Using EntireX RPC for CL under IBM i

This chapter covers the following topics:

- Creating a Sample Server in CL
- Verifying the Server
- Software AG IDL to CL Mapping

See also *Administering the EntireX RPC Server*.

Creating a Sample Server in CL

This section describes how to build a server application using the IBM i ILE language CL. The sample server will be named SENDMESS. Using the IBM i command SNGPGMMSG (send program message), it sends a message to a given IBM i user and returns a confirmation to the RPC client.

This section tells you how to

- Create the Client/Server Interface
- Create the Server
- Compile and Link the Server

Create the Client/Server Interface

Using the EntireX Workbench on your PC, create a Software AG IDL file similar to the following:

```
Library 'EXAMPLE' Is
 Program 'SENDMESS' Is
  Define Data Parameter
   1 UserID           (A10)  In
   1 Message_Text     (A70)  In
   1 Confirmation     (A40)  Out
  End-Define
```

Section *Software AG IDL to CL Mapping* describes how IDL data types are mapped to CL data items.

Create the Server

The server is implemented as an ILE CL program of type *PGM.

For our IDL example SENDMESS, the implemented server looks similar to the example below:

```
       PGM        PARM(&USER &MESSTEXT &CONFIRM)
 /*-------------------------------------------------------------------*/
       DCL        VAR(&USER) TYPE(*CHAR) LEN(10)     /* the user ID   */
       DCL        VAR(&MESSTEXT) TYPE(*CHAR) LEN(70) /* the text      */
       DCL        VAR(&CONFIRM) TYPE(*CHAR) LEN(40)  /* returned text */
 /*-------------------------------------------------------------------*/
       CHGVAR     VAR(&CONFIRM) VALUE(' ')           /* clean it */
       SNDPGMMSG  MSG(&MESSTEXT) TOUSR(&USER) MSGTYPE(*COMP)
```

```
         MONMSG     MSGID(CPF0000) EXEC(GOTO CMDLBL(BAD))
         CHGVAR     VAR(&CONFIRM) +
            VALUE('Message sent to user' *BCAT &USER)
         GOTO       CMDLBL(DONE)                      /* sending was ok */
 /*-------------------------------------------------------------------*/
 BAD:    CHGVAR     VAR(&CONFIRM) +
            VALUE('Message sending failed')
 DONE:   ENDPGM
```

Because servers are running in a multithreaded environment, your application programs must be thread-safe. This implies that all commands and subprograms accessed in your servers must allow multithreads.

Compile and Link the Server

Compile the server source using the IBM i command CRTBNDCL (create bound CL program).

The following example procedure demonstrates how to compile and bind an ILE CL program:

```
          PGM        /* Compile and Bind a CL Server program         */
 /*-------------------------------------------------------------------*/
          DCL        VAR(&MODNAME) TYPE(*CHAR) LEN(10) VALUE(SENDMESS)
          DCL        VAR(&LIBL) TYPE(*CHAR) LEN(10) VALUE(EXAMPLE)
          DCL        VAR(&SRCF) TYPE(*CHAR) LEN(10) VALUE(QCLSRC)
          DCL        VAR(&OPTL) TYPE(*CHAR) LEN(10) VALUE(*NONE)
          DCL        VAR(&DBGV) TYPE(*CHAR) LEN(10) VALUE(*ALL)
 /*-------------------------------------------------------------------*/
          MONMSG     MSGID(CPF6801) EXEC(GOTO CMDLBL(DONE))
                     /* If PF12 is pressed */

          CRTBNDCL   ??PGM(&LIBL/&MODNAME) ??SRCFILE(&LIBL/&SRCF) +
                     ??SRCMBR(&MODNAME) DFTACTGRP(*NO) +
                       ACTGRP(*CALLER) OUTPUT(*PRINT) +
                       OPTIMIZE(&OPTL) DBGVIEW(&DBGV)

          MONMSG     MSGID(LNC9001) EXEC(GOTO CMDLBL(ERRXT))
          GOTO       CMDLBL(DONE)
 /*-------------------------------------------------------------------*/
  ERRXT:  SNDPGMMSG  MSG('MSG: Program Linkage Failed')
  DONE:   RETURN
          ENDPGM
```

Important:
When linking/binding servers, the binding parameter ACTGRP(*CALLER) must be specified. This guarantees that the server application runs in the same activation group as the calling RPC Server.

