. As soon as the steps have been done the server environment is ready.

Application area

The functionality of the demo is based on two classes TSAPxRFCServerConnectionGS and TSAPxRFCServerFunctionGS. The following Listing 2 shows how the connection and function objects are initialized in the code on the fly.

Listing 2: Initialization tRFC server on fly

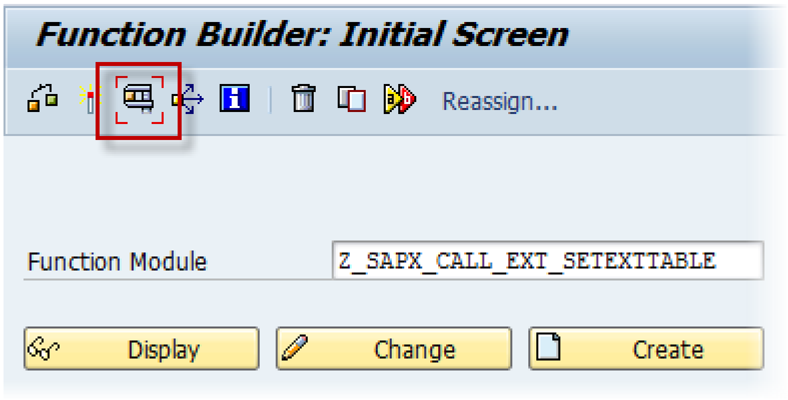
```
  SAPxRFCServerApplication.UseTransactionControl := True;
  ...
  // connection
  FSConnection := TSAPxRFCServerConnectionGS.Create;
  FSConnection.OnErrorEvent.Add(HandleError);
  FSConnection.OnCheckTIDEvent.Add(HandleCheckTID);
  FSConnection.OnCommitEvent.Add(HandleCommit);
  FSConnection.OnConfirmEvent.Add(HandleConfirm);
  FSConnection.OnRollbackEvent.Add(HandleRollback);
  ...

  // function
  FSFunction := TSAPxRFCServerFunctionGS.Create;
  FSFunction.Connection := FSConnection;
  FSFunction.OnExecute := HandleExecute;
  FSFunction.Name := 'SAPX_SERVER_RFCSETEXTTABLE';
  oTable := FSFunction.Tables.AddTable;
  oTable.Name := 'DATA';
  oField := oTable.Fields.AddField;
  oField.Name := 'TEXT';
  oField.DataType := dtCharGS;
  oField.CharacterSize := 72;
```

There are four handlers HandleCheckTID, HandleCommit, HandleConfirm and HandleRollback, which are specific for tRFC mode. Each of them responds for handling corresponding stage of executing a transaction. As well as in non-transactional mode the function has HandleExecute handler assigned that executes all actions required from the server function SAPX_SERVER_RFCSETEXTTABLE. In our case the procedure just passes through all entries in its table and writes them into a log.

Execution

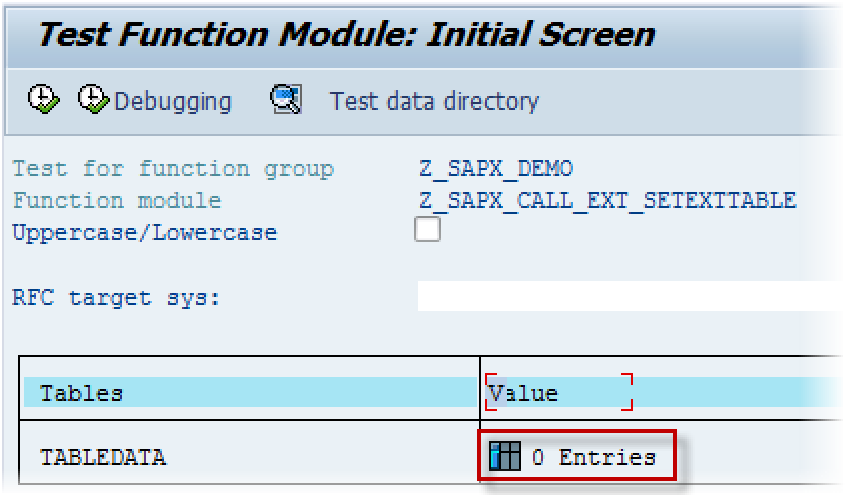
Run the demo application; specify Host and Port values for the server and press Start button. The Output memo displays a status of the server connection.

Then switch to the SAP GUI and log in the SAP system. Run transaction SE37, specify the function module as Z_SAPX_CALL_EXT_SETEXTTABLE and press Test



Function Module Z_SAPX_CALL_EXT_SETEXTTABLE

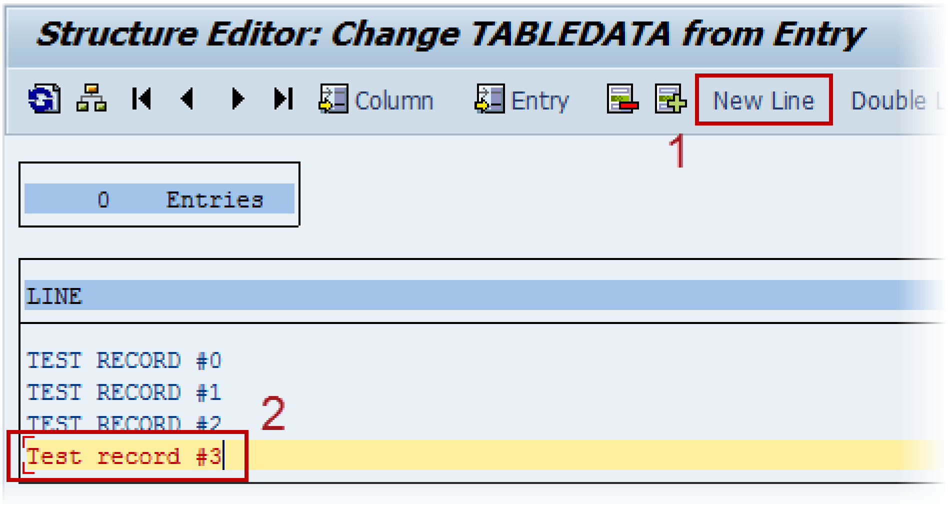
On the Test Function Module screen you need to add several entries into the TABLEDATA table. To open the structure editor press on the Entries area.



