webMethods EntireX

EntireX COBOL Wrapper

Version 10.8

October 2022

WEBMETHODS

This document applies to webMethods EntireX Version 10.8 and all subsequent releases.

Specifications contained herein are subject to change and these changes will be reported in subsequent release notes or new editions.

Document ID: EXX-EEXXCOBWRAPPER-108-20220601

Table of Contents

EntireX COBOL Wrapper

EntireX COBOL Wrapper provides access to RPC-based components from COBOL applications. It enables you to develop both client and server applications.

Introduction	Introduction to the COBOL Wrapper.
Using	Step-by-step guide on how to generate interactively and build (write, compile and link) clients and server applications with the COBOL Wrapper. Programming models for Batch, CICS and IMS COBOL RPC applications are introduced. This section contains the following subsections: ■ ***Using the COBOL Wrapper for the Client Side*** ■ ***Using the COBOL Wrapper for the Server Side*** ■ ***Generating COBOL Source Files from Software AG IDL Files***
Command-line Mode	Using the COBOL Wrapper in command-line mode.
Mapping	Mapping Software AG IDL data types, groups, arrays and structures to the COBOL programming language.
Reliable RPC	Introduction to reliable RPC; writing a client and a server for Reliable RPC; Broker configuration.
RPC Communication Area	Provides reference material for the RPC Communication Area.
Delivered Modules	Describes the delivered COBOL Wrapper modules.

1 About this Documentation

Document Conventions

Convention	Description
Bold	Identifies elements on a screen.
`Monospace font`	Identifies service names and locations in the format *`folder.subfolder.service`*, APIs, Java classes, methods, properties.
`Italic`	Identifies: Variables for which you must supply values specific to your own situation or environment. New terms the first time they occur in the text. References to other documentation sources.
`Monospace font`	Identifies: Text you must type in. Messages displayed by the system. Program code.
{ }	Indicates a set of choices from which you must choose one. Type only the information inside the curly braces. Do not type the { } symbols.
|	Separates two mutually exclusive choices in a syntax line. Type one of these choices. Do not type the | symbol.
[ ]	Indicates one or more options. Type only the information inside the square brackets. Do not type the [ ] symbols.
...	Indicates that you can type multiple options of the same type. Type only the information. Do not type the ellipsis (...).

Online Information and Support

Product Documentation

You can find the product documentation on our documentation website at **https://documentation.softwareag.com**.

In addition, you can also access the cloud product documentation via **https://www.softwareag.cloud**. Navigate to the desired product and then, depending on your solution, go to “Developer Center”, “User Center” or “Documentation”.

Product Training

You can find helpful product training material on our Learning Portal at **https://knowledge.softwareag.com**.

Tech Community

You can collaborate with Software AG experts on our Tech Community website at **https://tech-community.softwareag.com**. From here you can, for example:

- Browse through our vast knowledge base.
- Ask questions and find answers in our discussion forums.
- Get the latest Software AG news and announcements.
- Explore our communities.
- Go to our public GitHub and Docker repositories at **https://github.com/softwareag** and **https://hub.docker.com/publishers/softwareag** and discover additional Software AG resources.

Product Support

Support for Software AG products is provided to licensed customers via our Empower Portal at **https://empower.softwareag.com**. Many services on this portal require that you have an account. If you do not yet have one, you can request it at **https://empower.softwareag.com/register**. Once you have an account, you can, for example:

- Download products, updates and fixes.
- Search the Knowledge Center for technical information and tips.
- Subscribe to early warnings and critical alerts.
- Open and update support incidents.
- Add product feature requests.

Data Protection

Software AG products provide functionality with respect to processing of personal data according to the EU General Data Protection Regulation (GDPR). Where applicable, appropriate steps are documented in the respective administration documentation.

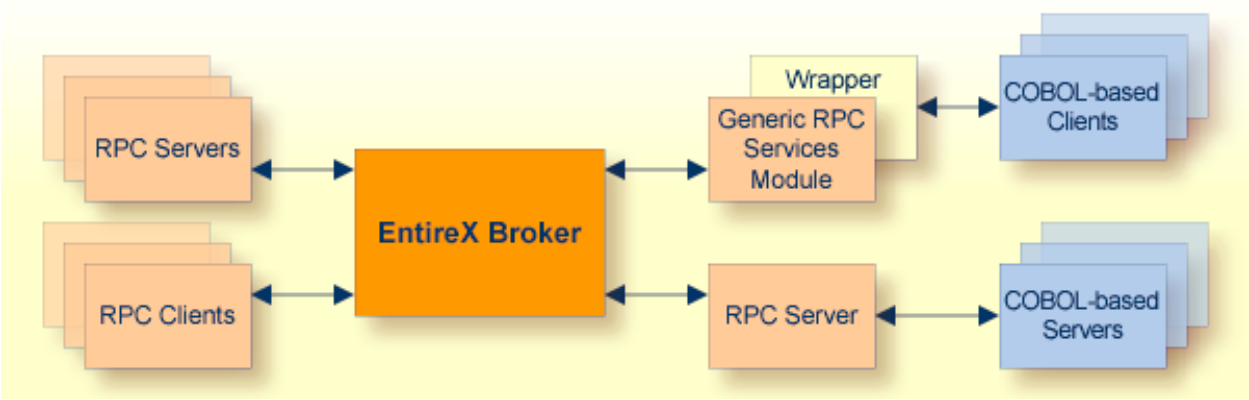
I Introduction to the COBOL Wrapper

2 Introduction to the COBOL Wrapper

EntireX COBOL Wrapper provides access to RPC-based components from COBOL applications. It enables you to develop both client and server applications.

Description

The COBOL Wrapper provides access to RPC servers for COBOL client applications and access to COBOL servers for any RPC client. The COBOL Wrapper generation tools of the Designer take as input a Software AG IDL file, which describes the interface of the RPC, and generate COBOL sources that implement the functions and data types of the interface.



The generated functions can be compiled with the COBOL compiler of your target platform.

The COBOL Wrapper works as follows:

- COBOL code is generated from the Software AG IDL file.
- Additionally for the client side, and depending on your target operating system and environment (e.g. Batch, CICS or IMS), a generic RPC services module is generated (see below).
- If required for the server side, a so-called server mapping file is created. A server mapping file is a Designer file with extension .cvm. See *Server Mapping Files for COBOL* in the Designer documentation.
- The Software AG IDL Compiler and an appropriate template are used for the COBOL code generation.

Generic RPC Services Module

In order to minimize the amount of code generated for a specific IDL file, all service-type functionality that is not specific to a given IDL file required by the client interface object is generated in a generic RPC services module.

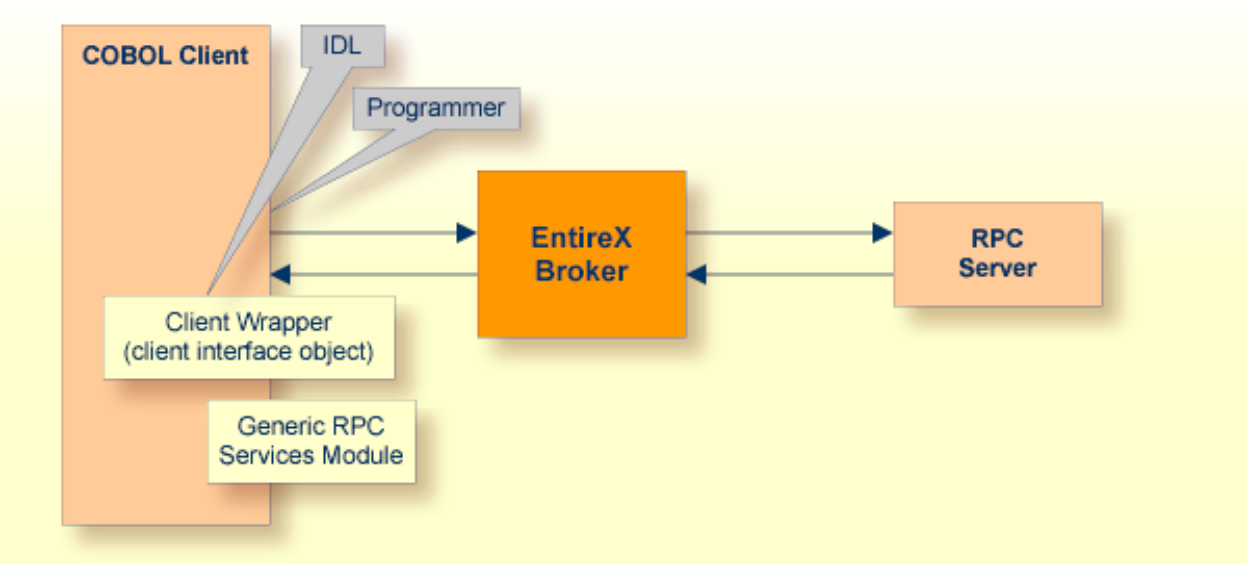
The generic RPC services module is used by RPC clients and contains the call to the broker stub, as well as other functions needed for RPC communication where an interface object is not needed, such as

- broker logon and logoff
- conversational support
- connecting RPC clients to RPC servers via the broker
- etc.

For more information, see ***Generation and Usage of Generic RPC Service Module COBSRVI***.

COBOL Client Applications

For a given IDL file, the Software AG IDL Compiler and a COBOL code generation template for clients are used to generate client interface objects and copybooks. See *Reslults for RPC Client* under ***Select an IDL File and Generate RPC Client or RPC Server***. The source code generated by the COBOL Wrapper can be compiled with your target COBOL compiler. Application developers use the generated generic RPC service module, the client interface object(s) and the copybooks to write COBOL applications that access RPC servers.

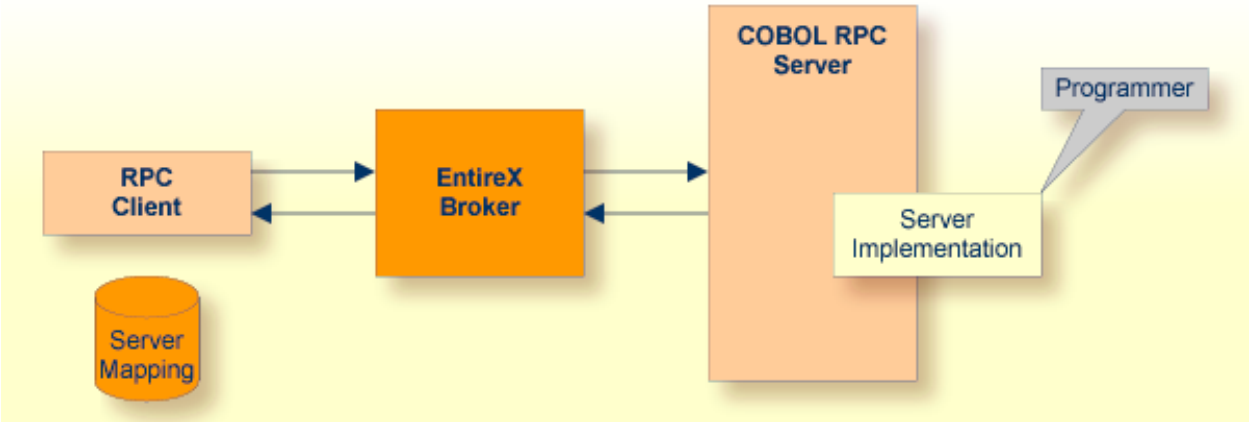


For more information, see *Using the COBOL Wrapper for the Client Side*.

COBOL Server Application

The Software AG IDL Compiler and a COBOL code generation template for servers are used to generate a server (skeleton) for a specific IDL. Additionally, depending on the IDL data types and whether IDL program names are customized, a so-called server mapping file is created (Designer file with extension .cvm).; See *When is a Server Mapping File Required?* If a server mapping file is created, you need to rebuild all RPC clients communicating with this RPC server program, see ***Using the COBOL Wrapper for the Server Side***.

Application developers use the generated server (skeleton) to write their own server code for each program in the IDL. The source code is compiled and linked with your target COBOL compiler.



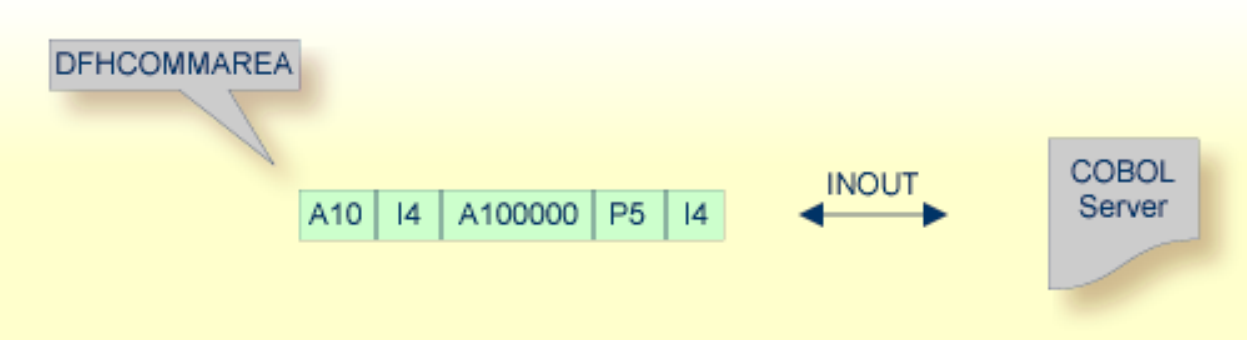
COBOL Server Interface Types

Depending on your requirements and generation settings, the COBOL Wrapper generates a server skeleton with one of the following interface types:

- CICS with DFHCOMMAREA Calling Convention
- CICS with Channel Container Calling Convention
- CICS with DFHCOMMAREA Large Buffer Interface
- Batch with Standard Linkage Calling Convention
- IMS BMP with Standard Linkage Calling Convention
- Compatibility between COBOL Interface Types and RPC Server
- Compatibility between COBOL Interface Types and EntireX Adapter Connection Types

CICS with DFHCOMMAREA Calling Convention

CICS programs using the standard DFHCOMMAREA for parameter passing.



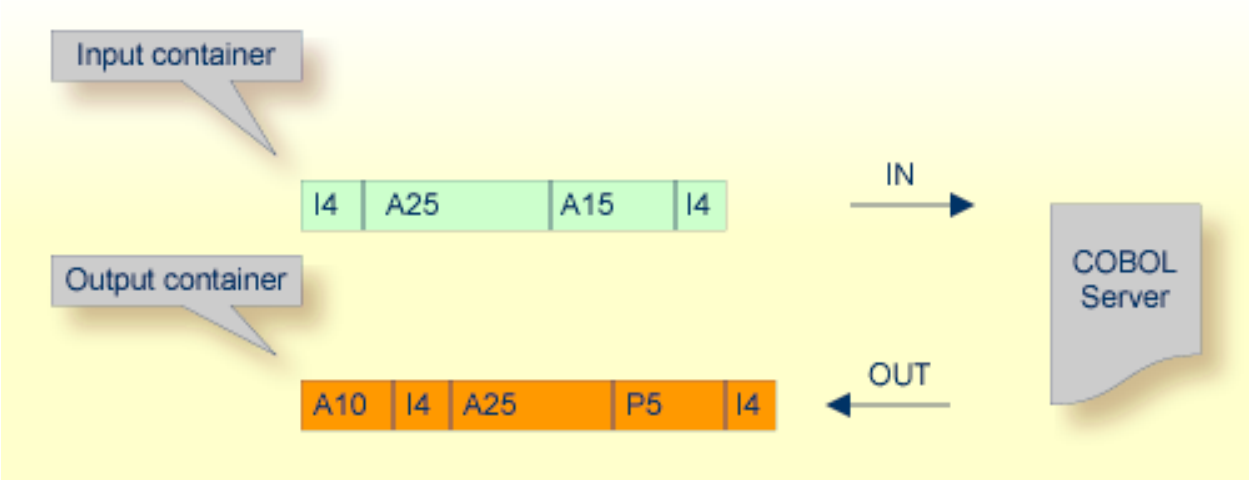
Technically, the generated COBOL server skeleton contains

- in the DFHCOMMAREA, the parameter structure

See *Server Interface Types* for more information on how to create COBOL servers with this interface type.

CICS with Channel Container Calling Convention

Channels and containers are IBM's approach to access more than 31 KB of data in CICS. There is no need for coding any channel container statements because all this is generated. Thus the programmer focus can be on the application logic.



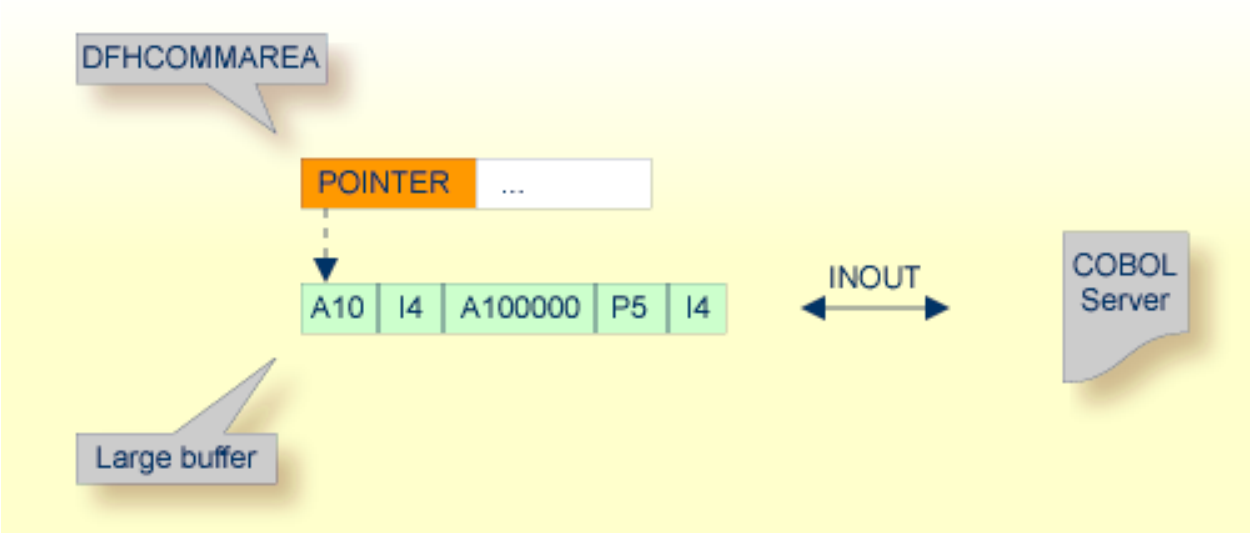
Technically, the generated COBOL server skeleton contains

- container layouts in the linkage section
- `EXEC CICS CONTAINER` statements for accessing the container on input and output

See *Server Interface Types* for more information on how to create COBOL servers with this interface type.

CICS with DFHCOMMAREA Large Buffer Interface

This type of program has a defined DFHCOMMAREA interface to access more than 31 KB of data in CICS. The interface is the same as the webMethods WMTLSRVR interface. This enables customers to use an easy and simple interface type to access more than 31 KB of data in CICS.



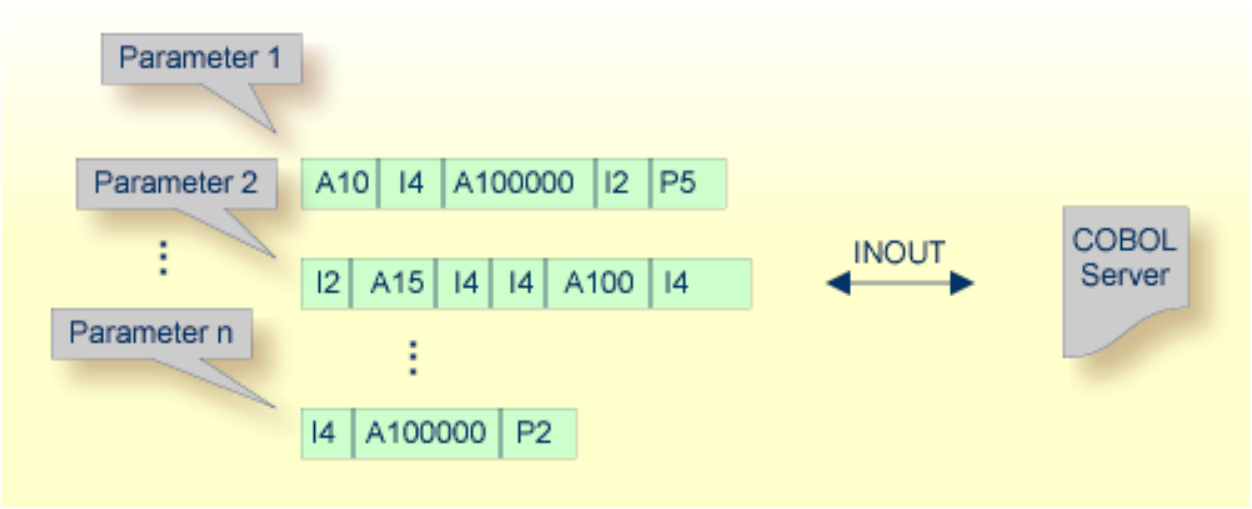
Technically,

- the generated server skeleton contains in the DFHCOMMAREA layout a *pointer* to a large buffer
- the parameter structure in the linkage section is accessed using the COBOL `SET ADDRESS` statement, using the large buffer pointer

See *Server Interface Types* for more information on how to create COBOL servers with this interface type.