Name the resulting server program like the program name in the IDL file and put it in a library whose name corresponds to the library name in the IDL file.

Example:

If a client performs an RPC which is based on the IDL program SENDMESS in the IDL library EXAMPLE, the remote RPC server will dynamically try to execute the ILE server program SENDMESS in the IBM i library EXAMPLE. If no corresponding program can be found, the access will fail.

The principles of calling a server are described under *Administering the EntireX RPC Server*.

See *Step 3: Verify the RPC Server using COBOL* for how to start an RPC server that can execute the server program SENDMESS.

Verifying the Server

To verify the server program SENDMESS, you can use a client Java program from the *EntireX Workbench*.

To start the Java Client

1. Confirm that an EntireX Broker and an EntireX RPC server are available in your network.
2. Start/submit an RPC server on your IBM i machine as described under *Starting the RPC Server under IBM i*.
3. Open the EntireX Workbench and invoke the IDL file in which you have specified the IDL program SENDMESS during step *Create the Client/Server Interface*.
4. From the menu bar choose Java and Generate and run Test. After compilation, a pop-up menu will offer you the option of running the Java client program. Select the server program SENDMESS.
5. On the Java menu, specify the message text and the user ID to whom you want to send the message. The RPC request will be sent to the RPC Server on your IBM i machine, which will run the ILE server program SENDMESS in library EXAMPLE. The Java menu will then report the result of the IBM i command SNDPGMMSG. If the Broker connection fails, you will receive an appropriate error message.

Software AG IDL to CL Mapping

This section describes the specific mapping of Software AG IDL data types to the CL programming language. Please note also the remarks and hints on the Software AG IDL data types valid for all language bindings under *Software AG IDL File*.

The following topics are covered here:

- Mapping IDL Data Types to CL Data Types
- Mapping Program and Library Names
- Mapping Arrays, Groups and Structures
- Mapping the Direction Attributes In, Out, InOut

Mapping IDL Data Types to CL Data Types

In the table below, the following metasymbols and informal terms are used for the Software AG IDL.

- The metasymbols [and] enclose optional lexical entities.

- The informal term *n* is a sequence of numeric characters, for example 123.

Software AG IDL	Description	CL Data Type	See Notes
A*n*	Alphanumeric	TYPE(*CHAR) LEN(*n*)	
P*(n - p)*[.*p*]	Packed decimal	TYPE(*DEC) LEN(*n* [*p*])	1

See also the hints and restrictions valid for all language bindings under *IDL Data Types*.

Notes:

1. *n* must be less than or equal to 15. The maximum value for *p* is 9.
   For example, the IDL definition P10.2 corresponds to TYPE(*DEC) LEN(12 2)

Other IDL data types have no appropriate equivalent in the CL language.

Mapping Program and Library Names

Do not use the special characters ’#’, ’$’, ’&’, ’+’, ’-’, ’.’, ’/’ and ’@’ within names of programs and libraries in the IDL file. These characters are not allowed within names of server programs and libraries created on IBM i.

Mapping Arrays, Groups and Structures

Arrays, Groups and Structures are not supported for the CL language.

Mapping the Direction Attributes In, Out, InOut

The IDL syntax allows you to define parameters as IN parameters, OUT parameters, or IN OUT parameters (which is the default if nothing is specified). This direction specification is reflected in the stubless call of the RPC Server as follows:

- Direction attributes do not change the call interface because parameters are always treated as "called by reference".
- Usage of direction attributes may be useful to reduce data traffic between RPC client and RPC server.
- Parameters with the IN attribute are sent from the RPC client to the RPC server.
- Parameters with the OUT attribute are sent from the RPC server to the RPC client.
- Parameters with the IN and OUT attribute are sent from the RPC client to the RPC server and then back to the RPC client.

Note that only the direction information of the top-level fields (Level 1) is relevant. Group fields always inherit the specification from their parent. A different specification is ignored.

See the attribute-list under *Software AG IDL Grammar* for the syntax on how to describe attributes within the Software AG IDL file and refer to direction-attribute.