Test for function group Z_SAPX_DEMO
Function module Z_SAPX_CALL_EXT_SETEXTTABLE
Uppercase/Lowercase
RFC target sys:

Tables	Value
TABLEDATA	0 Entries

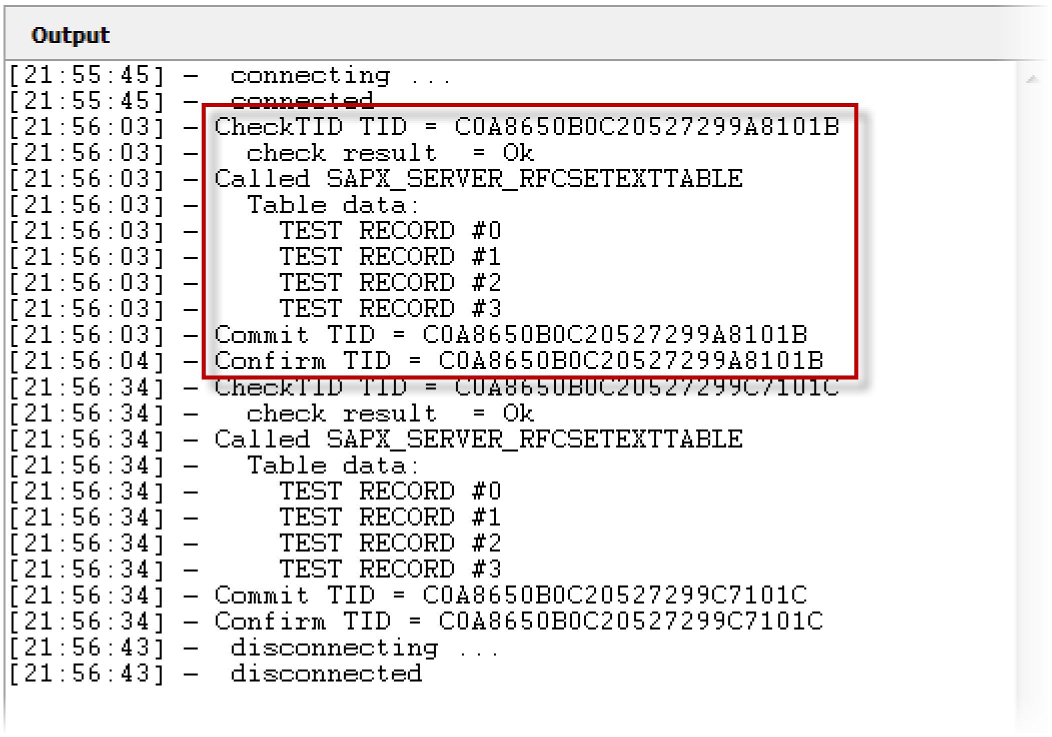
Press New Line and edit each entry as shown below.



```
LINE
TEST RECORD #0
TEST RECORD #1
TEST RECORD #2
Test record #3
```

As soon as the table data is prepared back to the previous the Test Function Module screen and press Execute. Execute the function again with the same data.

The application output shows the results of execution in two transactions (transactional identifiers or TID are different for each call). After handling the calls press Stop to shut the server down.



Output

```
[21:55:45] -  connecting ...
[21:55:45] -  connected
[21:56:03] - CheckTID TID = C0A8650B0C20527299A8101B
[21:56:03] -   check result  = Ok
[21:56:03] - Called SAPX_SERVER_RFCSETEXTTABLE
[21:56:03] -   Table data:
[21:56:03] -     TEST RECORD #0
[21:56:03] -     TEST RECORD #1
[21:56:03] -     TEST RECORD #2
[21:56:03] -     TEST RECORD #3
[21:56:03] - Commit TID = C0A8650B0C20527299A8101B
[21:56:04] - Confirm TID = C0A8650B0C20527299A8101B
[21:56:34] - CheckTID TID = C0A8650B0C20527299C7101C
[21:56:34] -   check result  = Ok
[21:56:34] - Called SAPX_SERVER_RFCSETEXTTABLE
[21:56:34] -   Table data:
[21:56:34] -     TEST RECORD #0
[21:56:34] -     TEST RECORD #1
[21:56:34] -     TEST RECORD #2
[21:56:34] -     TEST RECORD #3
[21:56:34] - Commit TID = C0A8650B0C20527299C7101C
[21:56:34] - Confirm TID = C0A8650B0C20527299C7101C
[21:56:43] -  disconnecting ...
[21:56:43] -  disconnected
```