Batch with Standard Linkage Calling Convention

Standard call interfaces with a given number of parameters are supported. Every parameter addresses a fixed COBOL structure.



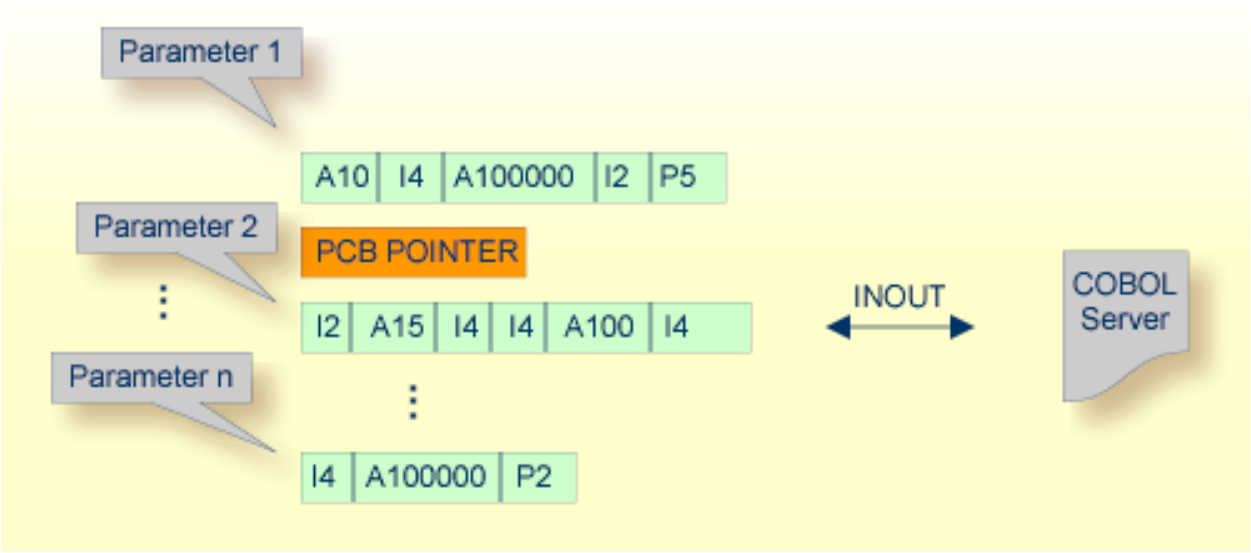
Technically, the generated COBOL server skeleton contains

- a parameter list `PROCEDURE DIVISION USING PARM1 PARM2 ... PARM`*n*
- the parameters in the linkage section as COBOL data items on level 1

See *Server Interface Types* for more information on how to create COBOL servers with this interface type.

IMS BMP with Standard Linkage Calling Convention

IMS batch message processing programs (BMP) with PCB parameters are directly supported.



Technically, the generated COBOL server skeleton contains

- IMS-specific PCB *pointers* within a parameter list.

See *Server Interface Types* for more information on how to create COBOL servers with this interface type.

Compatibility between COBOL Interface Types and RPC Server

To call your server program successfully, the target RPC runtime component used must support the interface type. The table below gives an overview of possible combinations of an interface type and RPC server.

	z/OS			UNIX/Windows			IBM i	BS2000
Interface Type of your Server Program	CICS	Batch	IMS	CICS ECI	CICS Socket Listener	IMS Connect	AS/400	Batch
CICS with DFHCOMMAREA Calling Convention	x			x	x			
CICS with DFHCOMMAREA Large Buffer Interface	x				x			
CICS with Channel Container Calling Convention	x				x			
Batch with Standard Linkage Calling Convention		x	x				x	x
IMS BMP with Standard Linkage Calling Convention			x					
IMS MPP Message Interface (IMS Connect)						x		
COBOL Converter								

Compatibility between COBOL Interface Types and EntireX Adapter Connection Types

The table below gives an overview of COBOL interface types and EntireX Adapter connection types.

Interface Type of your Server Program	EntireX Adapter Connection Type	Note
CICS with DFHCOMMAREA Calling Convention	CICS ECI Connection or CICS Socket Listener Connection	
CICS with DFHCOMMAREA Large Buffer Interface	CICS Socket Listener Connection	
CICS with Channel Container Calling Convention	CICS Socket Listener Connection	
Batch with Standard Linkage Calling Convention	AS/400 Connection	To call your server program on a platform other than IBM i, use an RPC Connection or Direct RPC Connection to an appropriate RPC Server for Batch (z/OS | BS2000).
IMS BMP with Standard Linkage Calling Convention	RPC Connection or Direct RPC Connection	Use the *RPC Server for IMS* as RPC server.

Interface Type of your Server Program	EntireX Adapter Connection Type	Note
IMS MPP Message Interface (IMS Connect)	IMS Connect Connection	
COBOL Converter	COBOL Converter Connection	

II Using the COBOL Wrapper

- *Using the COBOL Wrapper for the Client Side*
- *Using the COBOL Wrapper for the Server Side*
- *Generating COBOL Source Files from Software AG IDL Files*

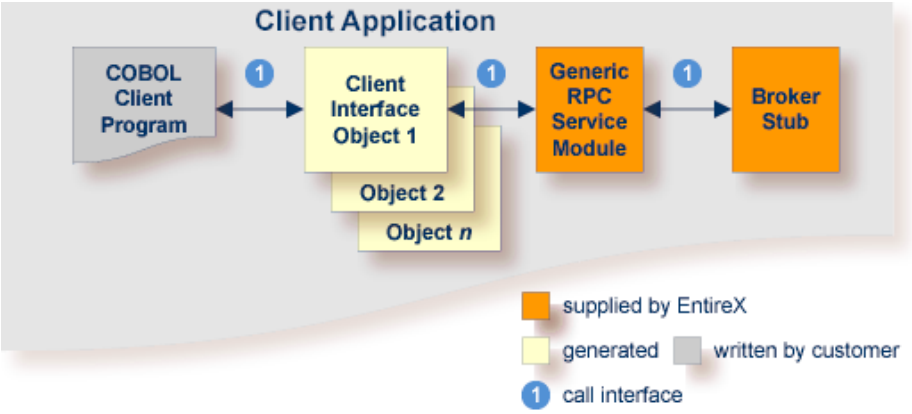
3 Using the COBOL Wrapper for the Client Side

The COBOL Wrapper provides access to RPC-based components from COBOL applications and enables you to develop both clients and servers. This chapter introduces the various possibilities for RPC-based client applications written in COBOL.

A step-by-step guide is provided in the section ***Writing Applications with the COBOL Wrapper***. Read this section first before writing your first RPC client program.

Using the COBOL Wrapper for CICS with Call Interfaces (z/OS and z/VSE)

This mode applies to z/OS and z/VSE.



The COBOL Wrapper can be used with a call interface, even in CICS. This means you can build a client application where every generated client interface object, the generic RPC services module and the broker stub are linked together or called dynamically by the COBOL client program, similar to the batch scenario. See *Using the COBOL Wrapper for Batch (z/OS, BS2000, z/VSE and IBM i)*.

Using a call interface within CICS may be useful if

- the maximum COMMAREA length for IDL data (about 31 KB) and other restrictions prevent you from using the *Using the COBOL Wrapper for CICS with `DFHCOMMAREA` Calling Convention (z/OS and z/VSE)* scenario
- you do *not* require a distributed program link (CICS DPL) to your client interface object(s)
- you prefer a call interface instead of `EXEC CICS LINK` to your client interface objects.

For platform z/OS this scenario supports the following:

- Long broker passwords. See *Using Broker Logon and Logoff*.
- Long RPC user IDs/passwords. See *Using RPC Authentication (Natural Security, Impersonation, Integration Server)*.
- IDL unbounded groups or arrays without maximum mapped to COBOL's `OCCURS 1 TO UNBOUNDED DEPENDING ON`.

To use the COBOL Wrapper with a call interface within CICS

1 Generate the client interface object(s) for the target operating system, for example "z/OS", and use the interface type "CICS with standard calling convention". See *Generating COBOL Source*

Files from Software AG IDL Files. Check the option **Generate the generic RPC service module COBSRVI**.

2 If necessary, use FTP to transfer the client interface object(s), and also the generic RPC service module COBSRVI, to the target platform where you write your client application.

3 Write your COBOL client program. If this is your first COBOL client program, refer to *Writing Standard Call Interface Clients*.

4 Using the CICS translator for COBOL provided with your CICS installation and a COBOL compiler supported by the COBOL Wrapper, translate and compile:

- the generated client interface object(s)
- if required, the generic RPC service module COBSRVI
- your COBOL client program

Take care the generated copybooks (see *Using the Generated Copybooks*) are accessed correctly by the compiler and not confused with the client interface objects, because the copybooks and client interface objects have identical file names. See your compiler documentation.

5 Using the standard linker (binder) of the target platform, link (bind) all translated and compiled modules, and, if required, the broker stub, together to the client application (that is, a CICS program), using the standard linker (binder) of the target platform.

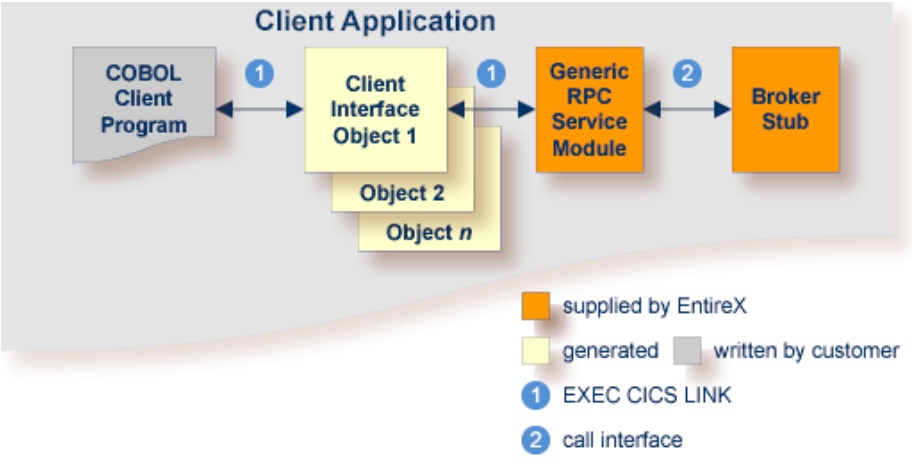
6 Install the client application within CICS.

7 Make sure the correct broker stub is used and can be called dynamically by the generic RPC service module COBSRVI.

- **z/OS**
  See the broker installation documentation and use a broker stub for CICS (for example CICSETB) from the common load library EXX108.LOAD. See also *Administering Broker Stubs*.
- **z/VSE**
  See the broker installation documentation and use a broker stub for CICS (for example BKIMC), see sublibrary EXX960.

Using the COBOL Wrapper for CICS with DFHCOMMAREA Calling Convention (z/OS and z/VSE)

This mode applies to z/OS and z/VSE.



In this scenario, the generic RPC services module is installed only once within CICS as a CICS program and shared by all COBOL RPC client programs. Also, the COBOL client program and every generated client interface object are installed each as separate individual CICS programs.

Use the COBOL Wrapper for CICS with DFHCOMMAREA calling convention in the following situations:

- You want to have an `EXEC CICS LINK` DFHCOMMAREA interface to your client interface object(s).
- You wish to separate the generic RPC service module and the broker stub from the client interface object(s).
- You require a program link to the client interface object(s).
- You can accept the following restrictions:
  - The maximum COMMAREA length suits your purposes. Because the RPC communication area is also transferred in the COMMAREA, the effective length that can be used for IDL data is shorter than the CICS COMMAREA length. Nearly 31 KB can be used for IDL data.
  - No support for long broker passwords and long RPC user IDs/passwords.
  - No support for IDL unbounded arrays without maximum. See *Mapping Fixed and Unbounded Arrays*.

    Check if *Using the COBOL Wrapper for CICS with Call Interfaces (z/OS and z/VSE)* is an alternative for you.

To use the COBOL Wrapper for CICS with DFHCOMMAREA calling convention

1 Generate the client interface object for the target operating system, for example "z/OS", and use interface type "CICS with DFHCOMMAREA calling convention". See *Generating COBOL Source Files from Software AG IDL Files*. Check the option **Generate the generic RPC service module COBSRVI**.

2 If necessary, use FTP to transfer the client interface object(s), and also the generic RPC service module COBSRVI, to the target platform where you write your client application.

3 Write your COBOL client program. If this is your first COBOL client program, refer to *Writing `EXEC CICS LINK` Clients*.

4 Using the CICS translator for COBOL provided with your CICS installation and a COBOL compiler supported by the COBOL Wrapper, translate and compile:

- the generated client interface object(s)
- if required, the generic RPC service module COBSRVI
- your COBOL client program.

Take care the generated copybooks (see *Using the Generated Copybooks*) are accessed correctly by the compiler and not confused with the client interface objects, because the copybooks and client interface objects have identical file names. See your compiler documentation.

5 Using the standard linker (binder) of the target platform, link (bind) the following programs to separate CICS programs:

- every generated client interface object
- if required, the generic RPC service module COBSRVI together with a broker stub
- your COBOL client program.

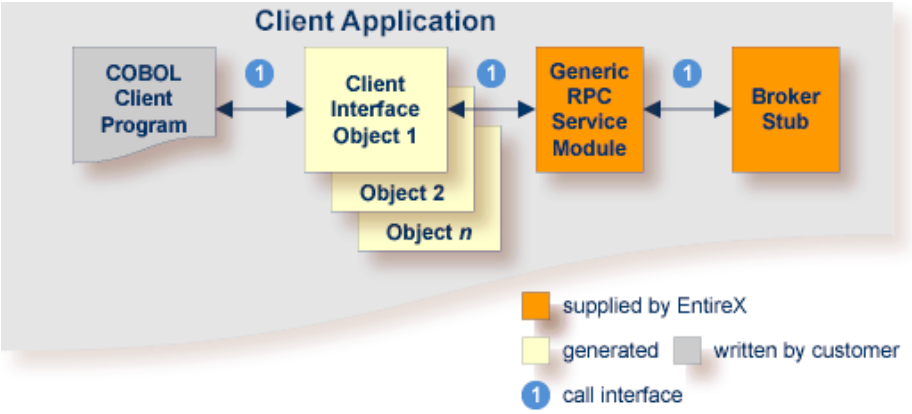
6 Install every client interface object, if required the CICS RPC service module COBSRVI and your COBOL client program as separate CICS programs.

7 Make sure the correct broker stub is used and can be called dynamically by the CICS generic RPC service module COBSRVI.

- **z/OS**
  See the broker installation documentation and use a broker stub for CICS (for example CICSETB) from the common load library EXX108.LOAD. See also *Administering Broker Stubs*.
- **z/VSE**
  See the broker installation documentation and use a broker stub for CICS (for example BKIMC), see sublibrary EXX960.

Using the COBOL Wrapper for Batch (z/OS, BS2000, z/VSE and IBM i)

This mode applies to z/OS, BS2000, z/VSE and IBM i.



In this scenario, every generated client interface object, the generic RPC services module and the broker stub are linked together or called dynamically by the COBOL client program.

For platform z/OS this scenario supports the following:

- Long broker passwords. See *Using Broker Logon and Logoff*.
- Long RPC user IDs/passwords. See *Using RPC Authentication (Natural Security, Impersonation, Integration Server)*.
- IDL unbounded groups or arrays without maximum mapped to COBOL's `OCCURS 1 TO UNBOUNDED DEPENDING ON`.

To use the COBOL Wrapper for batch

1 Generate the client interface object(s) for the target operating system, for example "z/OS", and use interface type "Batch with standard linkage calling convention". See *Generating COBOL Source Files from Software AG IDL Files*. Check the option **Generate the generic RPC service module COBSRVI**.

2 If necessary, use FTP to transfer the client interface object(s), and also the generic RPC service module COBSRVI, to the target platform where you write your client application.

3 Write your COBOL client program. If this is your first COBOL client program, refer to *Writing Standard Call Interface Clients*.

4 Using a COBOL compiler supported by COBOL Wrapper, compile:

- the generated client interface object(s)
- if required, the generic RPC service module COBSRVI

- your COBOL client program

Take care the generated copybooks (see ***Using the Generated Copybooks***) are accessed correctly by the compiler and not confused with the client interface objects, because the copybooks and client interface objects have identical file names. See your compiler documentation.

- **BS2000**
  The IDL types U or UV require a compiler that supports COBOL data type NATIONAL. See *BS2000 Prerequisites* for more information on supported compilers.
- **IBM i**
  Use the command CRTCBLMOD (create COBOL module) and compile all modules above to ILE modules using the following options:

```
CRTCBLMOD MODULE(COBSRVI) OPTION(*NOMONOPRC) LINKLIT(*PRC)
CRTCBLMOD MODULE(<client-interface-object-1>) LINKLIT(*PRC)
CRTCBLMOD MODULE(<client-interface-object-2>) LINKLIT(*PRC)
CRTCBLMOD MODULE(<client-interface-object..>) LINKLIT(*PRC)
CRTCBLMOD MODULE(<COBOL-client-program>) LINKLIT(*PRC)
```

- **Other Platforms**
  Use the standard COBOL compiler of the target platform.

5 Using the standard linker (binder) of the target platform, link (bind) the following programs:

- the generated client interface object(s)
- if required, the generic RPC service module COBSRVI
- if required, the broker stub
- your COBOL client program

Depending on the platform:

- **IBM i**
  Use the IBM i command `CRTPGM` to bind all compiled modules to an executable ILE program of type *PGM.

```
CRTPGM PGM(<COBOL-client-program>)
 MODULE(<COBOL-client-program> <client-interface-object-1> ↩
<client-interface-object-2> <client-interface-object...>)
 BNDSRVPGM(EXX/EXA)
```

  where `EXX` is the EntireX product library and `EXA` the broker stub.

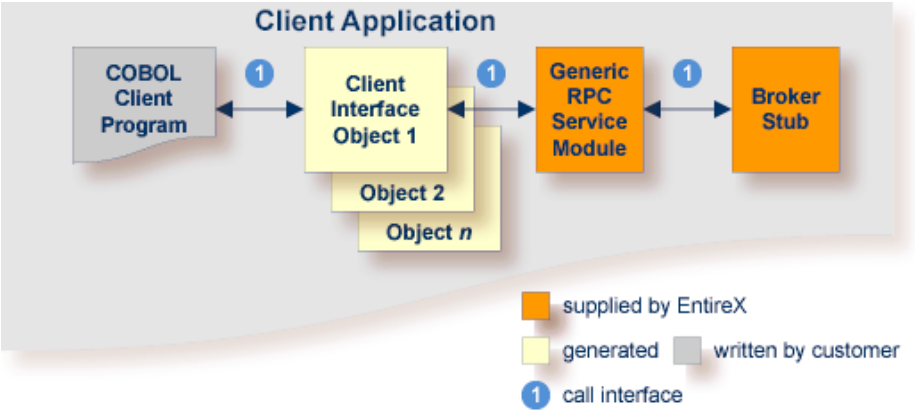
- **Other Platforms**
  Refer to your standard linker (binder) documentation.

6 Make sure that the correct broker stub module is used and, if linked (bound) dynamically, that it can be called dynamically.

- **z/OS**
  See the broker installation documentation and use a broker stub for batch (for example BROKER) from the common load library EXX108.LOAD. See also *Administering Broker Stubs*.
- **z/VSE**
  See the broker installation documentation and use a broker stub for batch (for example BKIMB), see sublibrary EXX960.
- **BS2000**
  The broker stub module BROKER is located in the broker LMS load library.
- **IBM i**
  The broker stub EXA is located by default in the EntireX product library EXX. Make sure this library is added to your library list.

Using the COBOL Wrapper for IMS (z/OS)

This mode applies to z/OS IMS modes BMP and MPP.



