RFC Client

This page explains the **SAP Adapter** in Bridge context. If you were looking for the same information regarding the PAS Designer, refer to SAP Adapter in the Designer guide.

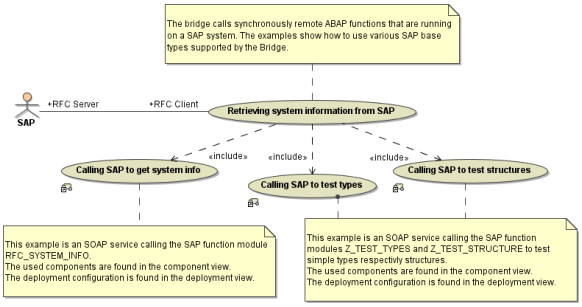
Calling SAP ABAP function modules is illustrated by an examples having to operations:

Example File (Builder project Add-ons/SAP):

<your example path>\Add-ons\SAP\uml\sapRFCClient.xml

- **Getting system info**: The example calls a function that exists on all SAP systems and retrieves basic system information.
- **Testing types**: The example tests simple types and structures by calling test functions in SAP. These functions are explained later on.

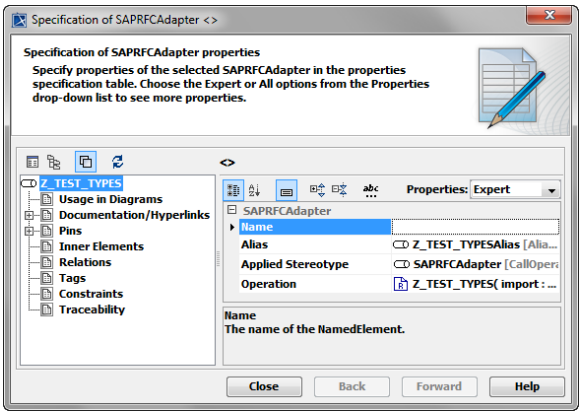
Figure: RFC Client Use Cases



Calling ABAP Functions via RFC

Calling ABAP functions via RFC is done in an Call Operation Action using the <<SAPRFCAdapter>>. Property **Operation** points to the operation of the <<SAPRFCModuleInterface>> that is being called by the adapter. The tagged value **alias** points to an alias that in turn is used in the component diagram to find the SAP system that is physically accessed.
The necessary classes and other UML elements can be generated with the help of the WSDL Generator for SAP RFC.

Figure: RFC Client Call Operation Action



The adapter interface follows the SAP ABAP conventions. Each SAP ABAP function has four parameter sections: import, export, changing and tables. In UML, these are mapped to the input and output parameters of the <<SAPRFCAdapter>>:

On this Page:

Related Pages:

- WSDL Generator for SAP RFC
- RFC Arguments
- tRFC Client
- SAP - ABAP Types Mappings
- IDoc Handling
- Frontend Components

Name	Type	Direction	Description
connectionString	**String**	in	Supplies the connection string (optional).
import	**Any**	in	The class specifying the type of this parameter must have stereotype <<SAPParameters>>. The attributes and associations of this class correspond to the parameters given by the **import** section of the ABAP function declaration – see figure Export parameters in SAP.
export	**Any**	out	The class specifying the type of this parameter must have stereotype <<SAPParameters>>. The attributes and associations of this class correspond to the parameters given by the **export** section of the ABAP function declaration
changing	**Any**	in/out	The class specifying the type of this parameter must have stereotype <<SAPParameters>>. The attributes and associations of this class correspond to the parameters given by the **changing** section of the ABAP function declaration
tables	**Any**	in/out	The class specifying the type of this parameter must have the <<SAPTables>>. The attributes and associations of this class correspond to the parameters given the **tables** section of the ABAP function declaration.

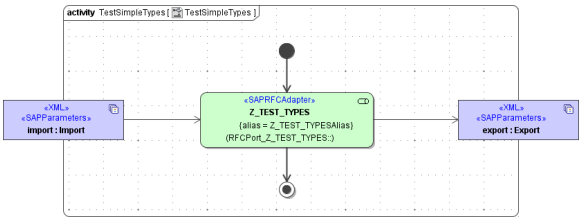
You can find more information about SAP parameters and tables on RFC Arguments.

Examples

Simple Types

The following example just sends and receives simple ABAP types. Inspecting the classes **Import** and **Export** you will see that they have simple typed attributes only.