4 Appendix A: Using the ABAP Workbench

4.1 Intro

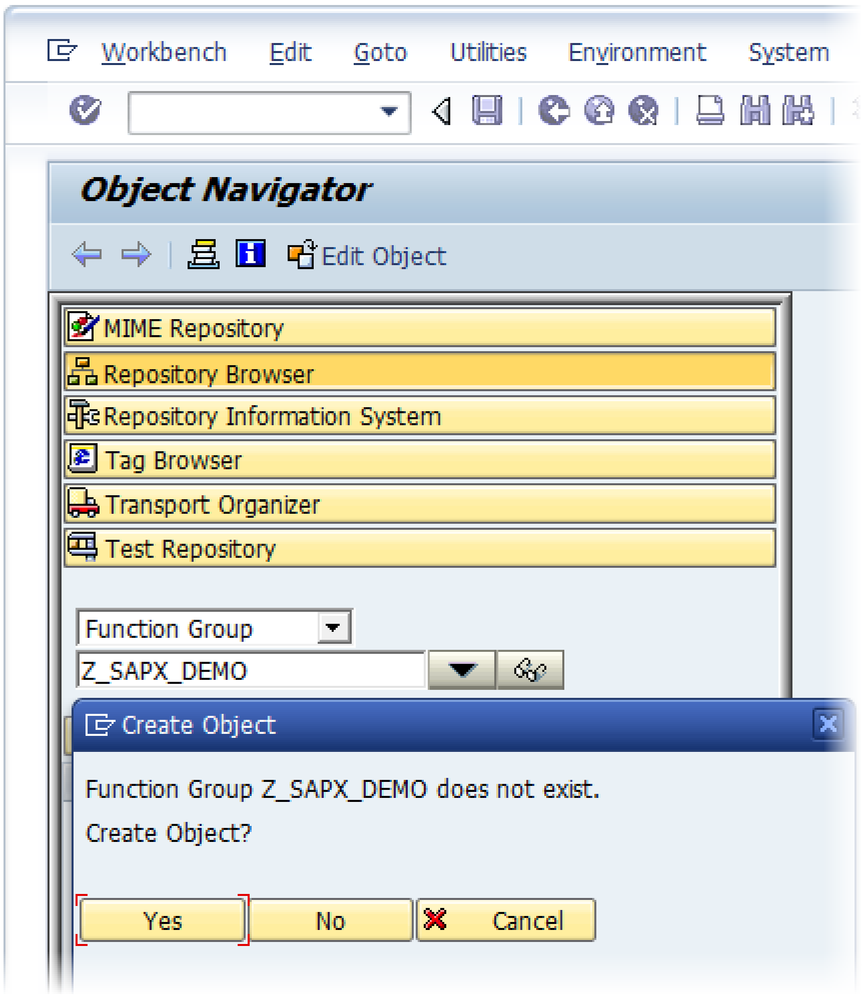
In the appendix you find some tips and short descriptions of use cases, which Connect for SAP® users usually face with. However, there is no knowledge for deep tuning, optimization and other ABAP related topics. Use **http://www.sap.com/** to get more details for that.

In the case if you want to trial Connect for SAP® and you have no available SAP system it makes sense to look at **http://www.idesaccess.com/** service or other similar resources.

4.2 How to define a Function Group

You use this procedure to create function groups in the Function Builder.

1. In the Object Navigator (transaction SE80), choose Function Group as the object type.
2. Enter the name of the new function group and choose Enter. If the object does not exist in the system, the Create Object dialog box appears. Function group names can be up to 26 alphanumeric characters long. You should observe the naming conventions for the first character: A to X for SAP developments, Y and Z for customers.



Function Group

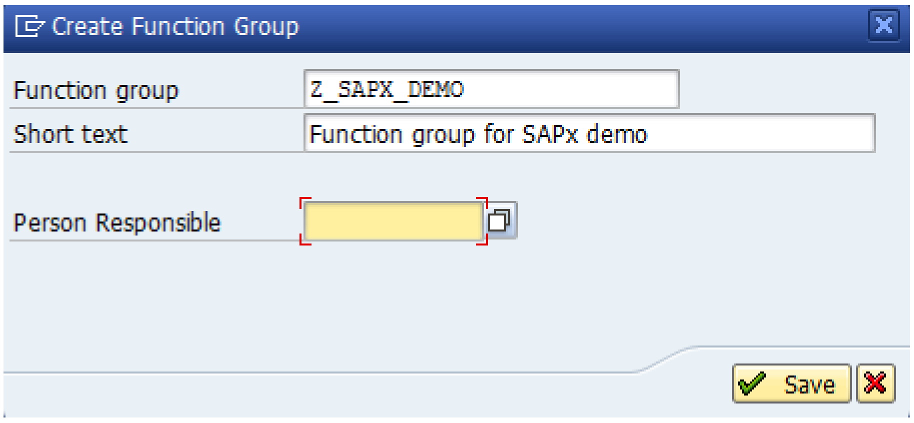
Z_SAPX_DEMO

Create Object

Function Group Z_SAPX_DEMO does not exist.

Create Object?

3. Choose Yes. The Create Function Group screen appears.

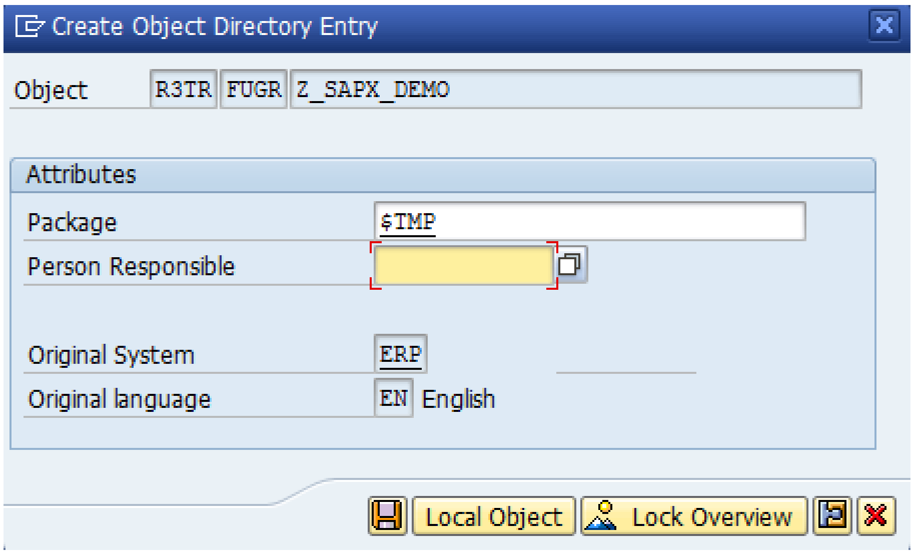


Create Function Group

Function group	Z_SAPX_DEMO
Short text	Function group for SAPx demo
Person Responsible	

4. In the Short Text field, enter a description for the new function group and choose Save. The Create Object Directory Entry screen appears.

5. In the Package field, enter a name of a package or choose the Local Object push button to save the new function group locally.