In this scenario, every generated client interface object, the generic RPC services module and the broker stub are linked together or called dynamically by the COBOL client program.

For platform z/OS this scenario supports the following:

- Long broker passwords. See *Using Broker Logon and Logoff*.
- Long RPC user IDs/passwords. See *Using RPC Authentication (Natural Security, Impersonation, Integration Server)*.
- IDL unbounded groups or arrays without maximum mapped to COBOL's `OCCURS 1 TO UNBOUNDED DEPENDING ON`.

To use the COBOL Wrapper for IMS

1. Generate the client interface object(s) for the target operating system "z/OS" and use the interface type "IMS BMP with standard linkage calling convention" or "IMS MMP with standard linkage calling convention". See *Generating COBOL Source Files from Software AG IDL Files*. Check the option **Generate the generic RPC service module COBSRVI**. .
2. If necessary, use FTP to transfer the client interface object(s), and also the generic RPC service module COBSRVI, to the target platform where you write your client application.
3. Write your COBOL client program. If this is your first COBOL client program, refer to *Writing Standard Call Interface Clients*.
4. Using a COBOL compiler supported by the COBOL Wrapper, compile:
   - the generated client interface object(s)
   - if required, the generic RPC service module COBSRVI

- your COBOL client program.

Take care the generated copybooks (see ***Using the Generated Copybooks***) are accessed correctly by the compiler and not confused with the client interface objects, because the copybooks and client interface objects have identical file names. Do not assign the data set with the client interface objects prior in sequence to the copybooks to SYSLIB. See your compiler documentation.

5 Link (bind) all compiled modules and, if required, the broker stub, together to an executable program, using the standard linker (binder) of the target platform.

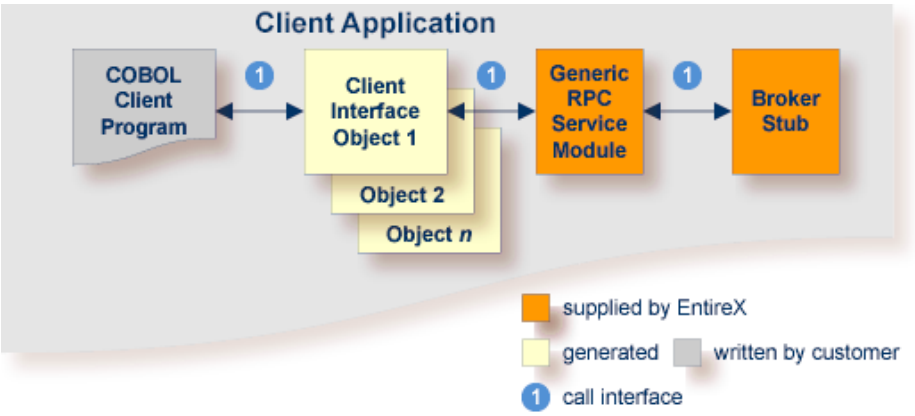
6 Make sure the correct broker stub is used and can be called dynamically. In the common load library EXX108.LOAD you can find broker stubs that can be used for

- IMS BMP (for example BROKER)
- IMS MPP (for example MPPETB)

See *Administering Broker Stubs*.

Using the COBOL Wrapper for IDMS/DC with Call Interfaces (z/OS)

This mode applies to z/OS.



The COBOL Wrapper can be used with a call interface in IDMS/DC. This means you can build an application where the COBOL client program, every generated client interface object, the generic RPC services module and the broker stub are linked together, similar to the batch scenario. See *Using the COBOL Wrapper for Batch (z/OS, BS2000, z/VSE and IBM i)*.

To use the COBOL Wrapper with a call interface within IDMS/DC

1. Generate the client interface object(s) for the target operating system "z/OS", and use the interface type "IDMS/DC with standard calling convention". See *Generating COBOL Source Files from Software AG IDL Files*. Check the option **Generate the generic RPC service module COBSRVI**.
2. If necessary, use FTP to transfer the client interface object(s), and also the generic RPC service module COBSRVI, to the target platform where you write your client application.
3. Write your COBOL client program. See *Writing Applications with the COBOL Wrapper*, in particular the section *The RPC Communication Area (Reference)*, and take into consideration the information given in *Software AG IDL to COBOL Mapping*.
4. Write your COBOL client program. If this is your first COBOL client program, refer to *Writing Standard Call Interface Clients*.
5. Using the standard linker (binder) of the target platform, link (bind) all translated and compiled modules, and, if required, the broker stub, together to an IDMS/DC program, using the standard linker (binder) of the target platform.
6. Install the IDMS/DC program within IDMS/DC.
7. Make sure the correct broker stub is used and can be called dynamically by the generic RPC service module COBSRVI.

See the broker installation documentation and use a broker stub for IDMS/DC (for example IDMSETB) from the common load library EXX108.LOAD. See also *Administering Broker Stubs*.

4 Using the COBOL Wrapper for the Server Side

The COBOL Wrapper provides access to RPC-based components from COBOL applications and enables you to develop both clients and servers. This chapter introduces the various possibilities for RPC-based server applications written in COBOL.

Using the COBOL Wrapper for CICS with DFHCOMMAREA Calling Convention (z/OS and z/VSE)

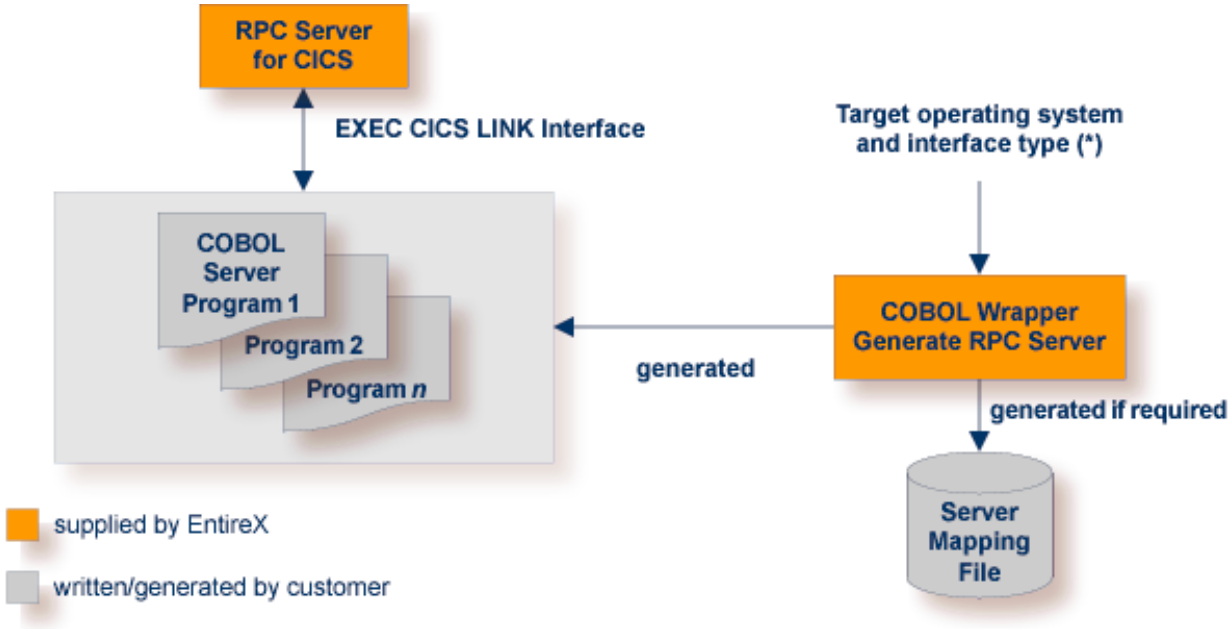
This section covers the following topics:

- Introduction
- Steps

See also *COBOL Scenarios* in the RPC Server for CICS documentation.

Introduction

This mode applies to z/OS and z/VSE.



(*) See *Target Operating System* and *Server Interface Types* under *Generating COBOL Source Files from Software AG IDL Files*.

In CICS, the RPC server sets up all of your server's parameters dynamically in the format required. Your server is called using `EXEC CICS LINK`.

Use the COBOL Wrapper for CICS with DFHCOMMAREA calling convention if

- you want to have a standard `EXEC CICS LINK` DFHCOMMAREA interface to your server
- you require a distributed program link (CICS DPL) to your server

- the DFHCOMMAREA length restriction (31 KB) suits your needs, otherwise consider the following interface types:
  - *Using the COBOL Wrapper for CICS with Channel Container Calling Convention (z/OS)*
  - *Using the COBOL Wrapper for CICS with `DFHCOMMAREA` Large Buffer Interface (z/OS and z/VSE)*

Steps

To use the COBOL Wrapper for CICS with DFHCOMMAREA calling convention

1 Generate the server (skeleton) for the target operating system, for example "z/OS", and use interface type "CICS with DFHCOMMAREA calling convention". See *Generating COBOL Source Files from Software AG IDL Files*.

2 If a server mapping file is required, it has to be provided. A server mapping file is a Designer file with extension .cvm. See *Server Mapping Files for COBOL* in the Designer documentation. The server mapping files are wrapped into RPC clients and provided with the RPC request. You need to rebuild all RPC clients communicating with this RPC server program. Select the appropriate wrapper (see *EntireX Wrappers* in the Designer documentation) and re-generate the client interface objects. For the *EntireX Adapter* you need to update your generated IS adapter as described under *To update an existing connection* in *Step 3: Create or Update an Adapter Connection* in the Integration Server Wrapper documentation.

3 If necessary, use FTP to transfer the server (skeleton(s)) to the target platform where you write your server.

4 Use the generated server (skeleton(s)) and complete it by applying your application logic. Note the information given in *Software AG IDL to COBOL Mapping* and *Returning Application Errors*.

5 Using the CICS translator for COBOL provided with your CICS installation and a COBOL compiler supported by the COBOL Wrapper, translate and compile your server.

6 Link (bind) the server to an executable program, using the standard linker (binder) of the target platform. Give your server a CICS program name that is the same as the `program-name` in the IDL file. See `program-definition` under *Software AG IDL Grammar* in the IDL Editor documentation.

7 Provide your server(s) to the RPC Server for CICS, EntireX Adapter, or RPC Server for CICS ECI:

   - Install your server(s) as separate CICS program(s).
   - If you are using the RPC Server for CICS, before using your server(s), check if you need to alter
     - CICS settings - for example `TWASIZE`; see *CICS Settings*

- for z/OS additionally *IBM LE Runtime Options* - for example AMODE24, how to trap ABENDS etc.

8 Test your server.

- If you are using a server mapping file (.cvm), the server mapping is taken from the RPC request and the program with the COBOL name (see Note) defined in the server mapping is executed. See ***Customize Automatically Generated Server Names***. If no corresponding program can be found, the access will fail.
- If no server mapping file is required (see *When is a Server Mapping File Required?* in the Designer documentation) or the server is generated with a previous version of EntireX without support for server mapping - the library name (see `library-definition` under *Software AG IDL Grammar* in the IDL Editor documentation) given in the IDL is ignored (see Note).

  Example: If a client performs an RPC request that is based on the IDL program name CALC, the RPC server will dynamically try to execute a program CALC. If no corresponding program can be found, the access will fail.

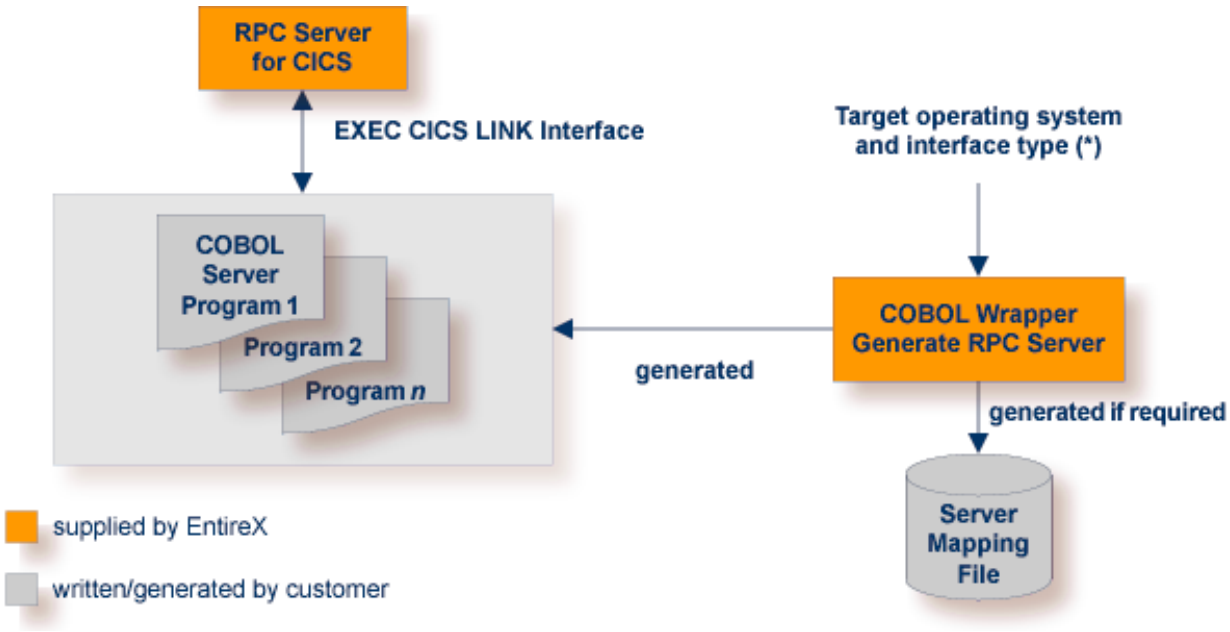
Using the COBOL Wrapper for CICS with Channel Container Calling Convention (z/OS)

This section covers the following topics:

See also *COBOL Scenarios* in the RPC Server for CICS documentation.

Introduction

This mode applies to z/OS.



(*) See *Target Operating System* and *Server Interface Types* under *Generating COBOL Source Files from Software AG IDL Files*.

In CICS, the RPC server sets up all of your server's parameters dynamically in the format required. Your server is called using `EXEC CICS LINK` passing the container(s) in the defined channel to your server. See ***Channel Name***.

Use the COBOL Wrapper for CICS with channel container calling convention if

- you require more than 31 KB of data to transfer to your server
- your IDL complies with CICS channel container IDL rules (see below). If your IDL does not match these rules, consider the interface type ***Using the COBOL Wrapper for CICS with `DFHCOMMAREA` Large Buffer Interface (z/OS and z/VSE)*** to implement your server.
- you want to have a standard CICS channel container interface to your server
- you require a distributed program link (CICS DPL) to your server.

CICS Channel Container IDL Rules

The following rules apply to CICS channel container IDL:

- A container is described with an IDL structure. See `structure-definition` under *Software AG IDL Grammar* in the IDL Editor documentation.
- The container name is the name of the IDL structure. A maximum of 16 characters are allowed by CICS for container names.
- IDL programs reference IDL structures only. No other parameters may be referenced.
- Multiple containers can be defined, see ***Example 3: Multiple Containers***.
- A variable number of containers can be defined using one-dimensional IDL unbounded arrays with maximum (see `array-definition` under *Software AG IDL Grammar* in the IDL Editor documentation). See also ***Example 4: Variable Number of Containers (Direction Out Only)***.

Restrictions

- IDL unbounded arrays (i.e. variable containers) for direction In and `INOUT` are not supported.
- Two and three-dimensional IDL unbounded arrays are not supported.

Example 1: Same Container for Direction In and Out

This example uses the same container for input and output. The container name is "CALC".

```
Library 'EXAMPLE' Is
  Program 'CONCALC' Is
   Define Data Parameter
    1 Container       ('CALC')        InOut
   End-Define

  Struct 'CALC' Is
   Define Data Parameter
    1 Operation       (A1)
    1 Operand_1       (I4)
    1 Operand_2       (I4)
    1 Function_Result (I4)
   End-Define
```

Example 2: Different Container for Direction In and Out

This example uses separate containers for input and output.

```
Library 'DFHCON' Is
 Program 'TWOC' Is /* Two Container - Separate for Input and Output
   Define Data Parameter
    1 ContainerIn   ('CONTAINER1')  In
    1 ContainerOut  ('CONTAINER2')  Out
   End-Define
Struct 'CONTAINER1' Is
  Define Data Parameter
   1 Just-Occupied-Space (A39000) /* 39K
   1 Request             (A1000/5) /* 5K
  End-Define
Struct 'CONTAINER2' Is
  Define Data Parameter
    1 Just-Occupied-Space (A49000) /* 49K
    1 Reply               (A250)
  End-Define
```

See IDL program `TWOC` under *Advanced CICS Channel Container RPC Server Example*.

Example 3: Multiple Containers

This example shows how more than one container is used per direction. Each container has its own structure layout.

```
Library 'DFHCON' Is
  Program 'MULTIC' Is
   Define Data Parameter
    1 InContainer1     ('INCONTAINER1')  In
    1 InContainer2     ('INCONTAINER2')  In
    1 InContainer3     ('INCONTAINER3')  In
    ...

    1 OutContainer1    ('OUTCONTAINER1') Out
    1 OutContainer2    ('OUTCONTAINER2') Out
    1 OutContainer3    ('OUTCONTAINER3') Out
    ...

   End-Define

  Struct 'INCONTAINER1' Is ...
  Struct 'INCONTAINER2' Is ...
  Struct 'INCONTAINER3' Is ...
  ...

  Struct 'OUTCONTAINER1' Is ...
  Struct 'OUTCONTAINER1' Is ...
  Struct 'OUTCONTAINER1' Is ...
  ...
```

Example 4: Variable Number of Containers (Direction Out Only)

This example shows how to specify a range of containers. At runtime, the called RPC server creates a variable number of containers from this range. Each container created has the same structure layout and a container name that is formed from the structure name as prefix and the structure index as suffix. In this example:

- `MULTIPLE` container names are `MULTIPLE0001` thru `MULTIPLE9999`.
- `OPTIONAL` container name is `OPTIONAL1`.

Note: Make sure IDL observes the 16-character length restriction for container names given by CICS.

```
Library 'DFHCON' Is
  Program 'VARC' Is
   Define Data Parameter
    1 Input            ('INPUT')          In
    1 Multiple         ('MULTIPLE'/V9999) Out /* 0 thru 9999 times
    1 Optional         ('OPTIONAL'/V1)    Out /* 0 or 1 times
   End-Define

  Struct 'INPUT' Is ...
  Struct 'MULTIPLE' Is ...
  Struct 'OPTIONAL' Is ...
```

Steps

To use the COBOL Wrapper for CICS with channel container calling convention

1 Generate the server (skeleton(s)) for the target operating system, for example "z/OS", and use interface type "CICS with channel container calling convention". See ***Generating COBOL Source Files from Software AG IDL Files***.

2 The generated server mapping file has to be provided. A server mapping file is a Designer file with extension .cvm. See *Server Mapping Files for COBOL* in the Designer documentation. The server mapping files are wrapped into RPC clients and provided with the RPC request. You need to rebuild all RPC clients communicating with this RPC server program. Select the appropriate wrapper (see *EntireX Wrappers* in the Designer documentation) and re-generate the client interface objects. For the *EntireX Adapter* you need to update your generated IS adapter as described under *To update an existing connection* in *Step 3: Create or Update an Adapter Connection* in the Integration Server Wrapper documentation.

3 If necessary, use FTP to transfer the server (skeleton(s)) to the target platform where you write your server.

4 Use the generated server (skeleton(s)) and complete it by applying your application logic. Note the information given in ***Software AG IDL to COBOL Mapping*** and *Returning Application Errors* under *Scenarios and Programmer Information* in the RPC Server for CICS documentation.

5 Using the CICS translator for COBOL provided with your CICS installation and a COBOL compiler supported by the COBOL Wrapper, translate and compile your server.

6 Link (bind) the server to an executable program, using the standard linker (binder) of the target platform. Give your server a CICS program name that is the same as the `program-name` in the IDL file (see `program-definition` under *Software AG IDL Grammar* in the IDL Editor documentation).

7 Provide your server(s) to the RPC Server for CICS.

- Install your server(s) as separate CICS program(s).
- If you are using the RPC Server for CICS, before using your server(s), check if you need to alter
  - CICS settings - for example `TWASIZE`; see *CICS Settings*
  - for z/OS additionally *IBM LE Runtime Options* - for example AMODE24, how to trap ABENDS etc.

8 Test your server.

- The generated server mapping is sent with the RPC request and the program with the COBOL name defined in the server mapping, is executed. See ***Customize Automatically Generated Server Names***. If no corresponding program can be found, the access will fail.

