Using the RPC Communication Area

This chapter explains how clients use the RPC communication area and covers the following topics:

- Purpose of the RPC Communication Area
- Using the RPC Communication Area with a Standard Call Interface
- Using the RPC Communication Area with EXEC CICS LINK

The RPC communication area is not relevant for servers.

Purpose of the RPC Communication Area

The RPC communication area is mainly used to specify parameters that are needed to communicate with the broker and are not specific to client interface objects. In this way it defines a context for PRC clients. Its purpose, among others, is

- to assign the `COMM-ETB-BROKER-ID` and server name, see `COMM-ETB-SERVER-CLASS`, `COMM-ETB-SERVER-NAME` and `COMM-ETB-SERVICE-NAME`
- to assign the broker's `COMM-ETB-USER-ID` and `COMM-ETB-TOKEN`
- for use with conversational RPC (see *Using Conversational RPC*) to hold, for example, the conversation ID, see `COMM-ETB-CONV-ID`
- for use with EntireX Security to hold the broker's `COMM-ETB-PASSWORD`, `COMM-ETB-SECURITY-TOKEN` and others
- to keep the results of the last RPC request, for example the error code

The RPC communication area is also the API to the generic RPC services, for example:

- Log on to broker and log off from broker. See *Using Broker Logon and Logoff*.
- Open conversation, close conversation and close conversation with commit. See *Using Conversational RPC*.
- When using reliable RPC function calls, do reliable RPC commit, do reliable RPC rollback, get reliable status. See *Reliable RPC for COBOL Wrapper*.
- Create a Natural Security token. See *Using the COBOL Wrapper with Natural Security and Impersonation*.

From a COBOL point of view, the RPC communication area is the copybook `ERXCOMM`. It is generated in the folder *include* for RPC client generation, see *Generating COBOL Source Files from Software AG IDL Files*.

The layout of the RPC communication area is described in section *The RPC Communication Area (Reference)*.

Using the RPC Communication Area with a Standard Call Interface

The COBOL Wrapper allows the RPC communication to be used in the following ways:

- Option External Clause
- Option Linkage Section
- Option Copybook

Option External Clause

With the RPC communication area option `External Clause` under *RPC Communication Area*, the RPC communication area is passed using the COBOL External clause to the client interface objects. Note that this is an extension to COBOL 85 standards, which might not be supported by every compiler.

The RPC communication area is allocated (declared) in the COBOL client application. The client interface objects are statically linked (it is not possible to call them dynamically) to the COBOL client application.

This kind of RPC communication area usage applies to the scenarios *Micro Focus* | *Batch* | *CICS* | *IMS*.

Examples

For examples on how the option `External Clause` is used, see *Step 1: Declare and Initialize the RPC Communication Area* and *Step 5: Issue the RPC Request* in *Writing Standard Call Interface Clients*.

Option Linkage Section

With the RPC communication area option `Linkage Section` under *RPC Communication Area*, the client interface objects are generated to pass the RPC communication area with an additional parameter to the client interface objects.

The RPC communication area is allocated (declared) in the COBOL client application in the working storage section. The client interface objects can be statically linked or called dynamically. For IBM compilers, refer to documentation on the DYNAM compiler option; for other compilers, to your compiler documentation.

This kind of RPC communication area usage applies to the scenarios *Micro Focus* | *Batch* | *CICS* | *IMS*.

Example

The example given below will pass the RPC communication area via the COBOL Linkage section to the client interface objects. It differs in two steps from the example in *Writing Standard Call Interface Clients* (which uses option `External Clause`):

Step 1 has no EXTERNAL attribute.

```
 01 ERX-COMMUNICATION-AREA.
     COPY ERXCOMM.
* Initialize RPC communication area
 INITIALIZE ERX-COMMUNICATION-AREA.
 MOVE "2000" TO COMM-VERSION.
```

Step 5 will include the RPC communication area as an extra parameter.

```
 CALL "CALC" USING  OPERATOR
                    OPERAND1
                    OPERAND2
                    FUNCTION-RESULT
                    ERX-COMMUNICATION-AREA
   ON EXCEPTION
*       Perform error-handling
   NOT ON EXCEPTION
      IF RETURN-CODE = ZERO
*        Perform success-handling
      ELSE
*        Perform error-handling
      END-IF
 END-CALL.
```

With this example the client interface objects are generated, for example for target platform "z/OS", client interface type "Batch with standard linkage calling convention" and RPC communication area "Linkage Section". See *Generating COBOL Source Files from Software AG IDL Files*.