Figure: Calling Z_TEST_TYPES

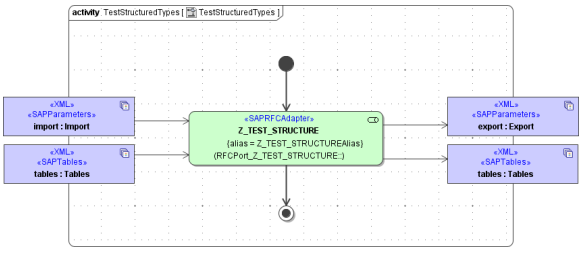


```
ABAP Function Z_TEST_TYPES:
 FUNCTION Z_TEST_TYPES.
 *"----------------------------------------------------------------------
 ""Local interface:
 *"  IMPORTING
 *"     VALUE(IP_INT) TYPE  I DEFAULT 0
 *"     VALUE(IP_DEC) TYPE  Z_DEC_TEST1 DEFAULT 0
 *"     VALUE(IP_FLT) TYPE  FLOAT DEFAULT 0
 *"     VALUE(IP_CHAR) TYPE  Z_CHAR_TEST1 DEFAULT 'abcdefghgh'
 *"     VALUE(IP_DAT) TYPE  Z_DATE_TEST1 DEFAULT '20021213'
 *"     VALUE(IP_TIM) TYPE  Z_TIME_TEST1 DEFAULT '130723'
 *"     VALUE(IP_RAW) TYPE  Z_RAW_TEST1 DEFAULT 0
 *"     VALUE(IP_SMALL_DEC) TYPE  Z_SMALL_DEC_TEST1 DEFAULT 0
 *"     VALUE(IP_NUMC) TYPE  Z_NUMC_TEST DEFAULT 0
 *"     VALUE(IP_LCHR) TYPE  Z_LCHR_TEST DEFAULT 0
 *"  EXPORTING
 *"     VALUE(EP_INT) TYPE  I
 *"     VALUE(EP_DEC) TYPE  Z_DEC_TEST1
 *"     VALUE(EP_FLT) TYPE  FLOAT
 *"     VALUE(EP_CHAR) TYPE  Z_CHAR_TEST1
 *"     VALUE(EP_DAT) TYPE  Z_DATE_TEST1
 *"     VALUE(EP_TIM) TYPE  Z_TIME_TEST1
 *"     VALUE(EP_RAW) TYPE  Z_RAW_TEST1
 *"     VALUE(EP_SMALL_DEC) TYPE  Z_SMALL_DEC_TEST1
 *"     VALUE(EP_NUMC) TYPE  Z_NUMC_TEST
 *"     VALUE(EP_LCHR) TYPE  Z_LCHR_TEST
 *"----------------------------------------------------------------------

   EP_INT = IP_INT + 1.
   EP_DEC = IP_DEC + '1.0'.
   EP_FLT = IP_FLT + '1.0'.
   EP_CHAR = IP_CHAR.
   EP_DAT = IP_DAT.
   EP_TIM = IP_TIM.
   EP_RAW = IP_RAW.
   EP_SMALL_DEC = IP_SMALL_DEC + '1.0'.
   EP_NUMC = IP_NUMC.
   EP_LCHR = IP_LCHR.

 ENDFUNCTION.
```

Complex Types

In contrast to the previous example, the activity diagram **TestStructuredTypes** handles complex parameters, so-called SAP structures. Anyway, in activity diagrams both structured and simple parameters are handled alike. The difference shows only in the class diagram where simple and structured parameters correspond to simple respectively complex attributes of the importing and exporting classes.

Figure: Calling Z_TEST_STRUCTURE

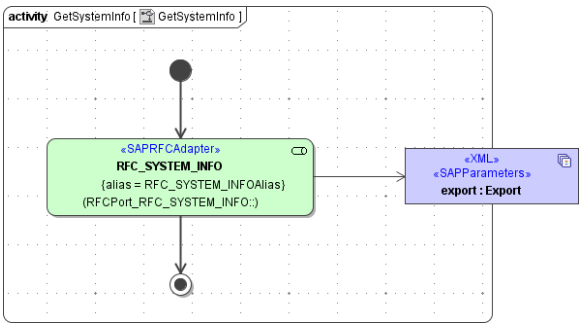


```
ABAP Function Z_TEST_STRUCTURE:
 FUNCTION Z_TEST_STRUCTURE.
 *"----------------------------------------------------------------------
 ""Local interface:
 *"  IMPORTING
 *"     VALUE(IP_STRUCT) LIKE  ZTSTRUCT STRUCTURE  ZTSTRUCT
 *"  EXPORTING
 *"     VALUE(EP_STRUCT) LIKE  ZTSTRUCT STRUCTURE  ZTSTRUCT
 *"----------------------------------------------------------------------

   EP_STRUCT-INT_ATTRIBUTE = IP_STRUCT-INT_ATTRIBUTE.
   EP_STRUCT-DEC_ATTRIBUTE = IP_STRUCT-DEC_ATTRIBUTE.
   EP_STRUCT-FLT_ATTRIBUTE = IP_STRUCT-FLT_ATTRIBUTE.
   EP_STRUCT-CHAR_ATTRIBUTE = IP_STRUCT-CHAR_ATTRIBUTE.
   EP_STRUCT-DAT_ATTRIBUTE = IP_STRUCT-DAT_ATTRIBUTE.
   EP_STRUCT-TIM_ATTRIBUTE = IP_STRUCT-TIM_ATTRIBUTE.
   EP_STRUCT-RAW_ATTRIBUTE = IP_STRUCT-RAW_ATTRIBUTE.
   EP_STRUCT-LCHR_ATTRIBUTE = IP_STRUCT-LCHR_ATTRIBUTE.

 ENDFUNCTION.
```