Using the COBOL Wrapper for CICS with DFHCOMMAREA Large Buffer Interface (z/OS and z/VSE)

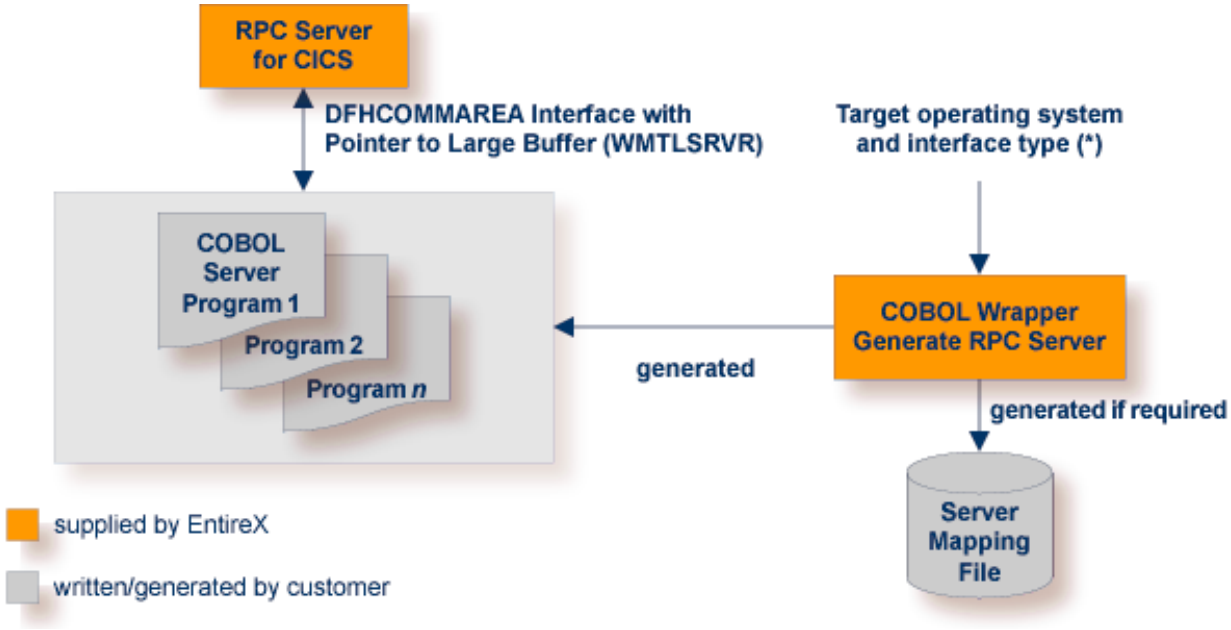
This section covers the following topics:

- Introduction
- Steps

See also *COBOL Scenarios* in the RPC Server for CICS documentation.

Introduction

This mode applies to z/OS and z/VSE.



(*) See *Target Operating System* and *Server Interface Types* under *Generating COBOL Source Files from Software AG IDL Files*.

In CICS, the RPC server sets up all your server's parameters dynamically in the format required. Your server is called by `EXEC CICS LINK`. Within the DFHCOMMAREA, pointers are passed to a large input/output buffer.

Use the COBOL Wrapper for CICS with DFHCOMMAREA large buffer interface in the following situations:

- You need to migrate COBOL programs implemented with webMethods WMTLSRVR interface to the RPC Server for CICS.
- You require more than 31 KB of data to transfer to your server.
- You cannot use the channel container calling convention because your IDL does not match the applicable rules; see ***CICS Channel Container IDL Rules*** under Using the COBOL Wrapper for CICS with Channel Container Calling Convention (z/OS). There are no IDL restrictions for this interface type - every IDL can be used.
- You prefer this interface type rather than the channel container interface type.
- You do *not* require a distributed program link (CICS DPL) to your server.

Steps

To use the COBOL Wrapper for CICS with large buffer interface

1 Generate the server (skeleton(s)) for the target operating system, for example "z/OS", and use interface type "CICS with DFHCOMMAREA large buffer interface". See ***Generating COBOL Source Files from Software AG IDL Files***.

2 The generated server mapping file has to be provided. A server mapping file is a Designer file with extension .cvm. See *Server Mapping Files for COBOL* in the Designer documentation. The server mapping files are wrapped into RPC clients and provided with the RPC request. You need to rebuild all RPC clients communicating with this RPC server program. Select the appropriate wrapper (see *EntireX Wrappers* in the Designer documentation) and re-generate the client interface objects. For the *EntireX Adapter* you need to update your generated IS adapter as described under *To update an existing connection* in *Step 3: Create or Update an Adapter Connection* in the Integration Server Wrapper documentation.

3 If necessary, use FTP to transfer the server (skeleton(s)) to the target platform where you write your server.

4 Use the generated server (skeleton(s)) and complete it by applying your application logic. Note the information given in ***Software AG IDL to COBOL Mapping*** and *Returning Application Errors* under *Scenarios and Programmer Information* in the RPC Server for CICS documentation.

5 Using the CICS translator for COBOL provided with your CICS installation and a COBOL compiler supported by the COBOL Wrapper, translate and compile your server.

6 Link (bind) the server to an executable program, using the standard linker (binder) of the target platform. Give your server a CICS program name that is the same as the `program-name` in the IDL file (see `program-definition` under *Software AG IDL Grammar* in the IDL Editor documentation).

7 Provide your server(s) to the RPC Server for CICS.

- Install your server(s) as separate CICS program(s).

- If you are using the RPC Server for CICS, before using your server(s), check if you need to alter
  - CICS settings - for example `TWASIZE`; see *CICS Settings*
  - for z/OS additionally *IBM LE Runtime Options* - for example AMODE24, how to trap ABENDS etc.

8 Test your server.

- The generated server mapping is sent with the RPC request and the program with the COBOL name defined in the server mapping, is executed. See ***Customize Automatically Generated Server Names***. If no corresponding program can be found, the access will fail.

Using the COBOL Wrapper for Batch (z/OS, BS2000 and IBM i)

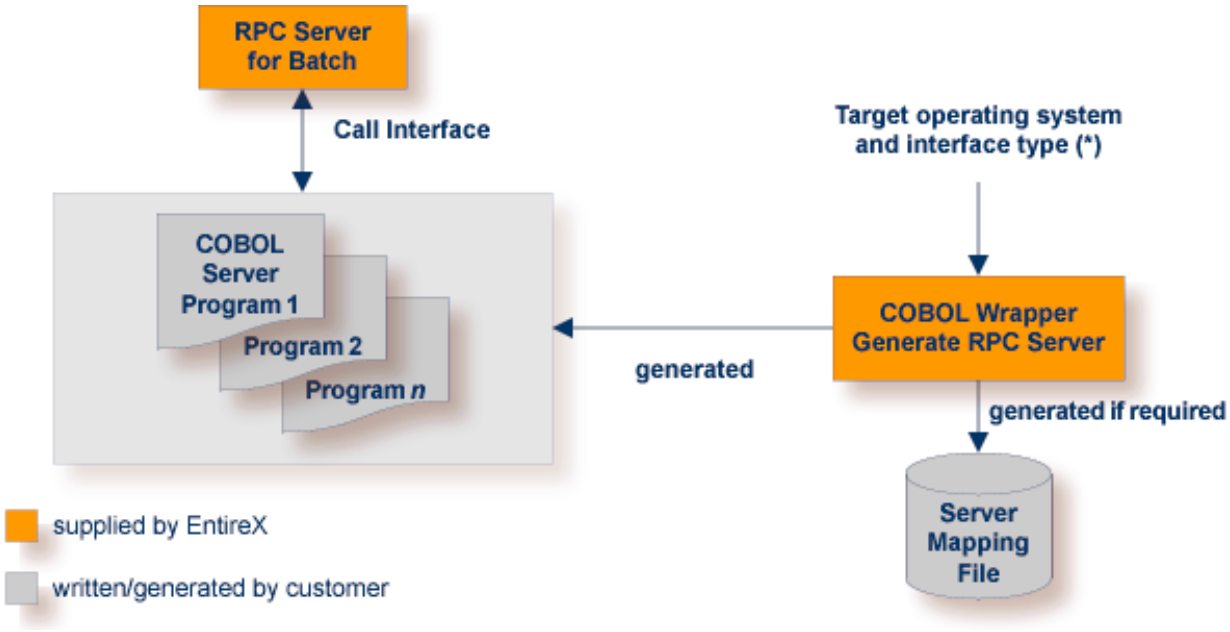
This section covers the following topics:

- Introduction
- Steps

See also *COBOL Scenarios* (z/OS | BS2000) in the RPC Server for Batch documentation.

Introduction

This mode applies to z/OS, BS2000 and IBM i.



(*) See *Target Operating System* and *Server Interface Types* under *Generating COBOL Source Files from Software AG IDL Files*.

In batch mode, the RPC server sets up all of your server's parameters dynamically in the format required. Your server is called dynamically using standard call interfaces.

Use the COBOL Wrapper for batch to build servers for the RPC Server for Batch.

Steps

To use the COBOL Wrapper for batch

1 Generate a server (skeleton(s)) for the target operating system, for example "z/OS", and use interface type "Batch with standard linkage calling convention". See ***Generating COBOL Source Files from Software AG IDL Files*** for details.

2 The generated server mapping file has to be provided. A server mapping file is a Designer file with extension .cvm. See *Server Mapping Files for COBOL* in the Designer documentation. The server mapping files are wrapped into RPC clients and provided with the RPC request. You need to rebuild all RPC clients communicating with this RPC server program. Select the appropriate wrapper (see *EntireX Wrappers* in the Designer documentation) and re-generate the client interface objects. For the *EntireX Adapter* you need to update your generated IS adapter as described under *To update an existing connection* in *Step 3: Create or Update an Adapter Connection* in the Integration Server Wrapper documentation.

3 If necessary, use FTP to transfer the server (skeleton(s)) to the target platform where you write your server.

4 Use the generated server (skeleton(s)) and complete it by applying your application logic. Note the information given in ***Software AG IDL to COBOL Mapping***.

- **z/OS**
  See *Returning Application Errors* in the RPC Server for Batch documentation.
- **IBM i**
  Your server has to be implemented as an ILE COBOL program of type *PGM.

5 Use a COBOL compiler supported by the COBOL Wrapper to compile your server.

- **BS2000**
  - The IDL types U or UV require a compiler that supports COBOL data type NATIONAL. See *BS2000 Prerequisites* for more information on supported compilers.
  - Compile them as OM or LLM modules.
- **IBM i**
  - Use the IBM i command `CRTCBLMOD` (create bound COBOL module).
  - As an alternative, you can compile and bind in one step, see the next step below.
- **Other Platforms**
  Use the standard COBOL compiler of the target platform.

6 Link (bind) your server to an executable program. Give the resulting server program the same name as the `program-name` in the IDL file. See `program-definition` under *Software AG IDL Grammar* in the IDL Editor documentation.

- **BS2000**
  There is no need to link the server modules with the BS2000 Common Runtime Environment (CRTE). The CRTE is included in the server's BLSLIB chain and loaded dynamically. If this is needed for any reason, the CRTE must be linked as a subsystem. All entries must be hidden to prevent duplicates. Linking the CRTE statically will consume resources and slow down the load time of the server modules.
- **IBM i**
  - Bind it as a dynamically callable program of type *PGM using the command `CRTPGM`. Make sure your server fulfills all requirements to be callable by the IBM i host server.
  - As an alternative to compiling with `CRTCBLMOD` (see step above) and binding with `CRTPGM` separately, you can compile and bind in one step with the command `CRTBNDCBL`.
- **Other Platforms**
  Use the standard linker (binder) of the target platform.

7 Provide your server to the endpoint, depending on the platform:

- **z/OS and BS2000**
  Add the server to the RPC Server for Batch STEPLIB chain.
- **IBM i**
  On your IBM i host server, put the server into a library corresponding to `as400.programpath`. See *Configuring the IBM AS/400 Side* in the RPC Server for AS/400 documentation.

8 Test your server.

- If you are using a server mapping file (.cvm), the server mapping is taken from the RPC request and the program with the COBOL name (see Note) defined in the server mapping is executed. See ***Customize Automatically Generated Server Names***. If no corresponding program can be found, the access will fail.
- If no server mapping file is required (see *When is a Server Mapping File Required?* in the Designer documentation) or the server is generated with a previous version of EntireX without support for server mapping - the library name (see `library-definition` under *Software AG IDL Grammar* in the IDL Editor documentation) given in the IDL is ignored (see Note).

  Example: If a client performs an RPC request that is based on the IDL program name CALC, the RPC server will dynamically try to execute a program CALC. If no corresponding program can be found, the access will fail.

Note: For IBM i this also depends on the setting `as400.programpath`. See *Configuring the IBM AS/400 Side* in the RPC Server for AS/400 documentation.

Using the COBOL Wrapper for IMS BMP (z/OS)

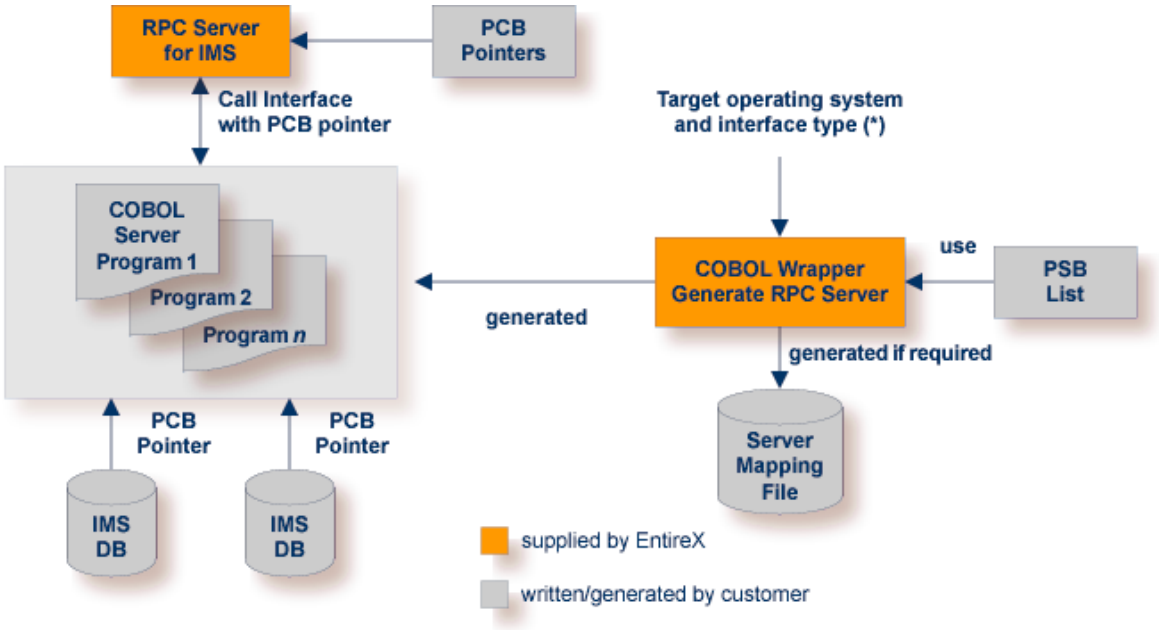
This section covers the following topics:

- Introduction
- Steps

See also *COBOL Scenarios* in the RPC Server for IMS documentation.

Introduction

This mode applies to z/OS IMS mode BMP.



(*)See *Target Operating System* and *Server Interface Types* under *Generating COBOL Source Files from Software AG IDL Files*.

In IMS BMP, the RPC Server for IMS sets up all of your server's parameters dynamically in the format required. Your server is called dynamically using standard call interfaces. IMS-specific PCB pointers can be provided as parameters in the linkage section.

Use the COBOL Wrapper for IMS BMP if you need to

- access IMS BMP programs with standard linkage calling convention
- access IMS databases through IMS PCB pointers and to pass them via parameters in the linkage section

- access the IMS PCB pointer IOPCB, for example to print data or to start an asynchronous transaction
- use the COBOL/ DLI interface module “CBLTDLI” which requires PCB pointers in its interface.

If PCB pointers have to be provided as parameters in the COBOL linkage section of your server, your IDL must comply with the IMS PCB Pointer IDL rules listed below. If no PCB pointers are required, the rules can be skipped.

IMS PCB Pointer IDL Rules

- An IMS PSB list contains the PCB pointers of your environment:
  - The IMS PSB list is a text file and can be created with any text editor.
  - Only one PCB pointer is listed per line.
  - The PCB pointer `IOPCB` is always the first pointer in the IMS PSB list.
  - The PCB pointers (except `IOPCB`) match the related PSB generation for your server.
  - The PCB pointers listed match the PCB pointers provided at runtime to the RPC Server for IMS (including `IOPCB`) in number and sequence.
  - The IMS PSB list is assigned in the IDL properties, see *Generating COBOL Source Files from Software AG IDL Files* or IDL *Generation Settings - Preferences*. Example:

```
IOPCB
DBPCB
```

- PCB pointers are described in the IDL as parameters. Thus they can be accessed in your server as any other parameter. Additionally, the following is required:
  - IDL parameters that are PCB pointers are marked with the attribute `IMS` (see `attribute-list` under *Software AG IDL Grammar* in the IDL Editor documentation).
  - IDL parameters that are PCB pointers must match a PCB pointer listed in the IMS PSB list, otherwise the RPC Server for IMS does not pass them as PCB pointers at runtime. This results in unexpected behavior. Example:

```
Library 'IMSDB' Is
   Program ' IMSDB' Is
      Define Data Parameter
      1 IN-COMMAND          (A3)   IN /* ADD, DEL, DIS
      1 IO-DATA                    IN OUT
        2 IO-LAST-NAME      (A10)
        2 IO-FIRST-NAME     (A10)
        2 IO-EXTENSION      (A10)
        2 IO-ZIP-CODE       (A07)
      1 DBPCB                      IN IMS /* this is a PCB pointer
        2 DBNAME            (A8)
        2 SEG-LEVEL-NO      (A2)
        2 DBSTATUS          (A2)
```

```
      2 FILLER1           (A20)
    1 OUT-MESSAGE         (A40)  OUT
    End-Define
```

Steps

To use the COBOL Wrapper for IMS BMP

1 Generate the server (skeleton(s)) for the target operating system "z/OS", use interface type "IMS BMP with standard linkage calling convention". If PCB pointers should be provided as COBOL linkage section parameters for your server, set the IMS PSB list; otherwise omit the IMS PSB list. See ***Generating COBOL Source Files from Software AG IDL Files***.

2 The generated server mapping file has to be provided. A server mapping file is a Designer file with extension .cvm. See *Server Mapping Files for COBOL* in the Designer documentation. The server mapping files are wrapped into RPC clients and provided with the RPC request. You need to rebuild all RPC clients communicating with this RPC server program. Select the appropriate wrapper (see *EntireX Wrappers* in the Designer documentation) and re-generate the client interface objects. For the *EntireX Adapter* you need to update your generated IS adapter as described under *To update an existing connection* in *Step 3: Create or Update an Adapter Connection* in the Integration Server Wrapper documentation.

3 If necessary, use FTP to transfer the server (skeleton(s)) to the target platform where you write your server.

4 Use the generated server (skeleton(s)) and complete it by applying your application logic. You can use the IMS-specific PCB pointers in your server as usual. Note the information given in ***Software AG IDL to COBOL Mapping*** and *Returning Application Errors* in the RPC Server for IMS documentation.

5 Using a COBOL compiler supported by the COBOL Wrapper, compile your server.

6 Link (bind) the server to an executable program, using the standard linker (binder) of the target program.

- Give the resulting server program the same name as the program in the IDL file (see `program-definition` under *Software AG IDL Grammar* in the IDL Editor documentation).

7 Provide the server to the RPC Server for IMS.

- Add the server to the RPC Server for IMS STEPLIB chain.

8 Test your server.

- If you are using a server mapping file (.cvm), the server mapping is taken from the RPC request and the program with the COBOL name (see Note) defined in the server mapping is executed. See ***Customize Automatically Generated Server Names***. If no corresponding program can be found, the access will fail.

- If no server mapping file is required (see *When is a Server Mapping File Required?* in the Designer documentation) or the server is generated with a previous version of EntireX without support for server mapping - the library name (see `library-definition` under *Software AG IDL Grammar* in the IDL Editor documentation) given in the IDL is ignored (see Note).