Create Object Directory Entry

Object	R3TR	FUGR	Z_SAPX_DEMO

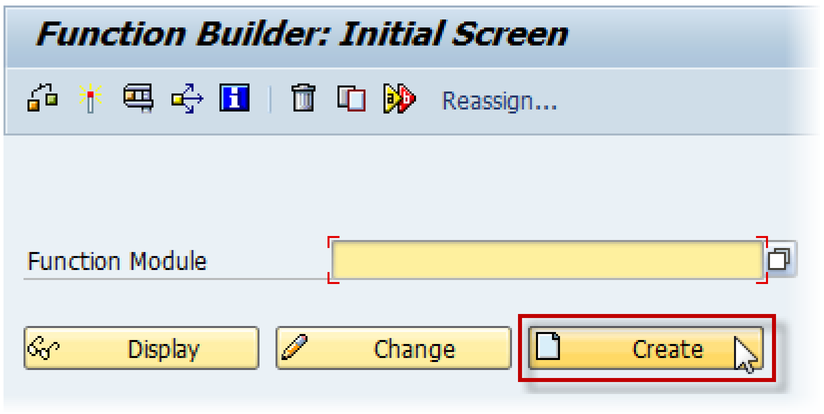
Attributes

Package	$TMP
Person Responsible	
Original System	ERP
Original language	EN English

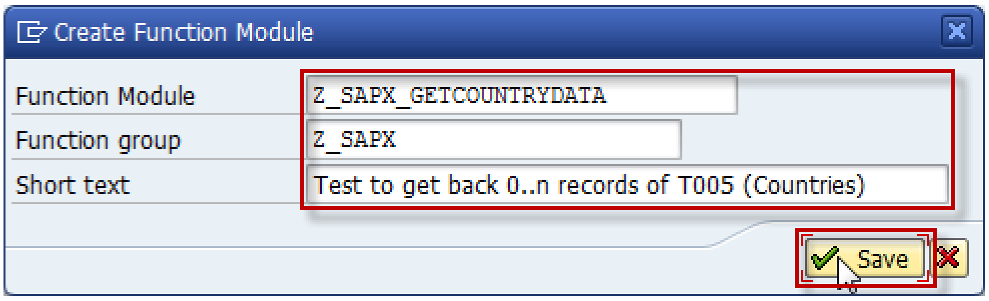
4.3 How to define and execute an RFC Function Module [SE37]

The next steps show an example of creation and execution of a new RFC function module Z_SAPX_GETCOUNTRYDATA. The function takes a filter string as input and returns a table containing some information for the countries defined by the filter.

1. Open Function Builder (transaction SE37) and press Create in Initial Screen.

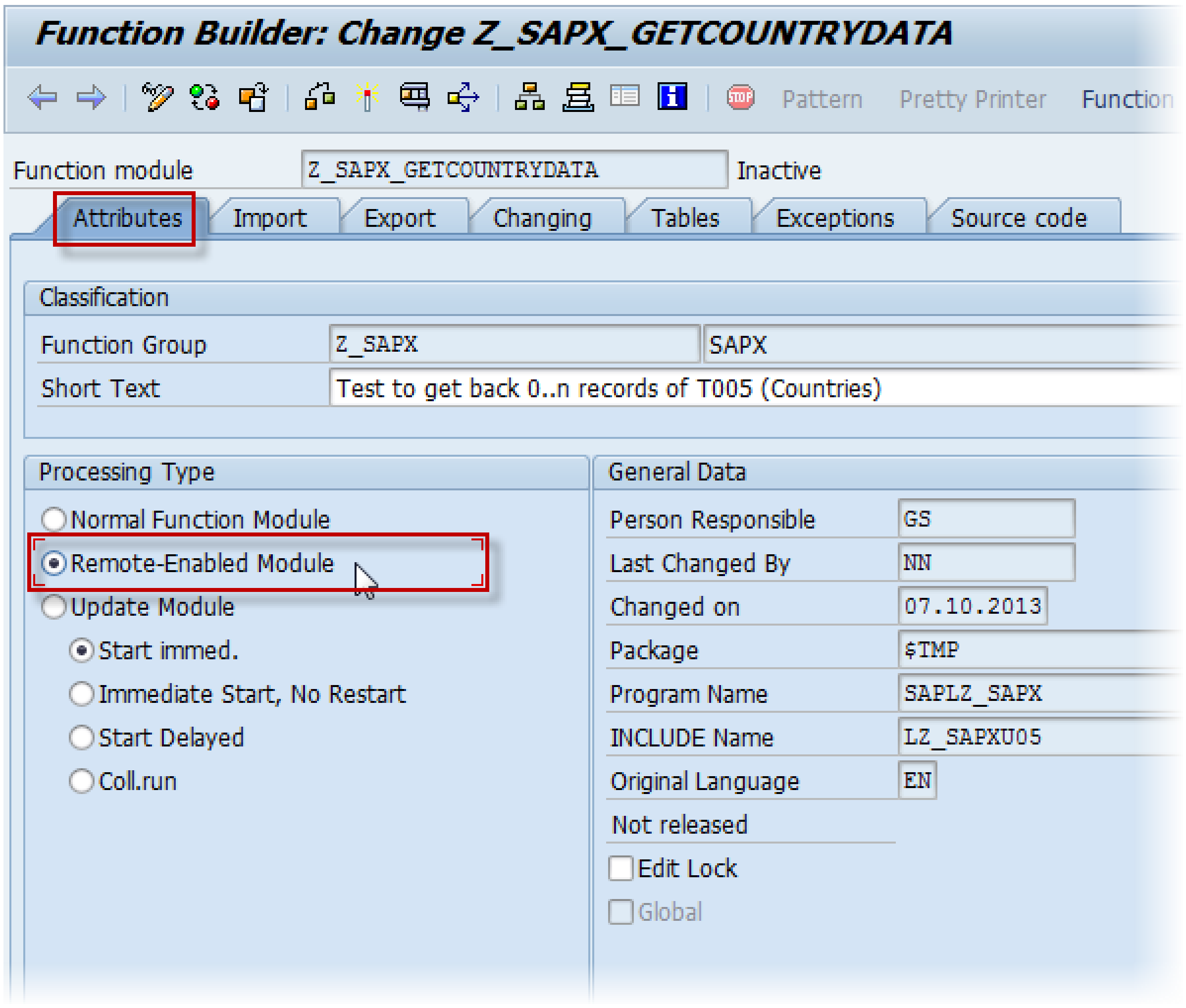


The Create Function Module window appears. Input a name of being created function module (using the prefix Z_SAPX_), an existing Function Group (see How to define a Function Group) if you want to create a new function group) and a short description. And press Save.



