Scenarios and Programmer Information

This chapter covers the following topics:

- COBOL Scenarios
- PL/I Scenarios
- C Scenarios
- Assembler Scenarios
- Aborting RPC Server Customer Code and Returning Error to RPC Client

---

COBOL Scenarios

Scenario I: Calling an Existing COBOL Server

To call an existing COBOL server

1. Use the *IDL Extractor for COBOL* to extract the Software AG IDL and, depending on the complexity, also a server mapping file. See *When is a Server Mapping File Required?* in the EntireX Workbench documentation.
2. Build an EntireX RPC client using any EntireX wrapper. See *EntireX Wrappers*. For a quick test you can:
   - use the IDL Tester; see *EntireX IDL Tester* in the EntireX Workbench documentation
   - generate an XML mapping file (XMM) and use the XML Tester for verification; see *EntireX XML Tester* in the XML/SOAP Wrapper documentation

See *Client and Server Examples for z/OS Batch* in the COBOL Wrapper documentation for COBOL RPC Server examples.

Scenario II: Writing a New COBOL Server

To write a new COBOL server

1. Use the *COBOL Wrapper* to generate a COBOL server skeleton and, depending on the complexity, also a server mapping file. See *When is a Server Mapping File Required?* in the EntireX Workbench documentation. Write your COBOL server and proceed as described under *Using the COBOL Wrapper for the Server Side*.
2. Build an EntireX RPC client using any EntireX wrapper. See *EntireX Wrappers*. For a quick test you can:
   - use the IDL Tester; see *EntireX IDL Tester* in the EntireX Workbench documentation

- generate an XML mapping file (XMM) and use the XML Tester for verification; see *EntireX XML Tester* in the XML/SOAP Wrapper documentation

See *Client and Server Examples for z/OS Batch* in the COBOL Wrapper documentation for COBOL RPC Server examples.

PL/I Scenarios

Scenario III: Calling an Existing PL/I Server

To call an existing PL/I server

1. Use the *IDL Extractor for PL/I* to extract the Software AG IDL.
2. Build an EntireX RPC client using any EntireX wrapper. See *EntireX Wrappers*. For a quick test you can:
   - use the IDL Tester; see *EntireX IDL Tester* in the EntireX Workbench documentation
   - generate an XML mapping file (XMM) and use the XML Tester for verification; see *EntireX XML Tester* in the XML/SOAP Wrapper documentation

See *Client and Server Examples for z/OS Batch* for PL/I RPC Server examples.

Scenario IV: Writing a New PL/I Server

To write a new PL/I server

1. Use the *PL/I Wrapper* to generate a PL/I server skeleton. Write your PL/I server and proceed as described under *Using the PL/I Wrapper for the Server Side*.
2. Build an EntireX RPC client using any EntireX wrapper. See *EntireX Wrappers*. For a quick test you can:
   - use the IDL Tester; see *EntireX IDL Tester* in the EntireX Workbench documentation
   - generate an XML mapping file (XMM) and use the XML Tester for verification; see *EntireX XML Tester* in the XML/SOAP Wrapper documentation

See *Client and Server Examples for z/OS Batch* in the PL/I Wrapper documentation for PL/I RPC Server examples.

C Scenarios

Scenario V: Writing a New C Server

To write a new C server

1. Use the *C Wrapper* to generate a C server skeleton and a C server interface object. Write your C server and proceed as described under *Using the C Wrapper for the Server Side (z/OS, UNIX, Windows, BS2000/OSD, IBM i)*.

2. Build an EntireX RPC client using any EntireX wrapper. See *EntireX Wrappers*. For a quick test you can:

   - use the IDL Tester; see *EntireX IDL Tester* in the EntireX Workbench documentation
   - generate an XML mapping file (XMM) and use the XML Tester for verification; see *EntireX XML Tester* in the XML/SOAP Wrapper documentation

Assembler Scenarios

Scenario VI: Writing a New Assembler Server

To write a new Assembler (IBM 370) server

1. Build an RPC server in Assembler. Here are some hints:

   - The RPC server is dynamically callable (no pre-initialization required).
   - The parameter interface is either compatible with the COBOL or PL/I calling convention (IDL level parameter will be passed in the address list). Configure the parameter `marshalling` accordingly for COBOL or PL/I.
   - The alignment of integer or float data types is considered. The HASM Assembler aligns integer or float data types to appropriate boundaries. For example:

     ```
     ...
           MyLabel   DSECT
           MyField1  DS   H                    I2
           MyField2  DS   F                    I4
           MyField3  DS   E                    F4
           MyField4  DS   L                    F8
     ```

   - The Batch RPC Server will not align these data types by default.
   - To force alignment by definition in your IDL file (see the aligned attribute within the `attribute-list`) before generating your RPC client. For information on whether your client supports the aligned attribute, see *Mapping the `aligned` Attribute* to C | COBOL | DCOM | .NET | Java | Natural | PL/I.

2. Build an EntireX RPC client using any EntireX wrapper. See *EntireX Wrappers*. For a quick test you can:

   - use the IDL Tester; see *EntireX IDL Tester* in the EntireX Workbench documentation
   - generate an XML mapping file (XMM) and use the XML Tester for verification; see *EntireX XML Tester* in the XML/SOAP Wrapper documentation

Aborting RPC Server Customer Code and Returning Error to RPC Client

Using RETURN-CODE Special Register (COBOL only)

The `RETURN-CODE` special register (an IBM extension to the COBOL programming language) is used by your RPC server to report an error.

Upon return, the value contained in the `RETURN-CODE` special register is detected by the Batch RPC Server and sent back to the RPC client instead of the application's data.

For IBM compilers the `RETURN-CODE` special register has the implicit definition:

```
RETURN-CODE GLOBAL PICTURE S9(4) USAGE BINARY VALUE ZERO
```

Special registers are reserved words that name storage areas generated by the compiler. Their primary use is to store information produced through specific COBOL features. Each such storage area has a fixed name, and must not be defined within the program. See your compiler documentation for more information.

The following rules apply to application error codes:

- The value range for application errors is 1-9999. No other values are allowed.
- On the RPC client side, the error is prefixed with the error class 1002 "Application User Error" and presented as error 1002*nnnn*.
- No application data is sent back to the RPC client in case of an error.
- It is not possible to return an error text to the RPC client.

  Example

```
. . .
        IF error occurred THEN
               MOVE <error-number> TO RETURN-CODE
               GO TO MAIN-EXIT
        END-IF.
        . . .

   MAIN-EXIT.
       EXIT PROGRAM.
END PROGRAM RETCODE.
```

Note:
To enable this feature, configure the Batch RPC Server with `return_code=yes`.