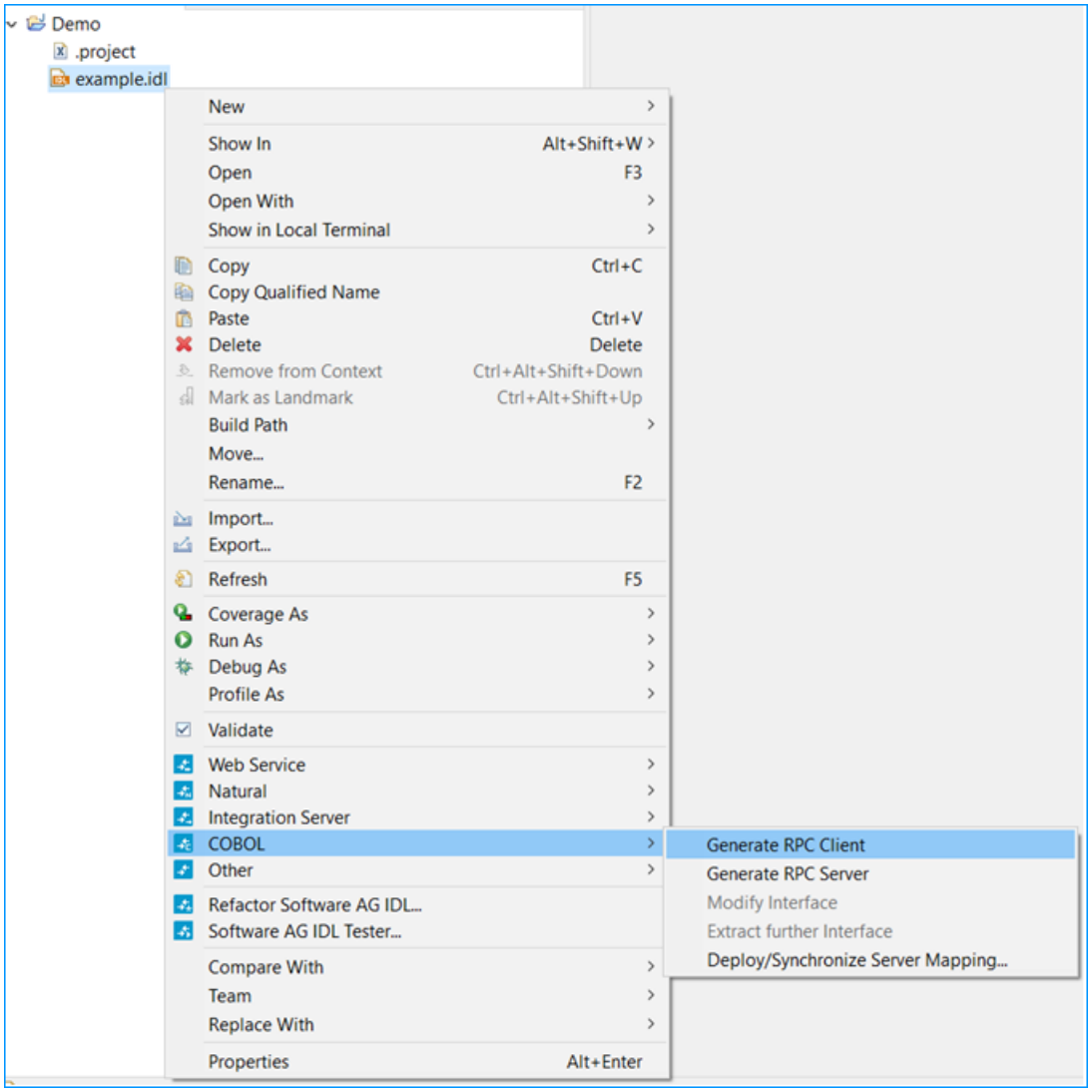
  Example: If a client performs an RPC request that is based on the IDL program name CALC, the RPC server will dynamically try to execute a program CALC. If no corresponding program can be found, the access will fail.

5 Generating COBOL Source Files from Software AG IDL Files

This chapter describes how to generate COBOL source files from Software AG IDL files.

Select an IDL File and Generate RPC Client or RPC Server

From the context menu, choose **COBOL > Generate RPC Client** and **Generate RPC Server** to generate the COBOL source files.



Note: In command-line mode, use command `-cobol:client` or `-cobol:server`. See *Using the COBOL Wrapper in Command-line Mode*. Note that existing files will always be over-written.

Results for **RPC client**:

- The folders *client* and *include* are created as subfolders to the **IDL-specific Output Folder** defined in the *Generation Settings - Properties*.
- The *client* folder contains the following:
  - client interface objects
  - optionally the generic RPC service module, which is only generated if the option **Generate Generic RPC Service Module COBSRVI** is set, see *Generation and Usage of Generic RPC Service Module COBSRVI*.
- The *include* folder contains the following:
  - the associated copybooks
  - the RPC communication area copybook `ERXCOMM`
  - optionally its extension copybook `ERXVSTR`
  - optionally the copybooks `COBINIT` and `COBEXIT`

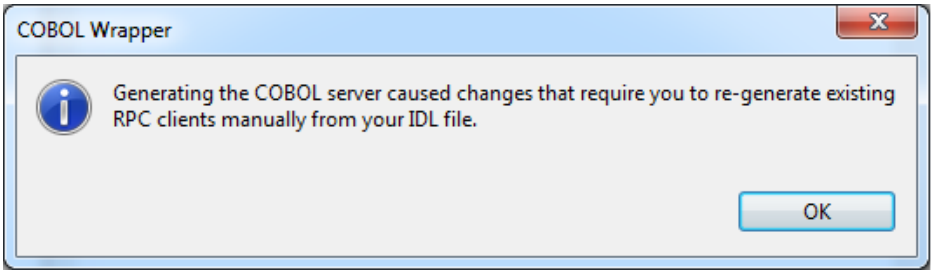
  For further information on the usage of copybooks generated in this folder, see *Using the Generated Copybooks*.

Results for **RPC server**:

- The folder *server* is created as a subfolder to the **IDL-specific Output Folder** defined in the *Generation Settings - Properties*. It contains the RPC server skeletons.

  Caution: Take care not to overwrite an existing RPC server implementation with an RPC server skeleton. We recommend moving your RPC server implementation to a different folder.

- If required, a server mapping file is generated, too. See *When is a Server Mapping File Required?* under *Server Mapping Files for COBOL* in the Designer documentation. The following dialog is displayed.

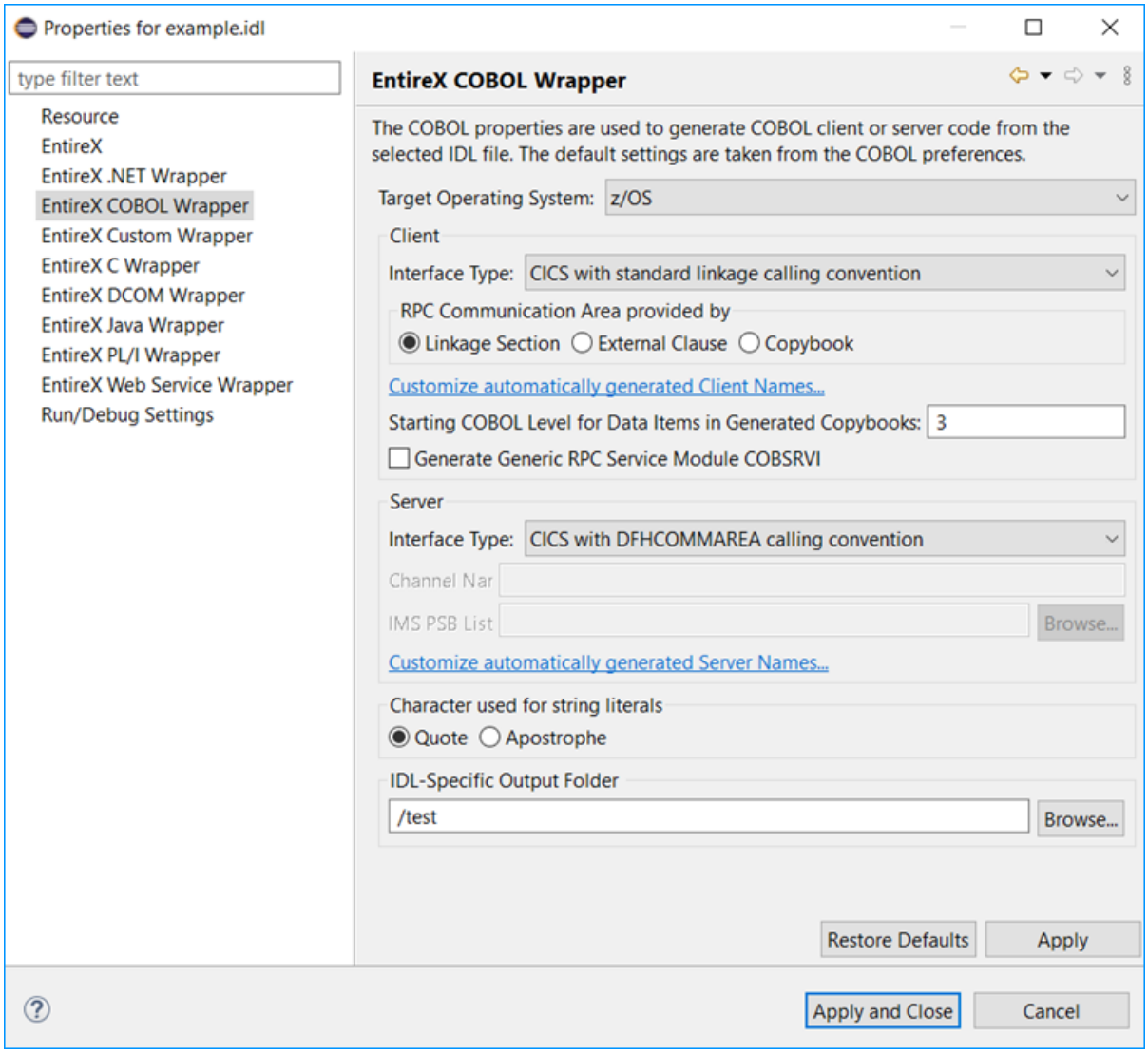


  You need to rebuild all RPC clients communicating with this RPC server program. Select the appropriate wrapper (see *EntireX Wrappers* in the Designer documentation) and re-generate the client interface objects. For the *EntireX Adapter* you need to update your generated IS adapter as described under *To update an existing connection* in *Step 3: Create or Update an Adapter Connection* in the Integration Server Wrapper documentation.

Generation Settings - Properties

Introduction

Whenever a new IDL file is created, defaults for the properties are copied from the preferences. See *Generation Settings - Preferences*. To set individual properties per IDL file for COBOL Wrapper generation, use the **Properties** wizard of the IDL file. The **Target Operating System** and the **Interface Type** are essential. They determine if other parameters such as **RPC Communication Area provided by** can be set or have to remain fixed. The parameter **IDL-specific Output** defines the location to store the source file subfolders. **Target Operating System** determines whether file extensions are generated or not.



In the following, we give a detailed description of the properties that need to be set for each type of generation:

- **For client and server generation:**
  - *Target Operating System*
  - *Characters Used for String Literals*
  - *IDL-specific Output Folder*
- **For client generation only:**
  - *Client Interface Types*
  - *Customize Automatically Generated Client Names*
  - *Starting COBOL Level for Data Items in Generated Copybooks*
  - *RPC Communication Area*

- *Generation and Usage of Generic RPC Service Module COBSRVI*

- **For server generation only:**
  - *Server Interface Types*
  - *Customize Automatically Generated Server Names*
  - *IMS PSB List*
  - *Channel Name*

Target Operating System

Select the target operating system for which COBOL code is to be generated. See *Platform Coverage* for a full list of supported operating system versions.

Value	Description
z/OS	IBM z/OS operating system.
z/VSE	IBM z/VSE operating system.
BS2000	Fujitsu Siemens BS2000 operating system.
IBM i	IBM i operating system.

Characters Used for String Literals

With this option you can specify how string literals are specified in the generated COBOL code. See your COBOL compiler documentation for information on how string literals are enclosed.

Value	Description
Quote	String literals will be enclosed in double quotes in the generated COBOL code.
Apostrophe	String literals will be enclosed in apostrophes (single quotes) in the generated COBOL code.

IDL-specific Output Folder

This field specifies the folder where the COBOL files will be stored, by default in the same folder as the IDL file. For a non-default location, enter another folder name or choose **Browse...**.

Client Interface Types

Interface Type	Target Operating System	More Information	Notes
CICS with Standard Linkage Calling Convention	z/OS, z/VSE	Follow the steps under *Using the COBOL Wrapper for CICS with Call Interfaces (z/OS and z/VSE)*.	1
CICS with DFHCOMMAREA Calling Convention	z/OS, z/VSE	Follow the steps under *Using the COBOL Wrapper for CICS with DFHCOMMAREA Calling Convention (z/OS and z/VSE)*.	2
Batch with Standard Linkage Calling Convention	z/OS, BS2000, z/VSE, IBM i	Follow the steps under *Using the COBOL Wrapper for Batch (z/OS, BS2000, z/VSE and IBM i)*.	1
IMS BMP with Standard Linkage Calling Convention	z/OS	Follow the steps under *Using the COBOL Wrapper for IMS (z/OS)*.	1
IMS MPP with Standard Linkage Calling Convention	z/OS	Follow the steps under *Using the COBOL Wrapper for IMS (z/OS)*.	1
IDMS/DC with Standard Linkage Calling Convention	z/OS	Follow the steps under *Using the COBOL Wrapper for IDMS/DC with Call Interfaces (z/OS)*.	1

Notes:

1. Use this option if you want to build an RPC client application that calls the client interface object(s) and the generic RPC module `COBSRVI` with a standard linkage interface.
2. Use this option if you want to build a CICS RPC client application that calls the client interface object(s) and the generic RPC module `COBSRVI` with a DFHCOMMAREA interface.

Customize Automatically Generated Client Names

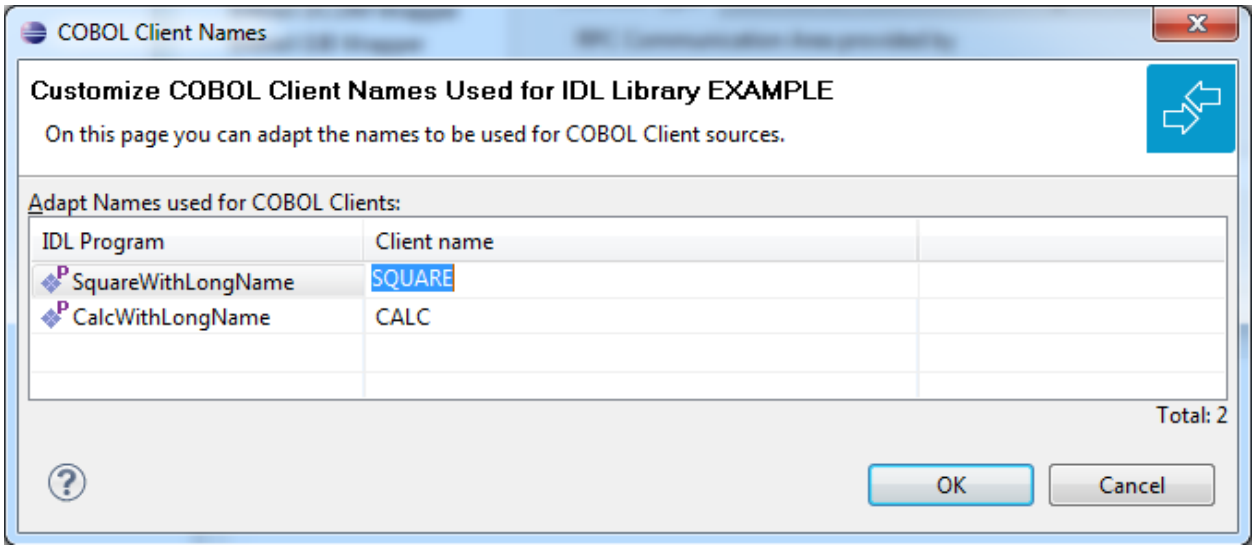
If you open the link **Customize automatically generated Client Names** on the **Properties** page you can adapt the names for the COBOL client interface objects (subprograms). When you call the page the first time, COBOL names are suggested based on the IDL program (`program-definition` under *Software AG IDL Grammar* in the IDL Editor documentation) or IDL program alias names. The page varies, depending on whether the target COBOL environment supports long COBOL names or not:

- z/OS and z/VSE
- IBM i

- BS2000

z/OS and z/VSE

Max. 8 characters (short names) are supported as COBOL names:



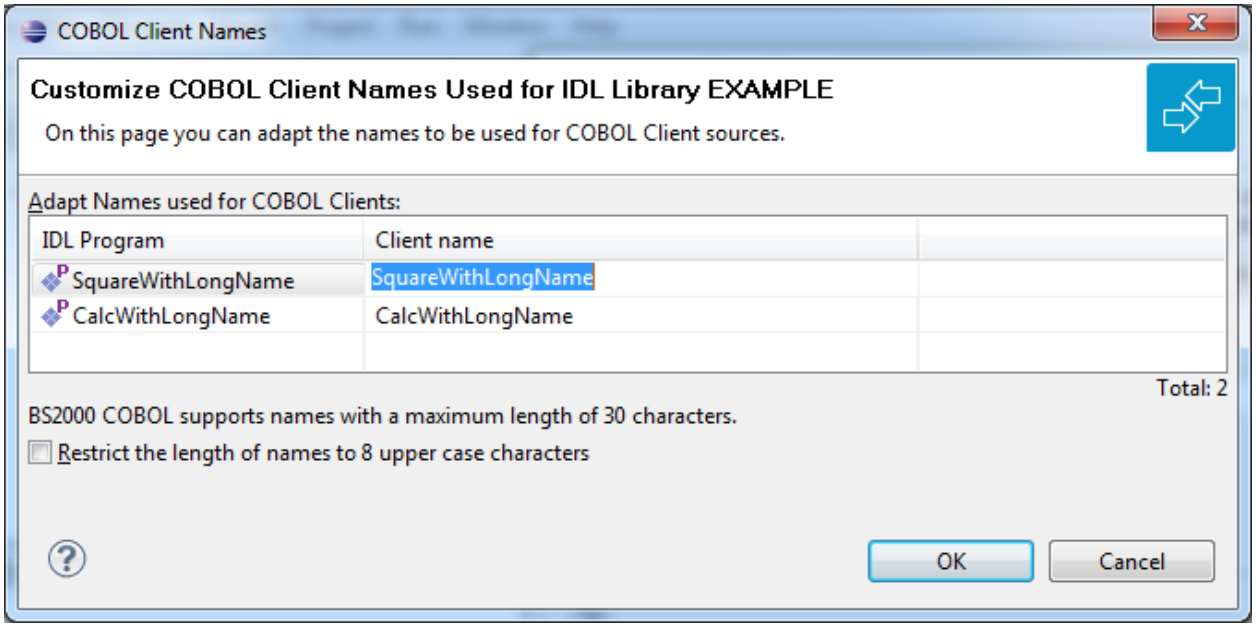
Note: If your IDL file contains more than one IDL library, the additional column **IDL Library** is displayed.

IBM i

Customization of client names for IBM i is the same as for z/OS and z/VSE. See *z/OS and z/VSE*.

BS2000

Max. 30 characters are supported as COBOL names. By default, names are generated with a maximum of 8 characters (short names).



Notes:

1. If your IDL file contains more than one IDL library, the additional column **IDL Library** is displayed.
2. With the check box **Restrict the length of names to 8 characters** you can flip between short names and long names. Both sorts of names (short and long) are stored in the property file. For generation you have to decide if short or long names are to be used.

Starting COBOL Level for Data Items in Generated Copybooks

With this option you can specify the starting COBOL level used in the generated copybooks for COBOL data items.

See *Using the Generated Copybooks* for syntax examples.

Specify a valid COBOL level in the range 1-49. The COBOL programming language maximum of 49 subtracted by the specified level must provide enough levels to hold all IDL levels. Note that IDL types may consume more than one COBOL level, for example:

- IDL unbounded groups require a COBOL level for every dimension. If they are defined on IDL level 1, an extra COBOL level is required
- IDL unbounded arrays require a COBOL level for every dimension plus one extra COBOL level
- some basic (scalar) IDL data types need extra COBOL levels

Notes:

1. Do not specify a level too deep because you may exceed the COBOL programming language maximum of 49 and the generated copybook cannot be compiled.
2. For compatibility with *Client and Server Examples for z/OS CICS*, the level must be 3 or above.
3. For compatibility with all other delivered examples, the level must be 2 or above.