Function Module Z_SAPX_GETCOUNTRYDATA
Function group Z_SAPX
Short text Test to get back 0..n records of T005 (Countries)

2. The Function Builder window appears. Switch to the Attributes tab and set Processing Type to Remote-Enabled Module.


Function Builder: Change Z_SAPX_GETCOUNTRYDATA

Function module: Z_SAPX_GETCOUNTRYDATA Inactive

Classification

Function Group	Z_SAPX	SAPX
Short Text	Test to get back 0..n records of T005 (Countries)	

Processing Type

- ○ Normal Function Module
- ● Remote-Enabled Module
- ○ Update Module
  - ● Start immed.
  - ○ Immediate Start, No Restart
  - ○ Start Delayed
  - ○ Coll.run

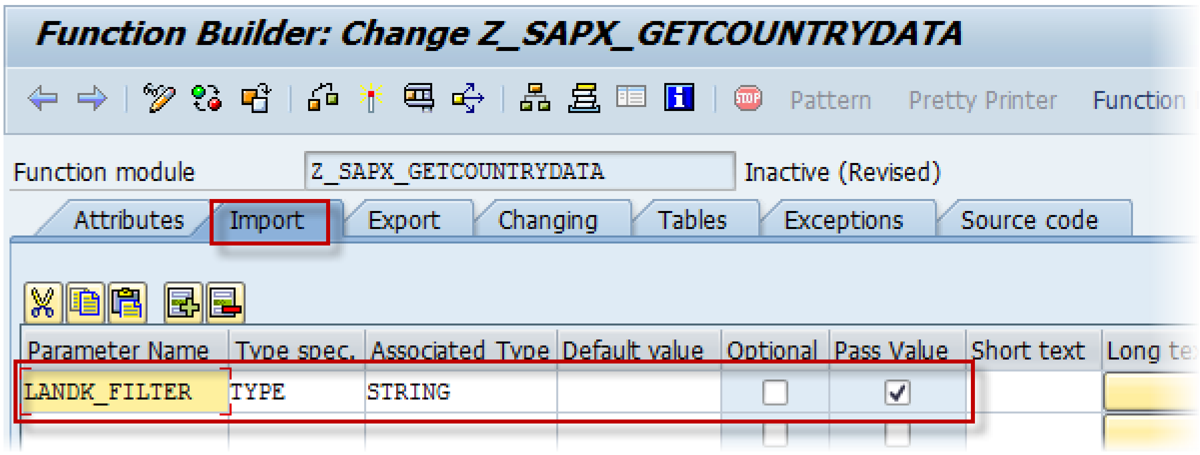
General Data

Person Responsible	GS
Last Changed By	NN
Changed on	07.10.2013
Package	$TMP
Program Name	SAPLZ_SAPX
INCLUDE Name	LZ_SAPXU05
Original Language	EN

Not released

☐ Edit Lock
☐ Global

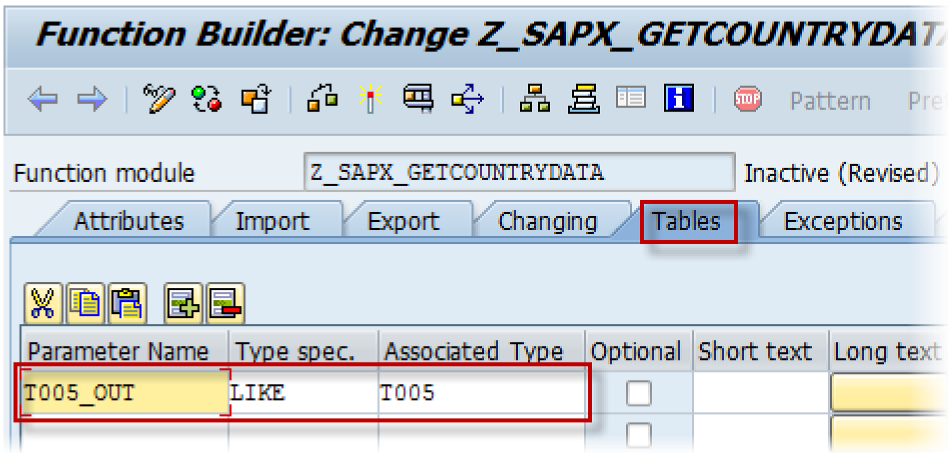
3. Switch to the Import tab and add a new parameter as shown below.


Function Builder: Change Z_SAPX_GETCOUNTRYDATA

Function module: Z_SAPX_GETCOUNTRYDATA Inactive (Revised)

Parameter Name	Type spec.	Associated Type	Default value	Optional	Pass Value	Short text	Long te
LANDK_FILTER	TYPE	STRING		☐	☑		

4. Switch to the Tables tab and add a new table as shown below.



Function module Z_SAPX_GETCOUNTRYDATA Inactive (Revised)