Option Copybook

With the RPC communication area option `Copybook` under *RPC Communication Area*, the client interface objects are generated with an RPC communication area in their working storage section.

The RPC communication area is not visible in the client application – it is local to the client interface objects. The client interface objects can be statically linked or called dynamically. For IBM compilers, refer to documentation on the DYNAM compiler option and for other compilers to your compiler documentation.

This kind of RPC communication area usage is available in z/OS operating system and Micro Focus environments. Refer to the scenarios *Micro Focus* | *Batch* | *CICS* | *IMS*.

Example

The example given below defines the RPC communication area inside of the client interface objects. Two steps are different from the example in *Writing Standard Call Interface Clients* (which uses option `External Clause`):

Step 1: Declare and Initialize the RPC Communication Area: Declare and initialize the RPC communication area

This step is obsolete in the client application and is omitted there. Default values for the RPC communication area are retrieved from EntireX workbench preferences or IDL-specific properties. If required, those default values can be overwritten in the *`COBINIT` Copybook*.

Step 6: Examine the Error Code: Examine the error code

Because the RPC communication area is not used for data exchange between the client application and the client interface objects, the `COMM-RETURN-CODE` field in the RPC communication area cannot be checked directly upon return from RPC calls. Therefore, the COBOL mechanism `RETURN-CODE` special register is used to provide errors from client interface objects to the client application. For IBM compilers, errors can be adapted in the copybook `COBEXIT` (see folder *include*).

After the RPC reply has been received, you can check if the call was successful using the `RETURN-CODE` special register:

```
 IF RETURN-CODE IS = ZERO
*    Perform success-handling
 ELSE
*    Perform error-handling
 END-IF.
```

Using the RPC Communication Area with EXEC CICS LINK

The RPC communication area is allocated (declared) in the COBOL client application and passed via a parameter in the DFHCOMMAREA to the client interface objects.

This kind of RPC communication area usage applies to the scenario *Using the COBOL Wrapper for CICS with `DFHCOMMAREA` Calling Convention (z/OS and z/VSE)*.

Example

Two steps are different from the example in *Writing a COBOL RPC Client Application* See *Writing Standard Call Interface Clients*.

Step 1 contains the application interface as well as the RPC communication area within one area:

```
01  CALC-AREA.
  05 OPERATOR                   PIC X.
  05 OPERAND1                   PIC S9(8) COMP.
  05 OPERAND2                   PIC S9(8) COMP.
  05 RESULT                     PIC S9(8) COMP.
  05 ERX-COMMUNICATION-AREA.
  COPY ERXCOMM.
* Initialize RPC communication area
 INITIALIZE ERX-COMMUNICATION-AREA.
 MOVE "2000" TO COMM-VERSION.
```

Step 5 uses *EXEC CICS LINK interface*:

```
 MOVE LENGTH OF CALC-AREA TO COMLEN.
 EXEC CICS LINK PROGRAM("CALC") COMMAREA(CALC-AREA)
           LENGTH(COMLEN) RESP(WORKRESP)
 END-EXEC.
 IF WORKRESP = DFHRESP(NORMAL)
    IF (COMM-RETURN-CODE = 0) THEN
*       Perform success-handling
    ELSE
*       Perform error-handling
```

```
    END-IF
 ELSE
*    Perform error-handling
 END-IF.
```

With this example, the client interface objects are generated e.g. for target platform "z/OS", client interface type "CICS with DFHCOMMAREA Calling Convention", and RPC communication area "Linkage Section". See *Generating COBOL Source Files from Software AG IDL Files*.