RPC Communication Area

The RPC communication area copybook `ERXCOMM` and its extension `ERXVSTR` are used to specify parameters that are needed to communicate with the broker and are not specific to client interface objects. These are for example the broker ID, client parameters such as `userID` and `password` and the server address such as *`class/servername/service`* etc.

Value	Description	Notes
`External Clause`	This kind of RPC communication area usage applies to the scenarios *CICS* | *Batch* | *IMS*. The RPC communication area copybooks are defined in the working storage section as COBOL data items with the `EXTERNAL` clause in the RPC client application. They are passed with the `EXTERNAL` clause to and the generated client interface object(s).	1
`Linkage Section`	This kind of RPC communication area usage applies to the scenarios *CICS* | *Batch* | *IMS*. The RPC communication area copybooks are defined in the working storage section as COBOL data items. They are passed via additional parameter between your RPC client application and the generated client interface object(s).	2
`Copybook`	This kind of RPC communication area usage is available in the z/OS operating system. Refer to the scenarios *CICS* | *Batch* | *IMS*. The RPC communication area copybooks are provided inside the generated client interface object(s). It is not visible in the RPC client application - it is local to the client interface objects. Default values are retrieved from Designer preferences or IDL-specific properties and can be overwritten in the copybook `COBINIT` (see folder *include*).	2

Notes:

1. The client interface objects are statically linked to the RPC client. It is not possible to call them dynamically.
2. 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.

Generation and Usage of Generic RPC Service Module COBSRVI

The generic RPC service module COBSRVI can be optionally generated in the folder *client* in the container folder. It acts as a runtime for RPC communication. See ***Generic RPC Services Module*** under *Introduction to the COBOL Wrapper*. The module depends on target environment (CICS, Batch...), client interface type (see ***Client Interface Types***) and operating system (z/OS, z/VSE...). Use this option to control the generation of this module.

Handling depends on the interface type:

- **CICS with DFHCOMMAREA calling convention**
  For this interface type, the generic RPC service module is installed once within CICS as a CICS program and shared by all RPC clients using this interface type. Details and the architecture of this scenario are described under ***Using the COBOL Wrapper for CICS with `DFHCOMMAREA` Calling Convention (z/OS and z/VSE)***.
  - *Check* this option if you want to install or replace the installed generic RPC service module in CICS with the version generated by the COBOL Wrapper. This makes sense in the following situations:
    - You have *not* installed the RPC examples on z/OS, because installation of this module is part of *Installing RPC Examples for z/OS*.
    - You need an update of the generic RPC service module because of a newer COBOL Wrapper version, for example an Eclipse update without mainframe update. For compile job and CICS CSD definitions see *Delivered Modules* for **z/OS** | **z/VSE**.
  - *Clear* this option if you have already installed the generic RPC service module in CICS (already installed RPC examples for z/OS or previous COBOL Wrapper project) and do not want to re-install it in CICS. This prevents the generation of the generic RPC service module.
- **All other calling conventions**
  - The preferred approach is to *check* this option. This will generate the generic RPC service module. The generated module is part of your client application. Its usage is described under *Using the COBOL Wrapper with a Call Interface* (CICS | Batch | IMS).
  - *Clear* this option if you can reuse the generic RPC service module from a previous COBOL Wrapper project. This will prevent the generation of the generic RPC service module. It is important that ***Target Operating System***, ***Client Interface Types*** and ***Characters Used for String Literals*** are the same.

Customize Automatically Generated Server Names

If you open the link **Customize automatically generated Server Names** on the properties page you can, adapt the names for the COBOL server (subprograms). When you call the page the first time, COBOL names are suggested based on the IDL program (`program-definition` under *Software AG IDL Grammar* in the IDL Editor documentation) or IDL program alias names. For further details on customizing names for the server side, see the platform-specific section under *Customize Automatically Generated Client Names*; the information here also applies to server names:

- ***z/OS and z/VSE***
- ***BS2000***

Notes:

1. Customization of server names is not supported under IBM i.
2. If the server names (automatically generated or customized) differ from the IDL program names, a server mapping file is required (Designer file with extension .cvm). It is generated during generation of RPC server and has to be used in subsequent steps. See *Server Mapping Files for COBOL* and ***Using the COBOL Wrapper for the Server Side***.

Server Interface Types

Interface Type	Target Operating System	Description
CICS with DFHCOMMAREA calling convention	z/OS, z/VSE	Use this option if you want to build a CICS RPC server application with a DFHCOMMAREA interface. Follow the steps under ***Using the COBOL Wrapper for CICS with `DFHCOMMAREA` Calling Convention (z/OS and z/VSE)***.
CICS with Channel Container calling convention	z/OS	Use this option if you want to build a CICS RPC server application with a channel container interface. To specify a channel name, see ***Channel Name***. Follow the steps under ***Using the COBOL Wrapper for CICS with Channel Container Calling Convention (z/OS)***.
CICS with DFHCOMMAREA large buffer interface	z/OS, z/VSE	Use this option if you want to build a CICS RPC server application with a large buffer interface. Follow the steps under ***Using the COBOL Wrapper for CICS with `DFHCOMMAREA` Large Buffer Interface (z/OS and z/VSE)***.
Batch with standard linkage calling convention	z/OS, BS2000, IBM i	Use this option if you want to build an application for an RPC Server for Batch. Follow the steps under ***Using the COBOL Wrapper for Batch (z/OS, BS2000 and IBM i)***.
IMS BMP with standard linkage calling convention	z/OS	Use this option if you want to build an RPC Server for IMS application for IMS BMP mode (no MPP) with standard call interfaces. If your server uses PCB pointers, see ***IMS PSB List***

Interface Type	Target Operating System	Description
		below. Follow the steps under *Using the COBOL Wrapper for IMS BMP (z/OS)*.

IMS PSB List

IMS PSB List applies to the server interface type “IMS BMP with standard linkage calling convention” only. If your server uses PCB pointers and requires that they are passed through the linkage section, an IMS PSB list is required. Your IDL must comply with the rules under *IMS PCB Pointer IDL Rules*. If no PCB pointers are required, omit the IMS PSB list. See *Server Interface Types* for more information.

Channel Name

Channel Name applies to the server interface type "CICS with Channel Container calling convention" only.

If a channel name is specified, the server is

- called with the given channel name
- generated with COBOL code to check for channel name validity.

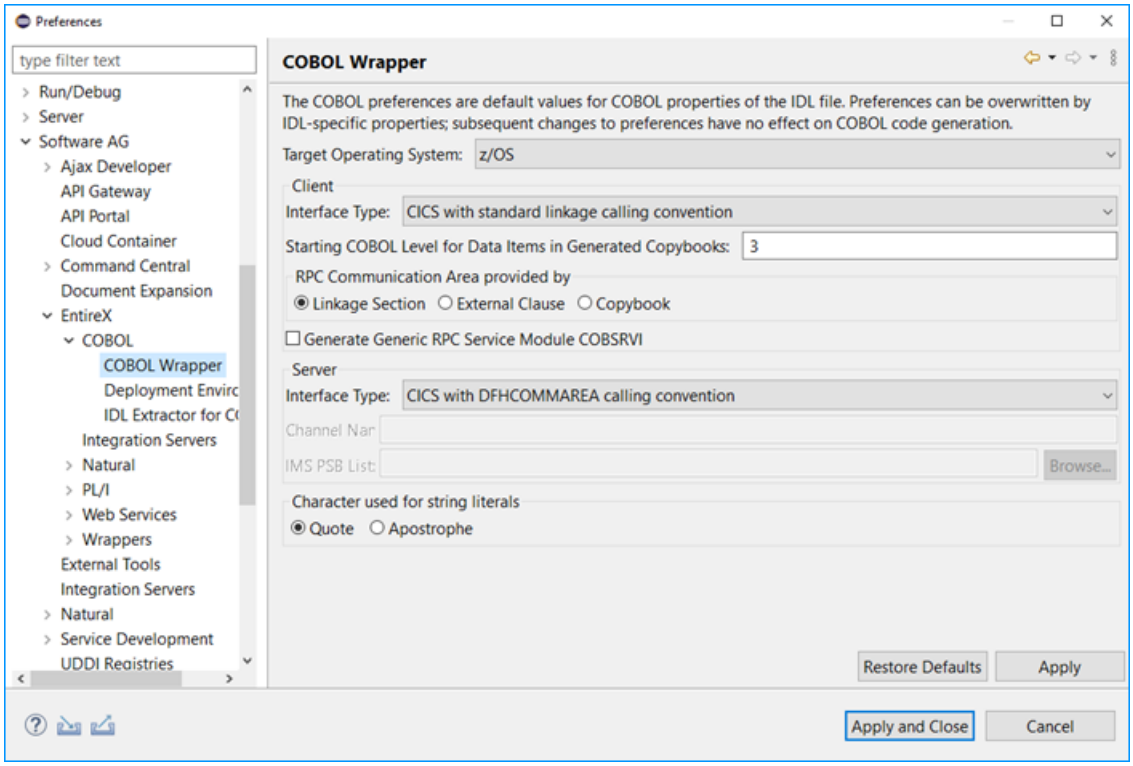
If no channel name is specified, the server is

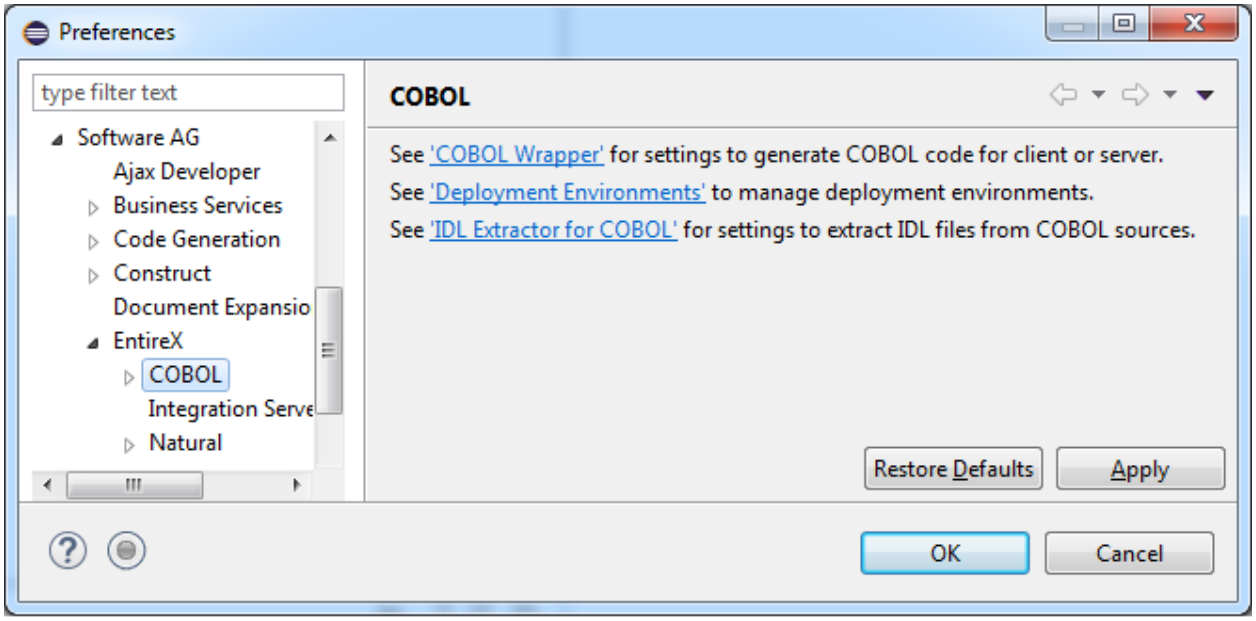
- called with the "EntireXChannel" channel name
- generated without COBOL code to check for channel name validity.

Your IDL must comply with the rules described under *CICS Channel Container IDL Rules*. See *Server Interface Types* for more information.

Generation Settings - Preferences

Use the **Preferences** page of the COBOL Wrapper to set the workspace defaults for the target operating system, interface types etc. The settings (except **Type of COBOL mapping**) are used as the defaults for the IDL properties when a new IDL file is created; see *Generation Settings - Properties*.




Preferences
type filter text
Software AG
Ajax Developer
Business Services
Code Generation
Construct
Document Expansio
EntireX
COBOL
Integration Serve
Natural
COBOL
See 'COBOL Wrapper' for settings to generate COBOL code for client or server.
See 'Deployment Environments' to manage deployment environments.
See 'IDL Extractor for COBOL' for settings to extract IDL files from COBOL sources.
Restore Defaults
Apply
OK
Cancel

6 Using the COBOL Wrapper in Command-line Mode

Commands are available to generate a COBOL RPC client or COBOL RPC server from a specified IDL file.

See also *Command-line Mode* under *Server Mapping Deployment Wizard* in the Designer documentation.

Command-line Options

See *Using EntireX in the Designer Command-line Mode* for the general command-line syntax.

Generate a COBOL RPC Client from IDL File

To generate a COBOL RPC client from the specified IDL file, use the following command with options in table below:

```
-cobol:client
```

Option	Description
`-comm`	The RPC communication area. Valid values: `EXTERNAL`, `LINKAGE`, `COPYBOOK`. See ***RPC Communication Area*** for more information. `EXTERNAL` *`External Clause`* `LINKAGE` *`Linkage Section`* `COPYBOOK` *`Copybook`* For possible combinations with `-target` and `-interface` option, see below.
`-folder`	Folder where the COBOL files will be stored.
`-help`	Display this usage message.
`-interface`	Interface type, either `DFHCOMMAREA` or `LINKAGE`. For possible combinations with `-target` and `-comm` option, see below.
`-literal`	Enclose string literals in quotes or apostrophes. Valid values: `QUOTE`, `APOST`. See ***Characters Used for String Literals*** for more information.
`-target`	Target operating system and environment, one of `BATCH_ZOS`, `BATCH_VSE`, `BATCH_BS2000`, `BATCH_I5OS`, `CICS_ZOS`, `CICS_VSE`, `IMS_MPP`, `IMS_BMP` or `IDMS_ZOS`. See ***Client Interface Types*** for more information. For possible combinations with the `-interface` and `-comm` option.

Option	Description			
	-target	-interface	-comm	Usage for
	CICS_ZOS	LINKAGE	LINKAGE EXTERNAL COPYBOOK	CICS with standard linking calling convention for z/OS.
		DFHCOMMAREA	LINKAGE	CICS with DFHCOMMAREA linkage calling convention for z/OS.
	CICS_VSE	LINKAGE	LINKAGE EXTERNAL	CICS with standard linkage calling convention for z/VSE.
		DFHCOMMAREA	LINKAGE	CICS with DFHCOMMAREA linkage calling convention for z/VSE.
	BATCH_VSE	LINKAGE	LINKAGE EXTERNAL	Batch with standard linkage calling convention for z/VSE.
	BATCH_BS2000	LINKAGE	LINKAGE EXTERNAL	Batch with standard linkage calling convention for BS2000.
	BATCH_I5OS	LINKAGE	LINKAGE EXTERNAL	Batch with standard linkage calling convention for IBM i.
	BATCH_ZOS	LINKAGE	LINKAGE EXTERNAL COPYBOOK	Batch with standard linkage calling convention for z/OS.
	IMS_BMP	LINKAGE	LINKAGE EXTERNAL COPYBOOK	IMS BMP with standard linkage calling convention for z/OS.
	IMS_MPP	LINKAGE	LINKAGE EXTERNAL COPYBOOK	IMS MPP with standard linkage calling convention for z/OS.
	IDMS_ZOS	LINKAGE	LINKAGE EXTERNAL COPYBOOK	IDMS/DC with standard linkage calling convention for z/OS.

Option	Description
`-copybooklevel`	Define the beginning level for COBOL data items in generated copybooks, see *Starting COBOL Level for Data Items in Generated Copybooks*. Valid values: 1-49.
`-rpcservice`	Option to generate the generic RPC service module COBSRVI. See *Generation and Usage of Generic RPC Service Module COBSRVI*. Valid values: `TRUE` - Generate generic RPC service module. `FALSE` - Do not generate the generic RPC service module.

Generate a COBOL RPC Server from IDL File

To generate a COBOL RPC server from the specified IDL file, use the following command with options in table below:

```
-cobol:server
```

Option	Description
`-channel`	A CICS channel name can be provided for the interface type 'CICS with Channel Container calling convention'. See *Using the COBOL Wrapper for CICS with Channel Container Calling Convention (z/OS)*. See also *Channel Name*.
`-folder`	Folder where the COBOL files will be stored.
`-help`	Display this usage message.
`-interface`	Interface type, one of `DFHCOMMAREA`, `DFHLBUFFER`, `DFHCHANNEL` or `LINKAGE`. See table below for possible combinations.
`-literal`	Enclose string literals in quotes or apostrophes. See *Characters Used for String Literals*.
`-target`	Target operating system and environment. For possible combinations with option `-interface`, see below and also *Server Interface Types*.

-target	-interface	Usage for
`CICS_ZOS`	`DFHCOMMAREA`	CICS with DFHCOMMAREA calling convention for z/OS.
	`DFHLBUFFER`	CICS with DFHCOMMAREA large buffer interface for z/OS.
	`DFHCHANNEL`	CICS with Channel Container calling convention for z/OS.
`CICS_VSE`	`DFHCOMMAREA`	CICS with DFHCOMMAREA calling convention for z/VSE.
	`DFHLBUFFER`	CICS with DFHCOMMAREA large buffer interface for z/VSE.
`BATCH_VSE`	`LINKAGE`	Batch with standard linkage calling convention for z/VSE.
`BATCH_BS2000`	`LINKAGE`	Batch with standard linkage calling convention for BS2000.
`BATCH_I5OS`	`LINKAGE`	Batch with standard linkage calling convention for IBM i.
`BATCH_ZOS`	`LINKAGE`	Batch with standard linkage calling convention for z/OS.
`IMS_BMP`	`LINKAGE`	IMS BMP with standard linkage calling convention for z/OS. This target may require a PSBLIST. See below.

Option	Description
`-psblist`	An IMS PSB list containing IMS PCB pointers can be provided for the server interface type *IMS BMP with standard linkage calling convention*. See ***Using the COBOL Wrapper for IMS BMP (z/OS)*** for scenarios on PCB pointer usage. See also ***IMS PSB List***.

Example Generating an RPC Client

```
<workbench> -cobol:client /Demo/example.idl -target CICS_ZOS
```

where *<workbench>* is a placeholder for the actual EntireX design-time starter as described under *Using EntireX in the Designer Command-line Mode*.

The name of the IDL file includes the project name. In the example, the project *Demo* is used. If the IDL file name describes a file inside the Eclipse workspace, the name is case-sensitive.

If the first part of the IDL file name is not a project name in the current workspace, the IDL file name is used as a relative (based on the IDL file) or absolute file name in the file system. Thus, the IDL files do not need to be part of an Eclipse project.

If you do not specify a folder (option `-folder`), the generated COBOL source files (client interface objects and the client declarations) will be stored in parallel to the IDL file, in the generated sub-folders *client* and *include*, e.g. *Demo/client* and *Demo/include*.

Example Generating an RPC Server

```
<workbench> -cobol:server /Demo/example.idl -target CICS_ZOS
```

where *<workbench>* is a placeholder for the actual EntireX design-time starter as described under *Using EntireX in the Designer Command-line Mode*.

The generated COBOL source files (server (skeletons))

- will be stored in parallel to the IDL file, in the generated subfolder *server*, e.g. *Demo/server*.
- will overwrite existing files from a previous command-line mode generation.

Caution: Take care not to overwrite an existing server implementation with a server skeleton. We recommend you to move your server implementation to a different folder.

Further Examples

Windows

Example 1

```
<workbench> -cobol:client C:\Temp\example.idl -folder src -target CICS_ZOS
```

Uses the IDL file *C:\Temp\example.idl* and generates the COBOL source files to the subfolder *src* of the IDL file. Slashes and backslashes are permitted in the file name. Output to standard output:

```
Using workspace file:\C:\myWorkspace\.
Run COBOL client wrapper with C:/Temp/example.idl and target CICS_ZOS.
Processing IDL file C:/Temp/example.idl
Store COBOL Source (1/2): C:\Temp\src/include/CALC
Store COBOL Source (2/2): C:\Temp\src/client/CALC
Exit value: 0
```

Example 2

```
<workbench> -cobol:client C:\Temp\*idl -folder C:\Temp\src -target CICS_ZOS
```

Generates COBOL source files for all IDL files in *C:\Temp*.

Example 3

```
<workbench> -cobol:client /Demo/example.idl -target CICS_ZOS
```

Uses the IDL file */Demo/example.idl* and generates the COBOL source files in parallel to the IDL file, here to the project */Demo*.