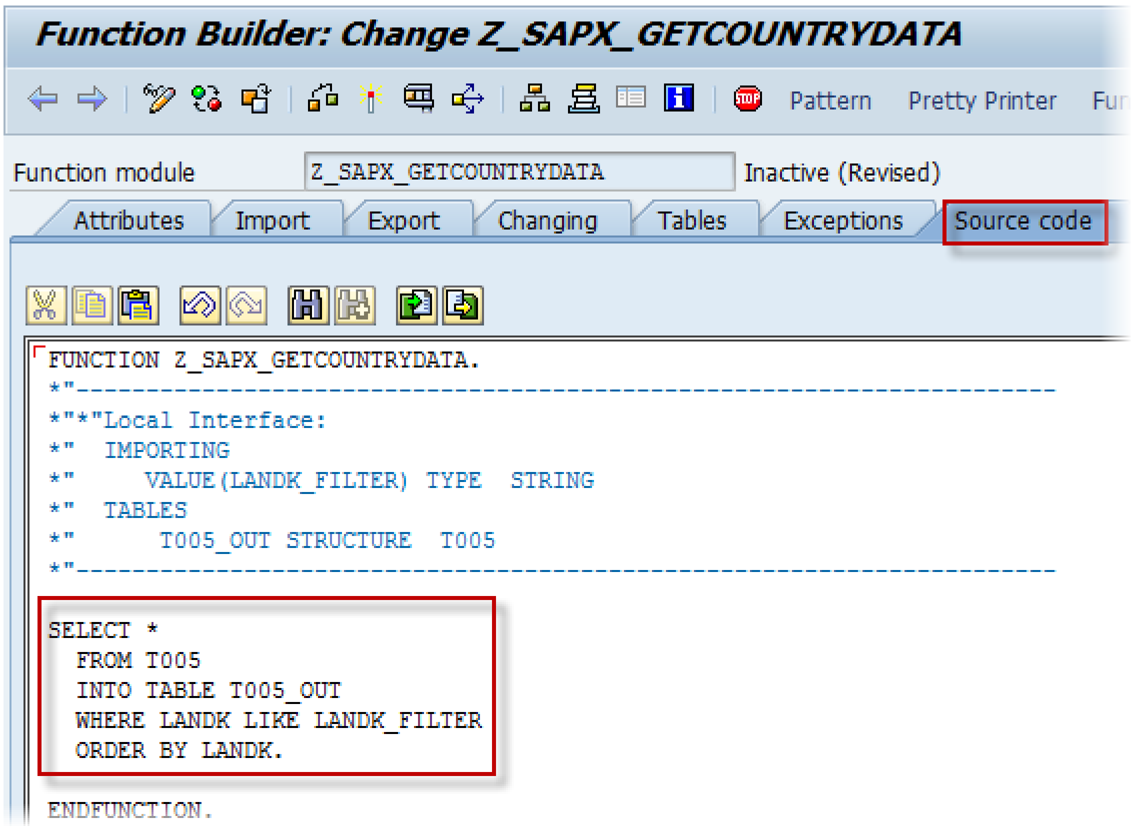
Parameter Name	Type spec.	Associated Type	Optional	Short text	Long text
T005_OUT	LIKE	T005	☐		
			☐		

5. Switch to the Source code tab and insert the next ABAP code:

Listing 3: Source code of Z_SAPX_GETCOUNTRYDATA

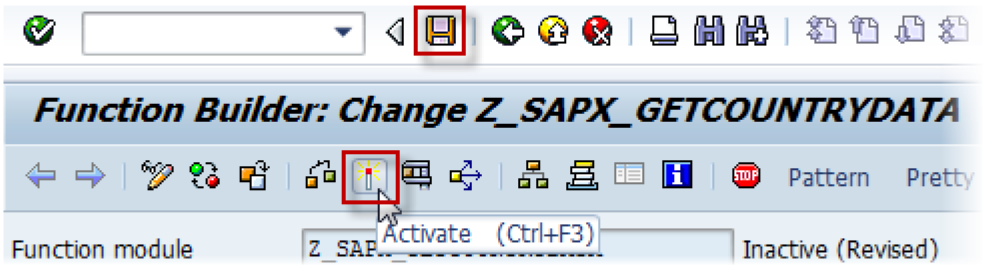
```
FUNCTION Z_SAPX_GETCOUNTRYDATA.
*"----------------------------------------------------------------------
*"*"Local Interface:
*"  IMPORTING
*"     VALUE(LANDK_FILTER) TYPE  STRING
*"  TABLES
*"       T005_OUT STRUCTURE  T005
*"----------------------------------------------------------------------

SELECT *
  FROM T005
  INTO TABLE T005_OUT
  WHERE LANDK LIKE LANDK_FILTER
  ORDER BY LANDK.

ENDFUNCTION.
```

After the insertion the window should look as shown below:

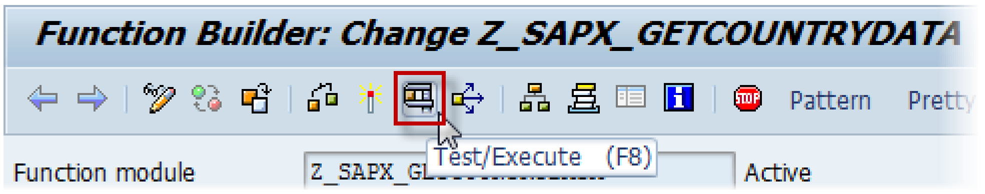


```
FUNCTION Z_SAPX_GETCOUNTRYDATA.
*"----------------------------------------------------------------------
*"*"Local Interface:
*"  IMPORTING
*"     VALUE(LANDK_FILTER) TYPE  STRING
*"  TABLES
*"      T005_OUT STRUCTURE  T005
*"----------------------------------------------------------------------

SELECT *
  FROM T005
  INTO TABLE T005_OUT
  WHERE LANDK LIKE LANDK_FILTER
  ORDER BY LANDK.

ENDFUNCTION.
```

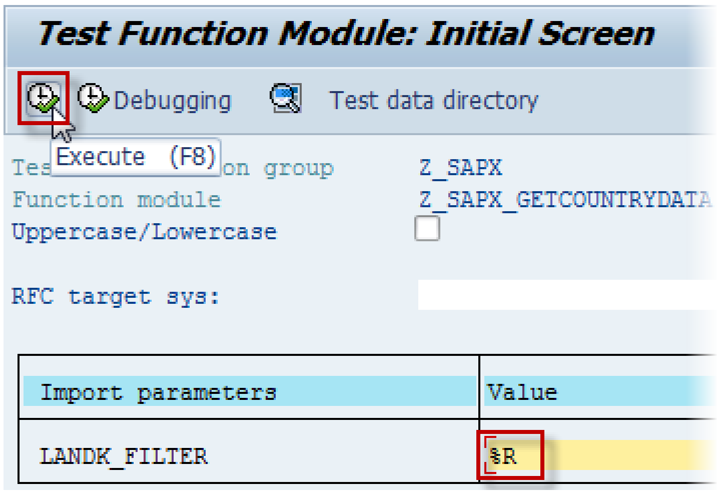
6. Press Save and Activate.



