

tRFC Service

The most common use case for tRFC servers is receiving IDocs from SAP. The tRFC server example implements an xUML service that makes the Bridge acting as a tRFC server and shows how to receive IDocs from SAP.

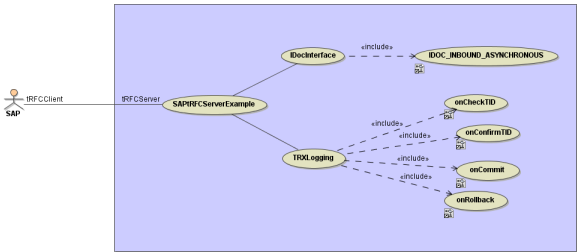
Example File (Builder project Add-ons/SAP):



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

The difference to RFC, tRFC implements an additional transaction handling. This transaction handling must guarantee that a transaction is executed exactly once. The following simple example shows how to use the transaction mechanism offered by SAP. The proper transaction control is implemented elsewhere (cf. in the SAP module).

Figure: tRFC Service Use Case Diagram



The operation **IDOC_INBOUND_ASYNCHRONOUS** gets called when SAP is transmitting an IDoc to the Bridge. The implementation of this operation can be found in the [next section](#).

The transaction handling is separated from the actual exchange of data and must be handled by implementing all of the following callback operations:

Callback Operation	Description	Return Value	
onCheckTID	This function will be activated, if a transactional RFC is called from an SAP system. The current TID (Transaction ID) has been handed over to the function. - The function has to store this TID in permanent storage and return 0. - If the same TID is called again later, the function must return a value <> 0. - If a transaction with the same TID has already been started by another process, and is not completed yet, the function has to wait until this transaction is completed.	0	ok
		not 0	TID already in use
onCommit	This function is called, if all RFC functions belonging to this transaction are done and the local transaction can be completed. It should be used to locally commit the transaction, if working with database.		
onRollback	This function is called instead of onCommit , if an error occurred in the RFC library while processing the local transaction. This function can be used to roll back the local transaction (working with database).		
onConfirmTID	This function is called, if the local transaction is completed. All information about this TID can be deleted.		

Implementing tRFC Operations

The tRFC function probably the most often implemented is **IDOC_INBOUND_ASYNCHRONOUS**. It is called when SAP sends interface documents (IDocs). This section explains the implementation of tRFC functions by giving a simple example implementation of **IDOC_INBOUND_ASYNCHRONOUS**. In other words, we build a simple IDoc server.

On this Page:

- Implementing tRFC Operations
- tRFC Service Components

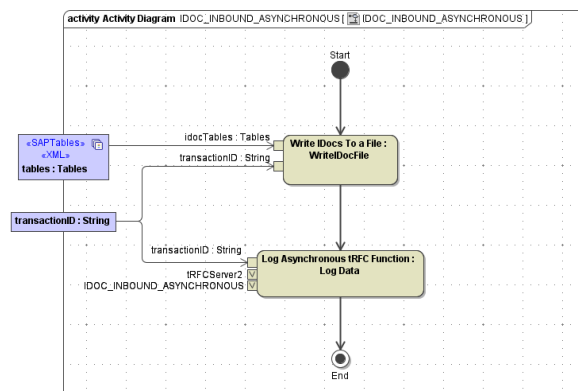
Related Pages:

- RFC Arguments
- RFC Service
- Frontend Components

In general, each RFC function can have the following parameters: import, export, changing and tables. For asynchronous tRFC functions like **IDOC_INBOUND_ASYNCHRONOUS** the parameters are restricted to **import** and **tables**. The types assigned to the import and table parameters must have the stereotype <<SAPParameters>> respectively <<SAPTables>> . For more details regarding these see [RFC Arguments](#) .

However, IDOC_INBOUND_ASYNCHRONOUS uses table parameters only. This function will be called, if the SAP system sends IDocs to an (t)RFC server. The IDoc is delivered as two tables. Both tables are associations of the IDOC_TABLES class (see figures at [RFC Arguments](#)). For details regarding IDOC_INBOUND_ASYNCHRONOUS, refer to the SAP documentation. The current example implementation just writes the IDocs to the file system and writes a notification to the service log.

Figure: IDOC_INBOUND_ASYNCHRONOUS



Only inbound data flow is allowed for SAP tRFC functions. That is tables and importing parameters. Additionally, when implementing a tRFC function, the modeler can access the current transaction ID via the **transactionID** input parameter.