Example 4

```
<workbench> -cobol:client -help
```

or

```
<workbench> -help -cobol:client
```

Both calls result in displaying a short help for the COBOL client wrapper.

Linux

Example 1

```
<workbench> -cobol:client /Demo/example.idl -folder src -target CICS_ZOS
```

If the project *Demo* exists in the workspace and *example.idl* exists in this project, this file is used. Otherwise, /*Demo*/*example.idl* is used from file system. The generated output will be stored in /*Demo*/*src*, the subfolder of /*Demo*.

Example 2

```
<workbench> -cobol:client /Demo/*.idl -folder src -target CICS_ZOS
```

Generates COBOL client interface objects for all IDL files in project *Demo* (or in folder /*Demo* if the project does not exist). The generated files are in /*Demo*/*src*.

Example 3

```
<workbench> -cobol:client -help
```

or

```
<workbench> -help -cobol:client
```

Both calls result in displaying a short help for the COBOL client wrapper.

7 Software AG IDL to COBOL Mapping

This chapter describes the specific mapping of Software AG IDL data types, groups, arrays and structures to the COBOL programming language. See also hints and restrictions on the Software AG IDL data types valid for all programming language bindings under *IDL Data Types* in the IDL Editor documentation.

Mapping IDL Data Types to COBOL Data Types

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

- The metasymbols "[" and "]" enclose optional lexical entities.
- The informal term *number* (or in some cases *number1*.*number2*) is a sequence of numeric characters, for example 123.

Software AG IDL	Description	COBOL Data Type	Note
A*number*	Alphanumeric	PIC X(*number*)	
AV	Alphanumeric variable length	not supported	
AV[*number*]	Alphanumeric variable length with maximum length	PIC X(*number*)	13
B*number*	Binary	PIC X(*number*)	11
BV	Binary variable length	not supported	
BV[*number*]	Binary variable length with maximum length	PIC X(*number*)	11,13
D	Date	PIC 9(8)	1
F4	Floating point (small)	USAGE COMP-1	4
F8	Floating point (large)	USAGE COMP-2	4
I1	Integer (small)	PIC X	9,11
I2	Integer (medium)	PIC S9(4) BINARY	10,12
I4	Integer (large)	PIC S9(9) BINARY	10,12
K*number*	Kanji	PIC G(*number*/2) DISPLAY-1	5
KV	Kanji variable length	not supported	

Software AG IDL	Description	COBOL Data Type	Note
KV[*number*]	Kanji variable length with maximum length	PIC G(*number*/2 DISPLAY-1)	5,13
L	Logical	PIC X	6,7
N*number1*[.*number2*]	Unpacked decimal	PIC S9(*number1*) [V(*number2*)]	2
NU*number1*[.*number2*]	Unpacked decimal unsigned	PIC 9(*number1*) [V(*number2*)]	2
P*number1*[.*number2*]	Packed decimal	PIC S9(*number1*) [V(*number2*)] PACKED-DECIMAL	2
PU*number1*[.*number2*]	Packed decimal unsigned	PIC 9(*number1*) [V(*number2*)] PACKED-DECIMAL	2
T	Time	PIC 9(15)	3
U*number*	Unicode	PIC N(*number*) NATIONAL	8
UV	Unicode variable length	not supported	
UV*number*	Unicode variable length with maximum length	PIC N(*number*) NATIONAL	8,13

See also hints and restrictions on the Software AG IDL data types valid for all programming language bindings under *IDL Data Types* in the IDL Editor documentation.

Notes:

1. The date corresponds to the format PIC 9(8). The value contained has the form YYYYMMDD. This form corresponds to COBOL DATE functions. This is an IBM extension of COBOL85 standard.
2. For COBOL, the total number of digits (*number1*+*number2*) is lower than the maximum of 99 that EntireX supports. See *IDL Data Types*. It varies by operating system and COBOL compiler. To enable more total number of digits than 18, a compiler directive (option) may be required.

   - **z/OS**
     The total number of digits (*number1*+*number2*) is restricted to 31 digits. The compiler option AR(E) is generated into the client interface objects and server skeletons if more than 18 digits are defined in the IDL.
   - **BS2000**
     The total number of digits (*number1*+*number2*) is restricted to 31 digits.
   - **z/VSE**
     The total number of digits (*number1*+*number2*) is restricted to 18 digits.

- **Other Operating Systems or Compilers**
  Refer to your COBOL compiler documentation to see whether compiler directives or options exist.

If you connect two endpoints, the total number of digits used must be lower or equal than the maxima of both endpoints. For the supported total number of digits for endpoints, see the notes under data types N, NU, P and PU in section *Mapping IDL Data Types* in the respective Wrapper or language-specific documentation.

3. The time corresponds to the format `PIC 9(15)`. The value contained has the form `YYYYMMDDHHIISST`. This form corresponds to COBOL `DATE/TIME` functions.
4. When floating-point data types are used, rounding errors can occur, so that the values of senders and receivers might differ slightly.
5. The length for IDL data type is given in bytes. For COBOL the length is in DBCS characters (2 bytes). IDL data type `K` is not supported under BS2000 because Fujitsu Siemens compilers do not support DBCS.
6. To inspect the Boolean value of a data item of IDL type Logical, you can specify PIC X followed by condition names (similar code is generated for scalar logical IDL types):

```
level-number data-name PIC X.
88           data-name-false value X'00'.
88           data-name-true  value X'01' thru X'FF'.
```

- **IBM i**
  The `SYMBOLIC CHARACTERS` clause in the `SPECIAL-NAMES` paragraph is not supported. The following COBOL statements demonstrate how you can define alternatively a character, named `HEX-00`, with a value of hexadecimal zero to be used for comparison:

```
WORKING-STORAGE SECTION.
01  HEX-00-B           PIC 9(4) BINARY VALUE 0.
01  HEX-00-H REDEFINES HEX-00-B.
   02  FILLER         PIC X.
   02  HEX-00         PIC X.
```

7. To set the Boolean value of a Logical data item, specify the following hexadecimal values in a one-byte data field (e.g. defined as `PIC X`.):
   - Case False: `Move X'00' to` *`data-name`*.
   - Case True: `Move X'01' to` *`data-name`*.
8. The length is given in Unicode code units following the Unicode standard UTF-16.
   - **z/OS and IBM Compiler**
     Unicode requires the IBM Enterprise compiler.

     Unicode is represented in UTF-16 big-endian format (CCSID 1200).

- **BS2000**
  Unicode requires a compiler that supports COBOL data type NATIONAL. See *BS2000 Prerequisites*.

  Unicode is represented in UTF-16 big-endian format.
- **Other Operating Systems or Compilers**
  Refer to your COBOL compiler documentation.

9. COBOL for operating systems z/OS, BS2000, z/VSE and IBM i does not have a corresponding data type for a compatible I1 mapping. The mapping to COBOL `PIC X` data type should be seen as a `FILLER` variable. If including an I1 data type into the interface is required, it is your responsibility as application developer to process the content of this parameter provided (during receive) and expected (during send) correctly. Negative values are given as the two's complement binary number.
10. The value range for COBOL data type `BINARY` on z/OS, BS2000, z/VSE and IBM i depends on the COBOL compiler settings:
    - With COBOL 85 standard, the mapped COBOL data type `BINARY` is more restrictive than the IDL data types I2 and I4. See *IDL Data Types*. This means that COBOL RPC clients cannot send (and COBOL RPC servers cannot return) the full value range defined by the IDL types I2 and I4. On the other hand, COBOL RPC clients and COBOL RPC servers may receive a value range (from a non-COBOL RPC partner) outside of the value range of your COBOL data type.
    - *Without* COBOL 85 standard, the value range of the COBOL data type `BINARY` depends on the binary field size, thus matches the IDL data type exactly. In this case, there are no restriction regarding value ranges.
    - To match the value range of IDL type I2 and I4 exactly, depending on the operating system, the following compiler directive (option) is generated into the client interface objects and server skeletons:
      - **z/OS and z/VSE**
        the IBM compiler option `TRUNC(BIN)`
      - **Other Operating Systems or Compilers**
        refer to your COBOL compiler documentation to see whether compiler directives or options exist.
11. COBOL does not have a corresponding data type for a compatible `B/BV` mapping. Thus the mapping is to COBOL `PIC X` data type. EntireX RPC transports the (binary) data as it is: no character conversion will be performed.
12. Supported for operating systems z/OS, BS2000, z/VSE and IBM i only.
13. With variable length fields with maximum (AV*n*, BV*n*, KV*n* and UV*n*), connecting COBOL to endpoints with a concept of real string types - such as Java, .NET, C, XML, Web services etc. - is straightforward. The transfer of data in the RPC data stream depends on the actual length of the data and not the field size, as seen in COBOL. For the COBOL side, the actual content length of such fields is determined using a trim mechanism. For AV*n*, all trailing `SPACE`s are ignored

before send. After receive, the content is padded with trailing SPACEs up to the COBOL field size. For BV*n*, HEX ZERO is used instead of SPACE; for UV*n*, Unicode code point U+0020. See also the notes under *IDL Data Types* in the IDL Editor documentation. If your application relies on trailing SPACEs, HEX ZEROs or Unicode code points U+0020, you cannot use a mapping to variable length fields with maximum (AV*n*, BV*n*, KV*n* and UV*n*); Use a mapping to fixed length types instead: A*n*, B*n*, K*n* and U*n*.

Mapping Library Name and Alias

Client Side

The IDL library name as specified in the IDL file (there is no 8-character limitation) is sent from a client to the server. Special characters are not replaced. The library alias is neither sent to the server nor used for other purposes on the COBOL client side.

Server Side

If you are using a so-called server mapping file, the target COBOL server program is located with the help of this file. A server mapping file is a Designer file with extension .cvm. See *Server Mapping Files for COBOL* in the Designer documentation. See also *Locating and Calling the Target Server* under z/OS (CICS, Batch, IMS) | BS2000 in the respective Administration or RPC Server documentation.

If you are *not* using a server mapping file, the IDL library name as specified in the IDL file is ignored.

Mapping Program Name and Alias

Client Side

The IDL program name as specified in the IDL file (there is no 8-character limitation) is sent from a client to the server. Special characters are not replaced. The program alias is not sent to the server, but during wrapping it is used to derive the suggestion for the source file names of the client interface objects (COBOL subprograms, copybooks) instead of using the IDL program names, see ***Customize Automatically Generated Client Names***.

Server Side

If you are using a so-called server mapping file, the target COBOL server program is located with the help of this file. A server mapping file is a Designer file with extension .cvm. See *Server Mapping Files for COBOL* in the Designer documentation. This provides the following advantages:

- IDL program names are not limited to 8 characters and do not have to match the target COBOL server program names.
- Target COBOL server program names (COBOL subprograms) can be customized during wrapping. See ***Customize Automatically Generated Server Names***.

If you are *not* using a server mapping file, the target COBOL server program must match the IDL program name. In this case:

- The length of the IDL program names is limited by your COBOL system (often 8 characters).
- The set of allowed characters for IDL program names is restricted by your COBOL system and the underlying file system.

It is your responsibility as application developer to ensure that these requirements are met. See *Locating and Calling the Target Server* under z/OS (CICS, Batch, IMS) | BS2000 in the respective Administration or RPC Server documentation.

Mapping Parameter Names

The parameter names, as given in the `parameter-data-definition` under *Software AG IDL Grammar* in the IDL Editor documentation of the IDL file, are mapped to fields within the `LINKAGE` section of the generated COBOL client interface objects and COBOL server skeletons.

When building fields within the `LINKAGE` section, the special characters '#', '$', '&', '+', '-', '.', '/', '@' and '_', allowed within names of parameters, are mapped to the character hyphen '-' valid for COBOL names. Example:

`HU$GO` results in `HU-GO`

Trailing and preceding special characters are also removed. Example:

`#HUGO$` results in `HUGO`

Subsequent special characters are replaced by one hyphen. Example:

`HU$#$GO` results in `HU-GO`

If the parameter name starts with a digit, e.g. '`1`', it is prefixed with the character '`P`'. Example:

`1HUGO` results in `P1HUGO`

Mapping Fixed and Unbounded Arrays

Client and Server Side

- **Fixed Arrays**
  Fixed arrays within the IDL file are mapped to fixed COBOL tables. See the `array-definition` under *Software AG IDL Grammar* in the IDL Editor documentation for the syntax on how to describe fixed arrays within the IDL file and refer to `fixed-bound-array-index`.
- **IDL Unbounded Arrays with Maximum**
  IDL unbounded arrays with maximum are mapped to COBOL tables with the `DEPENDING ON` clause. See *COBOL Tables with Variable Size - `DEPENDING ON` Clause* under *COBOL to IDL Mapping* in the IDL Extractor for COBOL documentation. Note the following:
  - The `from-value` of the `DEPENDING ON` clause is always 1.
  - ODO objects for justification of the number of occurrences are generated into the client interface objects and server skeletons.
  - When a 2/3 dimensional unbounded array is received from a partner, all vectors of the second dimension must have the same length, i.e. the array forms a rectangle. The same applies to the third dimension (all vectors must have the same length), the array forms a cuboid. If these rules are violated, unexpected behavior occurs. For illustration, see picture under `array-definition` under *Software AG IDL Grammar* in the IDL Editor documentation.
  - Sending a 2/3 dimensional unbounded array to a partner violating the rule above is not possible: COBOL does not allow you to set vector lengths differently.
  - A maximum upper bound given with the IDL unbounded array defines the maximum COBOL table size. The COBOL table can vary from 1 to this maximum. See `array-definition` under *Software AG IDL Grammar* in the IDL Editor documentation for the syntax on how to describe unbounded arrays within the IDL file and refer to `unbound-array-index`.
  - Depending on your target COBOL compiler, 2- and 3-dimensional unbounded arrays may not be supported (e.g. BS2000).
  - The Designer generates the COBOL interface objects and server (skeletons) without considering restrictions of the target COBOL compiler. See your COBOL compiler documentation for possible workarounds, for example using compiler switches or compiler options.

Client Side

- **IDL Unbounded Arrays without Maximum**
  IDL unbounded arrays without a maximum are mapped to COBOL tables with the `UNBOUNDED` keyword. There is no upper bound limitation also on the COBOL side. They are restricted to clients on z/OS with standard linkage calling convention (CICS | Batch | IMS) to its client interface object. See ***Using IDL Unbounded Groups or Arrays without Maximum*** for more information.

Server Side

- **IDL Unbounded Arrays without Maximum**
  IDL unbounded arrays without a maximum are not supported by COBOL Wrapper server generation and EntireX RPC servers under z/OS (CICS, Batch, IMS) | BS2000 | CICS ECI | IMS Connect | CICS Socket Listener.

Mapping Groups and Periodic Groups

Client and Server Side

- Groups within the IDL file are mapped to COBOL structures using level numbers. See the `group-parameter-definition` under *Software AG IDL Grammar* in the IDL Editor documentation for the syntax on how to describe groups within the IDL file.
- For groups with an array definition (including fixed, maximum upper bound or without upper bound) the same applies as for arrays, see ***Mapping Fixed and Unbounded Arrays***. Additionally note the following:
  - If unbounded groups are nested, and depending on your target COBOL compiler, they may not be supported (e.g. BS2000).
  - There is a restriction on the number of indices. Most COBOL compilers support a maximum of 7 indices.

    The Designer generates the COBOL interface objects and server (skeletons) without considering restrictions of the target COBOL compiler. See your COBOL compiler documentation for possibilities to work round the restrictions, for example using compiler switches or compiler options.

Mapping Structures

Client and Server Side

Structures within the IDL file are dissolved at the location where they are used. They are mapped to COBOL structures like groups. See the `structure-definition` under *Software AG IDL Grammar* in the IDL Editor documentation for the syntax on how to describe structures within the IDL file.

Mapping the Direction Attributes In, Out, InOut

The IDL syntax allows you to define parameters as In parameters, Out parameters, or InOut parameters (which is the default if nothing is specified). See the `attribute-list` under *Software AG IDL Grammar* in the IDL Editor documentation for the syntax on how to describe attributes within the IDL file and refer to `direction-attribute`.

Client Side

This direction specification is reflected in the generated COBOL interface object as follows:

- Direction attributes do not change the COBOL 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* in the IDL Editor documentation for the syntax on how to describe attributes within the IDL file and refer to `direction-attribute`.

Server Side

The direction attribute sent from any RPC client, for example EntireX Adapter, Java, DCOM, C, COBOL, .NET, XML and PL/I is considered. It is taken from the IDL file and any (optional) server mapping file (.cvm) to the related IDL file.

Mapping the ALIGNED Attribute

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

Client and Server Side

This attribute corresponds to the `SYNCHRONIZED` clause. If it is specified, data will be mapped according to the following rules:

Software AG IDL	COBOL Data Type	Alignment	Notes
F4	USAGE COMP-1 SYNC	+4	1
F8	USAGE COMP-2 SYNC	+8	1
I2	PIC S9(4) BINARY SYNC	+2	1
I4	PIC S9(8) BINARY SYNC	+4	1

Notes:

1. On IBM i, specify the compiler option *SYNC in the commands CRTCBLMOD or CRTBNDCBL for the usage of the SYNCHRONIZED clause.

Calling Servers as Procedures or Functions

Client and Server Side

The COBOL 85 standard does not support a concept of functions like the programming languages C or PL/I. Any Software AG IDL program definition is mapped to a COBOL program. See *Mapping Program Name and Alias*.

III Writing Applications with the COBOL Wrapper

- *Writing Standard Call Interface Clients*
- *Writing `EXEC CICS LINK` Clients*
- *Using the Generated Copybooks*
- *Using Broker Logon and Logoff*
- *Using Conversational RPC*
- *Using IDL Unbounded Groups or Arrays without Maximum*
- *Using RPC Authentication (Natural Security, Impersonation, Integration Server)*
- *Using the COBOL Wrapper with Non-secure Natural RPC Server*
- *Using SSL/TLS*
- *Using Internationalization with the COBOL Wrapper*

See also *Scenarios and Programmer Information* under z/OS (CICS, Batch, IMS) | CICS ECI.

8 Writing Standard Call Interface Clients

This chapter describes in five steps how to write your first COBOL RPC client program. It uses the standard call interface: ***CICS*** | ***Batch*** | ***IMS***.

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.

Step 1: Declare and Initialize the RPC Communication Area

The ***RPC Communication Area*** is your interface (API) to RPC communication and the generic RPC service module `COBSRVI`.

How to declare the communication area in your RPC client program depends on the generation option `External Clause`, `Linkage Section` or `Copybook` (see ***RPC Communication Area*** under *Generation Settings - Properties*) and whether only copybook `ERXCOMM` is used, or both copybooks `ERXCOMM` and `ERXVSTR` are used.
The optional `ERXVSTR` copybook is an extension to the `ERXCOMM` copybook. It enables an RPC client to specify long data strings (e.g. passwords). For usage see ***`ERXVSTR` Copybook*** under *Using the Generated Copybooks*.

See the following code snippets:

- Only Copybook ERXCOMM is Used
- Both Copybooks ERXCOMM and ERXVSTR are Used

Only Copybook ERXCOMM is Used

- **For External Clause Option**

```
* Declare RPC communication area
  01 ERX-COMMUNICATION-AREA EXTERNAL.
     COPY ERXCOMM.

* Initialize RPC communication area (see Note 1)
  INITIALIZE ERX-COMMUNICATION-AREA.

* Set version (see Note 2)
  MOVE "2000" to COMM-VERSION.
```

- **For Linkage Section option**

```
* Declare RPC communication area
  01 ERX-COMMUNICATION-AREA.
     COPY ERXCOMM.

* Initialize RPC communication area (see Note 1)
  INITIALIZE ERX-COMMUNICATION-AREA.

* Set version (see Note 2)
  MOVE "2000" to COMM-VERSION.
```

Both Copybooks ERXCOMM and ERXVSTR are Used

- **For External Clause option**

```
* Declare RPC communication area
  01 ERX-COMMUNICATION-AREA EXTERNAL.
     COPY ERXCOMM.
  01 ERX-COMMUNICATION-VSTR EXTERNAL.
     COPY ERXVSTR.

* Initialize RPC communication area (see Note 1)
  INITIALIZE ERX-COMMUNICATION-AREA.
  INITIALIZE ERX-COMMUNICATION-VSTR.

* Set version (see Note 2)
  MOVE "4000" to COMM-VERSION.
```

- **For Linkage Section option**

```
* Declare RPC communication area
  01 ERX-COMMUNICATION-AREA.
     COPY ERXCOMM.
  01 ERX-COMMUNICATION-VSTR.
     COPY ERXVSTR.

* Initialize RPC communication area (see Note 1)
  INITIALIZE ERX-COMMUNICATION-AREA.
  INITIALIZE ERX-COMMUNICATION-VSTR.

* Set version (see Note 2)
  MOVE "4000" to COMM-VERSION.
```

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

Notes:

1. The RPC communication area copybook ERXCOMM and - if used - its extension copybook ERXVSTR must be correctly initialized with the data formats. Do not move SPACES to them! Use, for example, a COBOL INITIALIZE statement.
2. If the copybook ERXCOMM only is used, COMM-VERSION is set to "2000". If both copybooks are used (ERXCOMM and its extension ERXVSTR), COMM-VERSION is set to "4000".