7. The function is ready to run. Press Test/Execute.



8. Specify a filter string as Import parameter and press Execute.

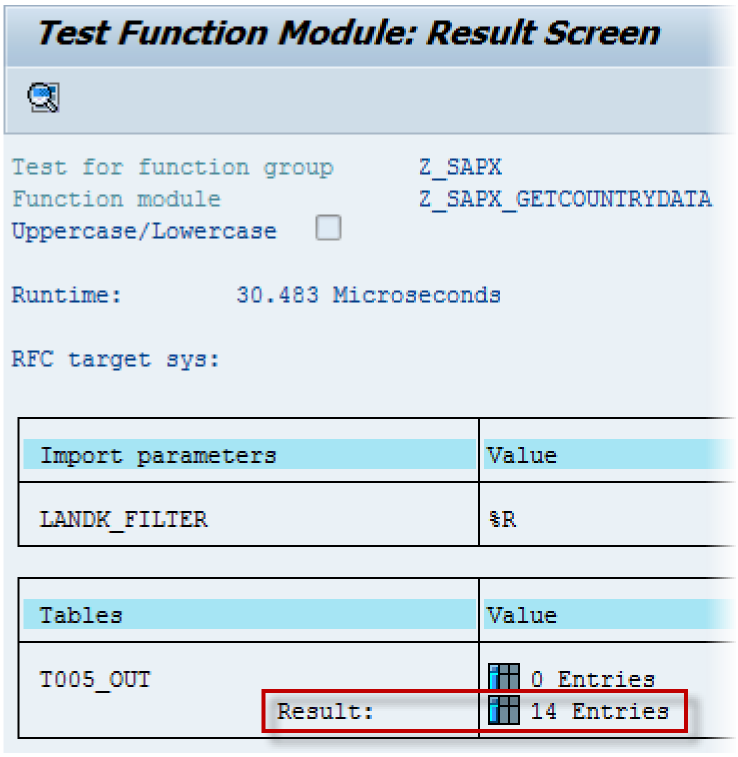


Test Function Module: Initial Screen

Test for function group	Z_SAPX
Function module	Z_SAPX_GETCOUNTRYDATA
Uppercase/Lowercase	☐
RFC target sys:	

Import parameters	Value
LANDK_FILTER	%R

9. During the execution, the table T005_OUT is filled by countries corresponding to the chosen filter.



Test Function Module: Result Screen

Test for function group	Z_SAPX
Function module	Z_SAPX_GETCOUNTRYDATA
Uppercase/Lowercase	☐
Runtime:	30.483 Microseconds
RFC target sys:	