System Info

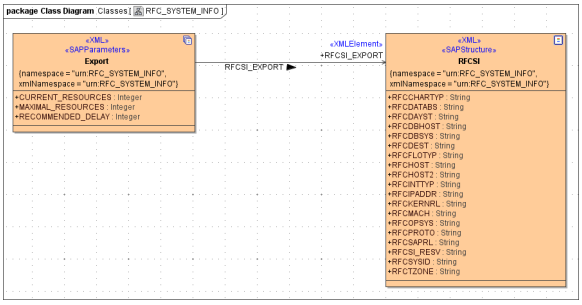
Another example of using SAP structure arguments in RFC calls is the RFC_GET_SYSTEM_INFO function. This function is available in every SAP system. It returns system information like exact version of the SAP system, used codepages, language settings, and so on. It is useful to test connectivity and retrieve basic system information. The following activity diagram shows how to call RFC_GET_SYSTEM_INFO. This activity diagram implements the **GetSystemInfo** operation of the **SystemInfoPort** class.

Figure: Calling RFC_SYSTEM_INFO



The SAP function RFC_SYSTEM_INFO returns only one parameter: **RFCSI_EXPORT** of type **RFCSI**. The association end **RFCSI_EXPORT** respectively the <<SAPStructure>> class **RFCSI** corresponds to these two entities:

Figure: RFC_SYSTEM_INFO output.



For details on modeling SAP structures see Parameters.

RFC Client Components

The example above calls ABAP functions in a SAP system. The frontend interface is SOAP. Therefore, the composite holds a SOAP service component (**RFCClientService**). This component contains two SOAP Port Types:

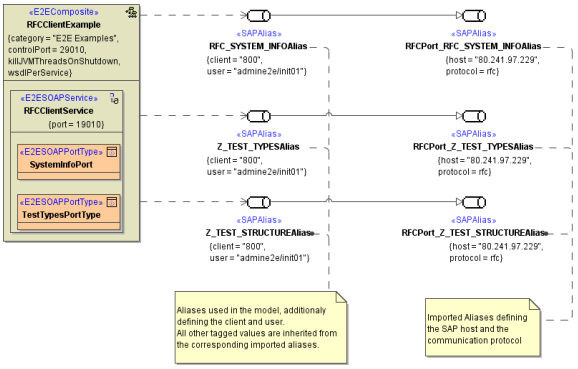
- **SystemInfoPort**: This class holds an operation that retrieves system information from the SAP system by calling **RFC_SYSTEM_INFO**.The operation **GetSystemInfo** returns the system data as SAP structure. Its implementation is found here.
- **TestTypesPort**: This class holds two operations, one for testing simple ABAP type supported by the Bridge, and one for testing SAP structures. They call the SAP ABAP functions **Z_TEST_TYPES** respectively **Z_TEST_STRUCTURE**. The activity diagrams that actually call these ABAP functions are TestSimpleTypes and TestStructuredTypes.

The composite **RFCClientExample** contains a SOAP service (**RFCClientService**) accessing the SAP system. The SAP system and the accessed SAP interfaces are modeled as aliases having the stereotype <<SAPAlias>> .

The SAP aliases are created automatically on importing the WSDL of the SAP function. Like for every other adapter, the SAP adapter can be configured by the tagged values on the alias. As you can not change the imported aliases to configure the client and user tagged values, create new aliases and define a generalization between the imported alias and the new one. Thus, the new alias will inherit all imported values (such as host and protocol) and you can make your adjustments to it. On re-import of the WSDL, all changes to the imported aliases will be passed through directly.

The tagged value definitions used for RFC client protocol are the same as for tRFC communication as explained in the tRFC Client Deployment diagram (see section tRFC Client Components). The tagged values itself do not differ with one exception: the **protocol** tagged value must be rfc.

Figure: RFC Client Component Diagram



On the composite, you can also set a service-wide **SAP value padding**: Never, Always and Mixed. See Frontend Components for more information.