Writing an RPC Client Application with the PL/I Wrapper

This chapter is a step-by-step guide for writing your first PL/I RPC client program.

- Step 1: Generic Declarations Required by the PL/I Wrapper
- Step 2: Declare the (Generated) Data Structures for (Generated) Interface Objects
- Step 3: Declare ENTRY Definitions to (Generated) Interface Objects
- Step 4: Required Settings in the RPC Communication Area
- Step 5: Optional Settings in the RPC Communication Area
- Step 6: Issue the RPC Request
- Step 7: Examine the Error Code

The example given here does not use function calls as described under *Using Broker Logon and Logoff*. It demonstrates an implicit broker logon (because no broker logon/logoff calls are implemented), where it is required to switch on the `AUTOLOGON` feature in the broker attribute file.

The following steps describe how to write a PL/I RPC client program. We recommend reading them first before writing your first RPC client program and following them if appropriate.

Step 1: Generic Declarations Required by the PL/I Wrapper

Step 1a: Embed PL/I Wrapper Preprocessor Definitions

The Preprocessor is always needed. Always embed `RPCPPD` and take care to set the correct values for your environment in the *PL/I Preprocessor Settings*.

```
%include  RPCPPD;
```

Step 1b: Declare PL/I Built-in Functions

These built-in functions are needed to communicate with the *Using the Generic RPC Services Module* and the generated RPC stubs:

```
DECLARE STORAGE        built in;
DECLARE SUBSTR         built in;
```

Step 1c: Declare API Constants to PL/I Wrapper

This delivered include file defines constants and generic definitions to the PL/I Wrapper:

```
/* RPC API Interface */
%include  RPCAPI;
```

Step 1d: Declare and Initialize the RPC Communication Area

Declare and initialize the *The RPC Communication Area (Reference)* in your RPC client program as follows:

```
/* Declare RPC communication area */
DECLARE 1 ERXCOM,
%include  RPCCOM; /* RPC communication area fields */

/* Initialize RPC communication area */
ERXCOM = '';
ERXCOM.COM_VERSION = ERX_COM_VERSION_1;
ERXCOM.COM_SIZE    = STORAGE(ERXCOM);
```

Step 2: Declare the (Generated) Data Structures for (Generated) Interface Objects

For every program definition of the IDL file, the templates generate an include file that describes the customer data of the interface as a PL/I structure. For ease of use, you can embed these structures into your RPC client program:

```
/* Declare customer data to generated interface objects */
%include CALC;
/* RESULT as a local variable because of function call */
DCL RESULT BIN FIXED (31);
```

However, if more appropriate, you can use your own customer data structures. In this case the PL/I data types and structures must match the interfaces of the generated interface objects, otherwise unpredictable results may occur.

Step 3: Declare ENTRY Definitions to (Generated) Interface Objects

This step is appropriate for `TARGET BATCH_xxx` only. For `TARGET CICS_xxx`, no `ENTRY` declarations are generated, because communication with the interface objects is through the CICS `COMMAREA`, where `ENTRY` declarations are not suitable.

For `TARGET BATCH_xxx`, the templates generate for every library-definition of the IDL file, an include file containing the `ENTRY` declarations to your client interface objects. We recommend embedding them into your RPC client program:

```
/* Declare ENTRY definitions to generated interface objects */
%include EXAMPLE;
```

Step 4: Required Settings in the RPC Communication Area

The following settings to the RPC communication area are required as a minimum to use the PL/I Wrapper. These settings have to be applied in your RPC client program. No defaults are generated into your interface objects:

```
/* assign the broker to talk with ... */
ERXCOM.COM_BROKER_ID      = 'ETB001';

/* assign the server to talk with ... */
ERXCOM.COM_SERVER_CLASS   = 'RPC';
ERXCOM.COM_SERVER_NAME    = 'SRV1';
ERXCOM.COM_SERVER = 'CALLNAT';

/* assign the user id to the broker ... */
ERXCOM.COM_CLIENT_USERID  = 'PLI-USER';
```

Step 5: Optional Settings in the RPC Communication Area

Here you specify optional settings to the RPC communication area used by the PL/I Wrapper, for example:

```
ERXCOM.COM_CLIENT_PASSWORD = 'PLI-PASS';
ERXCOM.COM_CLIENT_CODEPAGE = 'ECS0037';
ERXCOM.COM_CLIENT_TOKEN    = 'PLI-TOKEN';
ERXCOM.COM_SERVER_LIBARY   = 'MYLIB';
ERXCOM.COM_SERVER_WAIT     = '300S';
. . .
```

The client password can be given here if implicit broker logon is required in your environment. It is provided then through the interface object call, see also *Using Broker Logon and Logoff*.

Step 6: Issue the RPC Request

The procedure for issuing RPC requests varies, depending on whether you are using a call interface or an `EXEC CICS LINK` interface.

Using the Call Interface

This interface is used in the scenarios *Batch* and *CICS with Call Interfaces*.

```
RESULT = CALC(P_CALC.OPERATOR,
              P_CALC.OPERAND_1,
              P_CALC.OPERAND_2,
              ERXCOM);
```

The interface object `CALC` is called as PL/I function. See *Calling Servers as Procedures or Functions*.

Using the EXEC CICS LINK Interface

This interface is used in the scenario *CICS*.

```
/* move RPC Communication area to DFHCOMMAREA */
P_CALC.ERXCOM = ERXCOM;

/* call CICS program */
CICS_LEN   = STORAGE(P_CALC);
CICS_RESP1 = DFHRESP(NORMAL);
CICS_RESP2 = DFHRESP(NORMAL);
EXEC CICS LINK PROGRAM  ('CALC')
               RESP     (CICS_RESP1)
               RESP2    (CICS_RESP2)
```

```
               COMMAREA (P_CALC)
               LENGTH   (CICS_LEN);

/* move DFHCOMMAREA to RPC Communication area */
ERXCOM = P_CALC.ERXCOM;
```

Step 7: Examine the Error Code

When the RPC reply is returned, check that it was successful:

```
IF SUBSTR(ERXCOM.COM_ERROR,1,8) ^= ERX_S_SUCCESS then
DO;

/* error handling */
/* ... */

END;
```

The field COM_ERROR in the RPC communication area contains the error provided in a variable length char field. The 8-digit error number precedes the error text, and with the SUBSTR inbuilt function you can check the error number. In addition, you can use the COM_ERROR field simply in a PUT SKIP LIST statement for printouts.

For the error messages returned, see *Error Messages and Codes*.