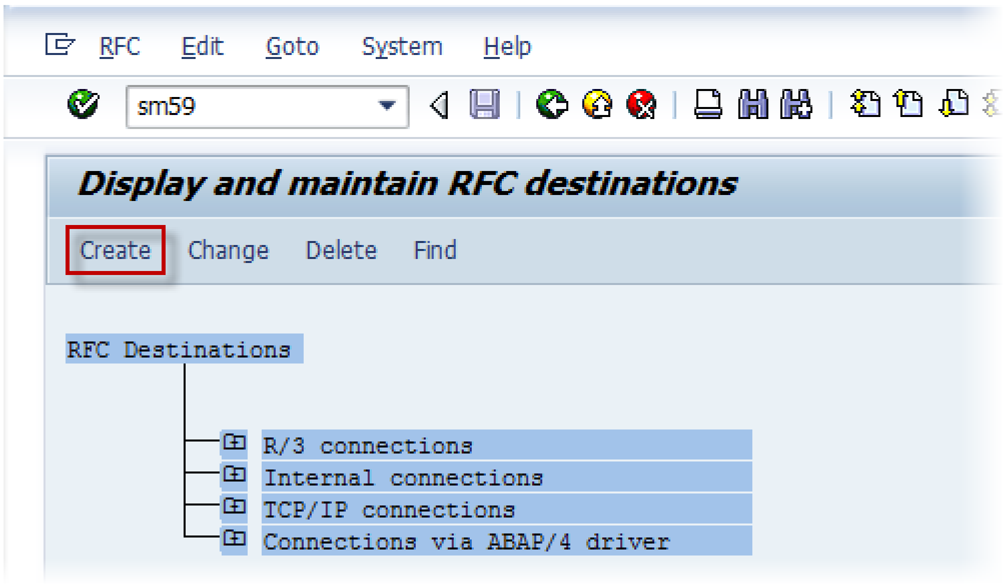
Import parameters	Value
LANDK_FILTER	%R

Tables	Value
T005_OUT	0 Entries
Result:	14 Entries

4.4 How to define a Server Destination [SM59]

4.4.1 Create an RFC destination

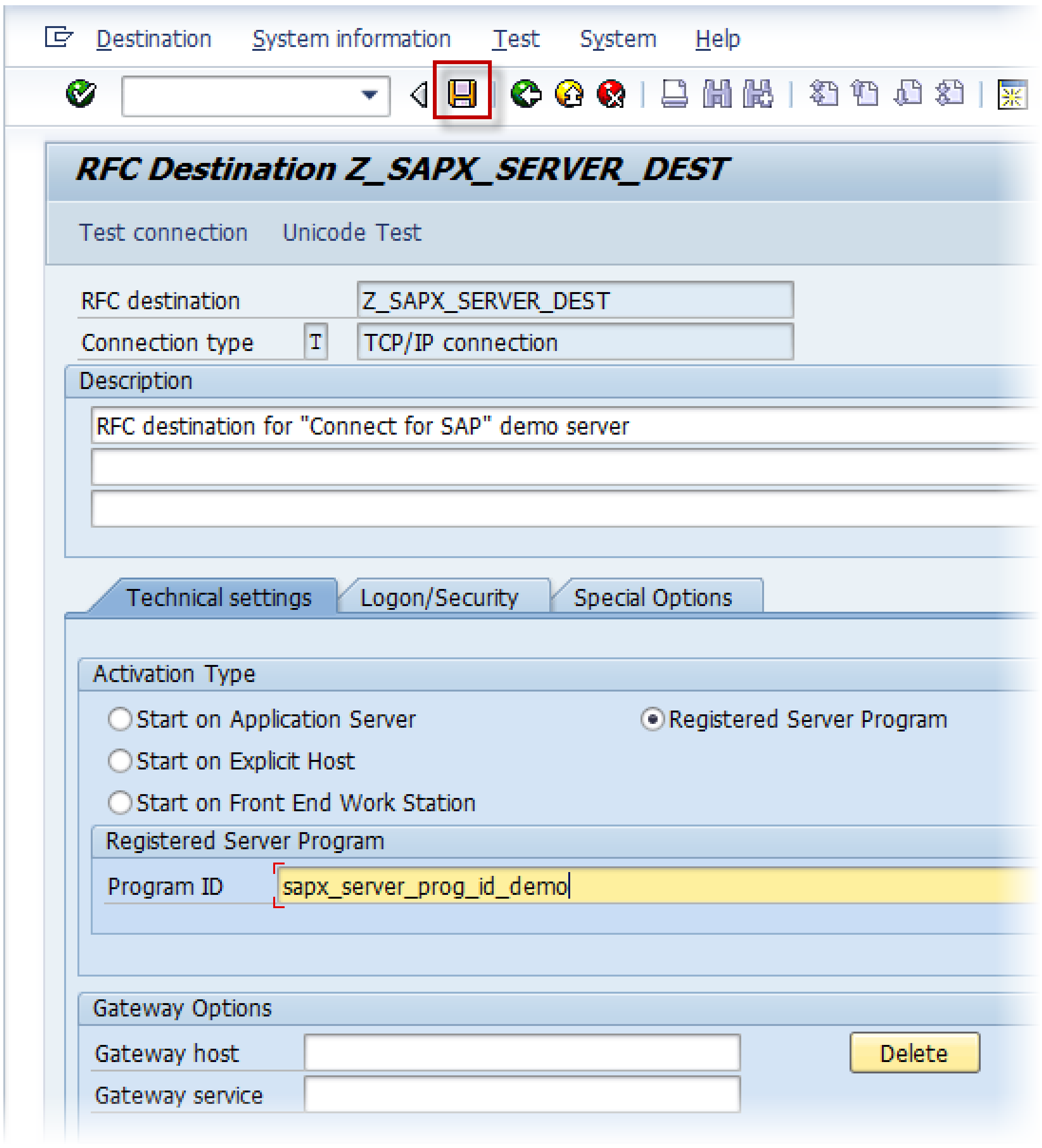
To create an RFC destination SAP system transaction SM59 (Display and maintain RFC destinations) is used.



Press Create and specify key parameters:

- RFC destination
- Connection type
- Activation type
- Program ID

Press Save button.



RFC Destination Z_SAPX_SERVER_DEST

RFC destination: Z_SAPX_SERVER_DEST
Connection type: T TCP/IP connection

Description
RFC destination for "Connect for SAP" demo server

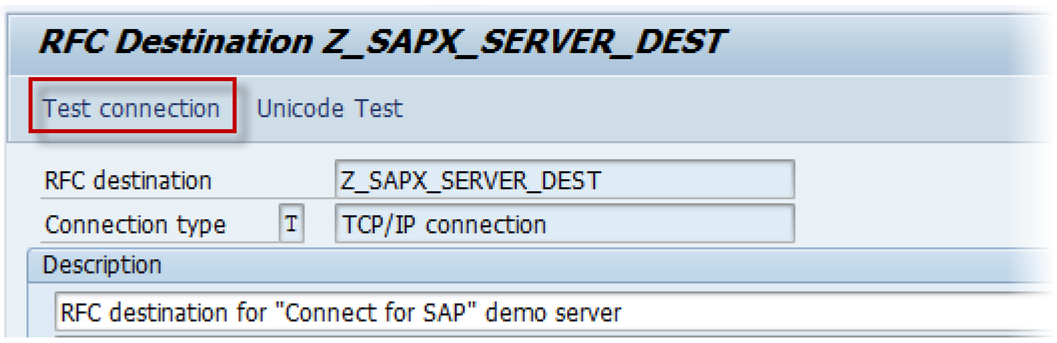
Activation Type
Start on Application Server [✓ Registered Server Program]
Start on Explicit Host
Start on Front End Work Station

Registered Server Program
Program ID: sapx_server_prog_id_demo

Gateway Options
Gateway host:
Gateway service:

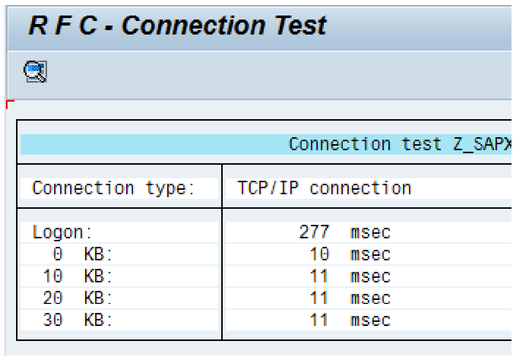
4.4.2 Test an RFC destination

Use SAP system transaction SM59 to test an existing RFC destination. Select an RFC destination you need to verify and press Test connection.



RFC destination: Z_SAPX_SERVER_DEST
Connection type: T TCP/IP connection
Description
RFC destination for "Connect for SAP" demo server

If the RFC destination is correct you will see the following report.



```
Connection test Z_SAPX
Connection type:   TCP/IP connection
Logon:                 277  msec
   0  KB:               10  msec
  10  KB:               11  msec
  20  KB:               11  msec
  30  KB:               11  msec
```

5 Appendix B: Connect for SAP® documents

The next table shows locations of Connect for SAP® documents:

Document	Location
Getting Started	SAPx\Docu\ Connect for SAP - Getting Started.pdf

Table 1. Locations of Connect for SAP® Documents