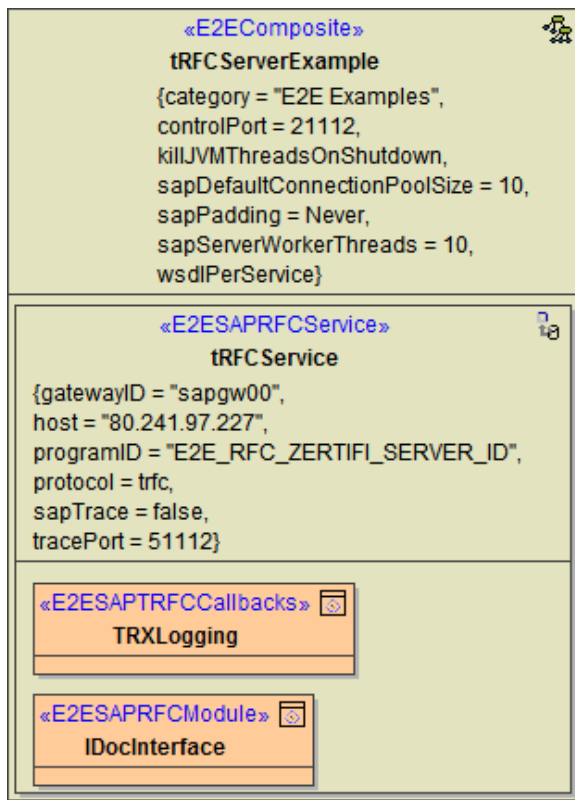
tRFC Service Components

Each <<E2ESAPRFCService>> that uses tRFC contains at least two residents:

- One and only one <<E2ESAPTRFCCallbacks>> class holding operations to control transactional behavior (for instance the TRXLogging class beneath).
- One or more <<E2ESAPRFCModule>> classes containing operations called by the SAP system to exchange data.

For instance, see the **IDocInterface** class depicted in the component diagram beneath. The implementation of these operations is similar to the implementation of standard RFC operations. The only difference results from the asynchronous nature of tRFC operations - they handle input only.

Figure: tRFC Service Component Diagram



The **tRFCService** has the following tagged values:

Tagged Value	Description	Mandatory /Optional
<<E2EComposite>>		
sapDefaultConnectionPoolSize	Default capacity of a single SAP connection pool (Bridge acting as a SAP client). If undefined, a default of 10 connections will be applied. Each distinct connection to a SAP system has its own pool. Connections are distinguished by the set of connection parameters (connection string). You can override the connection pool size for a specific connection on the corresponding SAP alias. On using dynamic SAP access, the default connection pool size is used.	optional
sapPadding	Service-wide setting for SAP values padding. This setting will be applied to all IDoc and SAP adapters within the service. i It is not recommended to use Mixed padding. This option is only available for reasons of backwards compatibility. Mixed padding is default for older services that have been compiled before the implementation of this tagged value, whereas Never is default, if no SAP padding is specified.	optional
sapServerWorkerThreads	Number of parallel request (workers) the Bridge (acting as an RFC server) can process. If this value is undefined, the Bridge will only process one request at a time (equivalent to sapServerWorkerThreads=1). i Each active worker requires one license slot (concurrent connection). For more information on licensing and concurrent connections, refer to License for Running xUML Services.	optional
<<E2ESAPRFCService>>		
host	Optional gateway host name. Default is localhost .	optional

gatewayID	The port number of the SAP gateway.	mandatory
programID	The programID to which the service is registered on the gateway.	mandatory
protocol	Must be trfc when using the tRFC protocol. Must be rfc when using the RFC protocol.	mandatory
sapTrace	The effect of this flag being true is two fold: First, the SAP RFC libraries will write trace file information (.trc) into the directory the configuration has been deployed to. Second, by using the SAP transaction *SMGW (SAP gateway monitor) we can monitor the dataflow from and to the gateway the server is registered on.	optional
tracePort	The same operations that are called by the SAP system can also be called by SOAP test tool. However, the SOAP test tool requires an HTTP TCP/IP port. This port can be defined in the tracePort tagged value. If this value is not set, the trace port is given by controlPort + 50000.	optional

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

Basically, tRFC server components must be registered at the so-called SAP gateway. For example, component **tRFCService** is registered at SAP gateway port **3300** via program ID **E2E_RFC_ZERTIFI_SERVER_ID**.

The program ID must be defined on the SAP system in transaction **SM59**. This transaction enables the user to define TCP/IP connections (connection type **T**). Each such connection has an associated program ID that must be exactly the same string as used for the tagged value **programID** on the [<<E2ESAPRFCService>>](#) component. Note, that this string is case sensitive. On the [<<E2ESAPRFCService>>](#) component, you also specify the port number of the SAP gateway the server registers on.

The Model Compiler throws an error, if a composite service contains more than one [<<E2ESAPRFCService>>](#) component. This constraint is given by the very nature of a SAP gateway. Each operating system process must register exactly one program ID at the gateway, otherwise the dispatching of SAP requests to the RFC service is not unique. Now, each xUML service runs in its own Bridge process, thus unique dispatching can only be achieved by allowing only one RFC service - i.e. one program ID - per xUML service.