Step 2: Declare the IDL Data Structures for Client Interface Objects

For every program definition of the IDL file, the COBOL Wrapper generates an ***IDL interface copybook*** with the description of the customer's interface data as a COBOL structure. For ease of use you can include these structures into your RPC client program as shown below.

```
* Declare customer data to generated RPC Stubs
   01 CALC-AREA.
      COPY CALC.
```

However, as an alternative, you can use your own customer data structures. In this case the COBOL data types and structures must match the interfaces of the generated client interface objects, otherwise unpredictable results may occur.

```
* Declare customer data to generated RPC Stubs
 01 CALC-AREA.
    10 PARAMETER.
       15 OPERATOR                  PIC X.
       15 OPERAND1                  PIC S9(9) BINARY.
       15 OPERAND2                  PIC S9(9) BINARY.
       15 RESULT                    PIC S9(9) BINARY.
```

Step 3: Required Settings in the RPC Communication Area

The following settings to the RPC communication area are required as a minimum to use the COBOL Wrapper. These settings have to be applied in your RPC client program. It is not possible to generate any defaults into the client interface objects.

```
* assign the broker to talk with ...
 MOVE "localhost:1971" to COMM-ETB-BROKER-ID.
* assign the server to talk with ...
 MOVE "RPC"            to COMM-ETB-SERVER-CLASS.
 MOVE "SRV1"           to COMM-ETB-SERVER-NAME.
 MOVE "CALLNAT"        to COMM-ETB-SERVICE-NAME.
* assign the user id to the broker ...
 MOVE "ERXUSER"        to COMM-USERID.
```

Step 4: Optional Settings in the RPC Communication Area

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

```
MOVE "EXAMPLE"        to COMM-LIBRARY.
MOVE "00000300"       to COMM-ETB-WAIT.
MOVE "PASSWORD"       to COMM-PASSWORD.  (Note 1)
```

Notes:

1. For *Implicit Logon*, if required in your environment, the client password can be given here. It is provided then through the client interface object. If you have to issue an *Explicit Logon*, see ***Using Broker Logon and Logoff***.

Step 5: Issue the RPC Request and Check for Success

How to issue the request in your RPC client program depends on the generation option `External Clause`, `Linkage Section` or `Copybook` (see ***RPC Communication Area***) and usage of the copybooks `ERXCOMM` and `ERXVSTR`. See following code snippets:

- **External Clause option**

```
   CALL "CALC" USING  OPERATOR OPERAND1 OPERAND2 RESULT
       ON EXCEPTION
*         Perform error-handling
       NOT ON EXCEPTION
          IF COMM-RETURN-CODE = ZERO
*            Perform success-handling
          ELSE
*            Perform error-handling (See Note 1)
          END-IF
   END-CALL.
```

- **Linkage Section option; copybook ERXCOMM is used only**

```
   CALL "CALC" USING  OPERATOR OPERAND1 OPERAND2 RESULT
                      ERX-COMMUNICATION-AREA
       ON EXCEPTION
*         Perform error-handling
       NOT ON EXCEPTION
          IF COMM-RETURN-CODE = ZERO
*            Perform success-handling
          ELSE
*            Perform error-handling (See Note 1)
          END-IF
   END-CALL.
```

- **Linkage Section option; both copybooks ERXCOMM and ERXVSTR are used**

```
   CALL "CALC" USING  OPERATOR OPERAND1 OPERAND2 RESULT
                      ERX-COMMUNICATION-AREA
                      ERX-COMMUNICATION-VSTR
       ON EXCEPTION
*      Perform error-handling
       NOT ON EXCEPTION
          IF COMM-RETURN-CODE = ZERO
*            Perform success-handling
          ELSE
*            Perform error-handling (See Note 1)
          END-IF
   END-CALL.
```

- **Copybook option**

```
   CALL "CALC" USING  OPERATOR OPERAND1 OPERAND2 RESULT
       ON EXCEPTION
*      Perform error-handling
       NOT ON EXCEPTION
          IF RETURN-CODE = ZERO
*            Perform success-handling
          ELSE
*            Perform error-handling (See Note 2)
          END-IF
   END-CALL.
```

Notes:

1. The field `COMM-RETURN-CODE` in the RPC communication area contains the error provided by the COBOL Wrapper. For the error messages returned, see *Error Messages and Codes*.
2. 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 `COBEXIT` copybook (see folder *include*).

9 Writing EXEC CICS LINK Clients

This chapter describes in five steps how to write your first COBOL RPC client program for an EXEC CICS LINK interface. You can also write an RPC client for CICS with a standard call interface, see *Using the COBOL Wrapper for CICS with DFHCOMMAREA Calling Convention (z/OS and z/VSE)*.

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.

Step 1: Declare IDL Structures and RPC Communication Area

For every program definition of the IDL file, the COBOL Wrapper generates an *IDL interface copybook* with the description of the customer's interface data as a COBOL structure. For ease of use you can include these structures together with the RPC communication area copybook ERXCOMM into your RPC client program. The RPC communication area is your interface (API) to RPC communication and the *Generation and Usage of Generic RPC Service Module COBSRVI*.

Definition is physically in a specific order (see code snippet below):

- A parent label on COBOL level 01 (here label CALC-AREA) is followed by the IDL interface copybook (here copybook CALC).
- Then comes a COBOL label with the RPC communication area below (here label RPC-COMMUNICATION-AREA together with copybook ERXCOMM).

Label names could be different in your application, but the physical sequence of labels and copybooks are important. See following code snippets:

See following code snippets:

```
* Declare customer data to generated client interface objects
 01 CALC-AREA.
   COPY CALC
* Declare RPC communication area
   03 ERX-COMMUNICATION-AREA.
      COPY ERXCOMM.
```

However, as an alternative, you can use your own customer data structures. In this case the COBOL data types and structures must match the interfaces of the generated client interface objects, otherwise unpredictable results may occur.

```
* Declare customer data to generated client interface objects
 01 CALC-AREA.
   03 OPERATOR                   PIC X.
   03 OPERAND1                   PIC S9(8) COMP.
   03 OPERAND2                   PIC S9(8) COMP.
   03 RESULT                     PIC S9(8) COMP.
* Declare RPC communication area
   03 ERX-COMMUNICATION-AREA.
      COPY ERXCOMM.
```

Step 2: Initialize the RPC Communication Area

```
  . . .

* Call subprogram to initialize the RPC Communication Area (see Note 1)
  CALL "INIT-RPC" USING ERX-COMMUNICATION-AREA.
* Set version (see Note 2)
  MOVE "2000" to COMM-VERSION.
  . . .

* Subprogram to initialize the RPC communication area
  IDENTIFICATION DIVISION.
  PROGRAM-ID. INIT-RPC.
  DATA DIVISION.
  LINKAGE SECTION.
   01 RPC-COMMUNICATION-AREA.
      COPY ERXCOMM.
  PROCEDURE DIVISION USING RPC-COMMUNICATION-AREA.
  MAIN SECTION.
* Initialize the RPC Communication Area (see Note 3)
      INITIALIZE RPC-COMMUNICATION-AREA.
      EXIT PROGRAM.
  END PROGRAM INIT-RPC.
```

Notes:

1. If your generated ***IDL interface copybook*** contains a COBOL table with an `OCCURS DEPENDING ON` clause, originating from an IDL unbounded array, it is important to set the ODO object to the required value for upper-bound before you call the initialization subprogram. (Refer to *Fixed and Unbounded Arrays* in the IDL Editor documentation.) See the following code snippet:

```
. . .
01 IDL-AREA.
   03 IDL-FIELD1                PIC X(8).
   03 IDL-FIELD2                PIC X(32).
   03 . . .
   03 ODO-OBJECT                PIC 9(8) BINARY.
   03 ODO-SUBJECT OCCURS 1 TO 24 DEPENDING ON ODO-OBJECT.
      04 ODO-FIELD1             PIC X(5).
      04 ODO-FIELD1             PIC X(1).
      04 . . .
    03 ERX-COMMUNICATION-AREA.
       COPY ERXCOMM.
. . .
* Set the ODO object to required value for input
      MOVE <upper-bound> TO ODO-OBJECT.
      MOVE . . .
* Initialize RPC communication area
      CALL "INIT-RPC" USING ERX-COMMUNICATION-AREA.
. . .
```

2. Because only copybook `ERXCOMM` is used, `COMM-VERSION` is set to "2000".
3. The RPC communication area copybook `ERXCOMM` must be correctly initialized with the data formats. Do not move `SPACES` to it! Use, for example, a `COBOL INITIALIZE` statement.

Step 3: Required Settings in the RPC Communication Area

The following settings to the RPC communication area are required as a minimum to use the COBOL Wrapper. These settings have to be applied in your RPC client program. It is not possible to generate any defaults into the client interface objects.

```
* assign the broker to talk with ...
 MOVE "localhost:1971" to COMM-ETB-BROKER-ID.
* assign the server to talk with ...
 MOVE "RPC"            to COMM-ETB-SERVER-CLASS.
 MOVE "SRV1"           to COMM-ETB-SERVER-NAME.
 MOVE "CALLNAT"        to COMM-ETB-SERVICE-NAME.
* assign the user id to the broker ...
 MOVE "ERXUSER"        to COMM-USERID.
```

Step 4: Optional Settings in the RPC Communication Area

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

```
MOVE "EXAMPLE"        to COMM-LIBRARY.
MOVE "00000300"       to COMM-ETB-WAIT.
MOVE "PASSWORD"       to COMM-PASSWORD.  (Note 1)
```

Notes:

1. For *Implicit Logon*, if required in your environment, the client password can be given here. It is provided then through the client interface object. If you have to issue an *Explicit Logon*, see ***Using Broker Logon and Logoff***.

Step 5: Issue the RPC Request and Check for Success

See following code snippets:

```
 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 (See Note 1)
    END-IF
 ELSE
*    Perform error-handling
 END-IF.
```

Notes:

1. The field `COMM-RETURN-CODE` in the RPC communication area contains the error provided by the COBOL Wrapper. For the error messages returned, see *Error Messages and Codes*.

10 Using the Generated Copybooks

This chapter explains how clients built with the COBOL Wrapper use the generated copybooks.

IDL Interface Copybooks

The IDL interface copybooks (see folder *include*) are the API of the COBOL client application using client interface objects. You can customize the ***Starting COBOL Level for Data Items in Generated Copybooks*** according to your needs.

If IDL unbounded groups without maximum (/V) or arrays without maximum (for example (A100/V)) are contained in the IDL, these are mapped with keyword `UNBOUNDED`. In this case:

- cut and paste the top-level COBOL groups where
  - anywhere deeper the keyword `UNBOUNDED` is contained into the `LINKAGE SECTION`
  - no keyword `UNBOUNDED` is contained into the `WORKING STORAGE SECTION`

  For details see ***Using IDL Unbounded Groups or Arrays without Maximum***.

For all other IDLs, a starting level greater than one allows you to

- embed (include) the generated copybook into other existing COBOL structures:

```
1 MYGROUP.
  10 . . .
  10 . . .
  10 MYIDL.
  COPY MYIDL.
```

- specify usage clauses such as `EXTERNAL`, `GLOBAL` etc.:

```
1 MYIDL1 GLOBAL.
  COPY MYIDL1.
```

- use multiple generated copybooks with duplicate parameter names on IDL level 1 in the same COBOL program:

```
1 MYIDL1.
  COPY MYIDL1.
1 MYIDL2.
  COPY MYIDL2.
```

More information:

- For IDL unbounded groups or arrays without maximum, see the `array-definition` under *Software AG IDL Grammar* in the IDL Editor documentation for the syntax on how to describe

unbounded arrays within the IDL file and refer to `unbound-array-index`. For COBOL mappings, see *Mapping Fixed and Unbounded Arrays* and *Mapping Groups and Periodic Groups*.

- For writing a standard call interface client according to scenario *CICS* | *Batch* | *IMS*, see *Writing Standard Call Interface Clients*.
- For writing a client according to scenario *Using the COBOL Wrapper for CICS with `DFHCOMMAREA` Calling Convention (z/OS and z/VSE)*, see *Writing `EXEC CICS LINK` Clients*.

ERXCOMM Copybook

The `ERXCOMM` copybook (see folder *include*) holds RPC metadata for RPC clients. Here you provide parameters that are needed to communicate with the broker and are not specific to client interface objects. Upon return from an RPC request it provides, for example, the error code. In this way it defines a context for RPC clients.

For usage, refer to the following sections:

- *Copybook `ERXCOMM`* under *The RPC Communication Area (Reference)*
- *Step 1: Declare and Initialize the RPC Communication Area* in section *Writing Standard Call Interface Clients*

ERXVSTR Copybook

The `ERXVSTR` copybook (see folder *include*) is an extension to the `ERXCOMM` copybook for RPC clients. It enables an RPC client to specify long broker passwords, and long user IDs/passwords for RPC authorization. Its usage is optional.

The RPC communication area extension copybook `ERXVSTR` is generated for *Target Operating System* z/OS and RPC clients using a call interface to its client interface object, meaning one of the following *Client Interface Types* is selected:

- *CICS with Standard Linkage Calling Convention*
- *Batch with Standard Linkage Calling Convention*
- *IMS BMP with Standard Linkage Calling Convention*
- *IMS MPP with Standard Linkage Calling Convention*

For usage, refer to the following sections:

- *Using Broker Logon and Logoff*
- *Using RPC Authentication (Natural Security, Impersonation, Integration Server)*

- ***Copybook ERXVSTR*** under *The RPC Communication Area (Reference)*
- ***Step 1: Declare and Initialize the RPC Communication Area*** in section *Writing Standard Call Interface Clients*

COBINIT Copybook

The `COBINIT` copybook (see folder *include*) is generated if option `Copybook` for ***RPC Communication Area*** is selected. Its purpose is to set communication parameters such as `COMM-ETB-BROKER-ID`, `COMM-ETB-SERVER-NAME` etc. into the RPC Communication Area. You can set parameters made available through ***ERXCOMM Copybook*** and also ***ERXVSTR Copybook***.

COBEXIT Copybook

The `COBEXIT` copybook (see folder *include*) is generated if option `Copybook` for ***RPC Communication Area*** is selected. Its purpose is to check and map error codes. COBOL statements that have been commented out are generated into the copybook as an example.

11 Using Broker Logon and Logoff

Introduction

This section explains how clients built with the COBOL Wrapper use explicit broker logon and logoff functions. The logon call is the first call to the broker, before any RPC call. The `COMM-ETB-USER-ID` field (and the `COMM-ETB-TOKEN` field, where provided) must not change from logon, through all calls of client interface objects, until final logoff. The logoff call should be issued as soon as RPC communication is no longer needed. This is similar to *`LOGON` and `LOGOFF`*, *`USER-ID` and `TOKEN`* and *Authentication* under *Writing Applications using EntireX Security* in the ACI Programming documentation.

To use explicit broker logon and logoff you need the following components:

- the ***Delivered Modules*** are provided to log on to and log off from the broker
- the copybook `ERXCOMM` if a 32-byte broker password is sufficient; see RPC communication area ***Copybook `ERXCOMM`***
- the copybook `ERXVSTR` to use a long broker password; see RPC communication area ***Copybook `ERXVSTR`***
- We strongly recommend using **SSL/TLS** if you send an authentication as described here with the COBOL Wrapper to a secure partner. See also *SSL/TLS Parameters for SSL Clients* under *SSL/TLS, HTTP(S), and Certificates with EntireX* in the platform-independent Administration documentation.

Logging on Using Short Broker Passwords (all Interface Types)

This approach allows a maximum of 32 bytes for the broker password. The code you write depends on the interface type:

- Call Interface
- EXEC CICS LINK Interface

See ***Client Interface Types***.

Call Interface

This interface type applies to the scenarios ***CICS*** | ***Batch*** | ***IMS***.

To log on to the Broker with a short password

1 Declare and initialize the RPC communication area as described under ***Only Copybook `ERXCOMM` is Used*** under ***Step 1: Declare and Initialize the RPC Communication Area*** in section *Writing Standard Call Interface Clients*.

2 Log on to the broker with the logon function `LO` provided by the generic RPC services module, using the Call Interface.

```
* Set function broker logon
 MOVE "LO"   TO COMM-FUNCTION.
* Set broker user ID in RPC Communication Area
 MOVE "COB-USER" TO COMM-ETB-USER-ID.

* Optionally set broker password/kernelsecurity to use EntireX Security
 MOVE "COB-PASS" TO COMM-ETB-PASSWORD.
 MOVE "Y"        TO COMM-KERNEL-SECURITY.

* Call generic RPC service module to call broker (see Note 1)
 CALL "COBSRVI" USING ERX-COMMUNICATION-AREA
 ON EXCEPTION
*   Perform error-handling
 NOT ON EXCEPTION
    IF (COMM-RETURN-CODE = 0) THEN
*       Perform success-handling
    ELSE
*       Perform error-handling (see Note 2)
    END-IF
 END-CALL.
* begin of application logic
 ...
```

3 Issue your RPC requests as usual, without using explicit logon and logoff.

To log off from the Broker with a short password

- Log off from the broker with the log off function `LF` provided by the generic RPC services module, using a `CALL` statement.

EXEC CICS LINK Interface

This interface type applies to the scenario ***Using the COBOL Wrapper for CICS with*** *DFHCOMMAREA* ***Calling Convention (z/OS and z/VSE)***.

To log on to the Broker with a short password

1 Declare the RPC communication area as described under ***Step 1: Declare IDL Structures and RPC Communication Area*** in section *Writing* *EXEC CICS LINK* *Clients*.

2 Initialize the RPC communication area as described under ***Step 2: Initialize the RPC Communication Area*** under *Writing* *EXEC CICS LINK* *Clients*.

3 Log on to the broker with the logon function `LO` provided by the generic RPC services module, using `EXEC CICS LINK`.

```
  . . .
 MOVE "LO"   TO COMM-FUNCTION.
* Set broker user ID in RPC Communication Area
 MOVE "COB-USER" TO COMM-ETB-USER-ID.

* Optionally set broker password/kernelsecurity to use EntireX Security
 MOVE "COB-PASS" TO COMM-ETB-PASSWORD.
 MOVE "Y"        TO COMM-KERNEL-SECURITY.

* Call generic RPC service module to call broker
 EXEC CICS LINK PROGRAM  ("COBSRVI")
                RESP     (CICS-RESP1)
                RESP2    (CICS-RESP2)
                COMMAREA (ERX-COMMUNICATION-AREA)
                LENGTH   (LENGTH OF ERX-COMMUNICATION-AREA)
 END-EXEC.
 IF WORKRESP = DFHRESP(NORMAL)
    IF (COMM-RETURN-CODE = 0) THEN
*       Perform success-handling
    ELSE
*       Perform error-handling (see Note 2)
    END-IF
 ELSE
*    Perform error-handling
 END-IF.
* begin of application logic
...
```

4 Issue your RPC requests as usual, without using explicit logon and logoff.

To log off from the Broker with a short password

- Log off from the broker with the log off function `LF` provided by the generic RPC services module, using `EXEC CICS LINK`.

Notes:

1. If you are only using copybook `ERXCOMM` only, pass only the address of `ERXCOMM` to the generic RPC service module.
2. The field `COMM-RETURN-CODE` in the RPC communication area contains the error provided by the COBOL Wrapper. For the error messages returned, see *Error Messages and Codes*.

Logging on Using Long Broker Passwords (z/OS with Call Interface)

This section applies to the scenarios ***CICS***, ***Batch*** and ***IMS*** with the `CALL` interface.

With this approach you can use long passwords. It requires the `ERXVSTR` copybook.

The RPC communication area extension copybook `ERXVSTR` is generated for ***Target Operating System*** z/OS and RPC clients using a call interface to its client interface object, meaning one of the following ***Client Interface Types*** is selected:

- ***CICS with Standard Linkage Calling Convention***
- ***Batch with Standard Linkage Calling Convention***
- ***IMS BMP with Standard Linkage Calling Convention***
- ***IMS MPP with Standard Linkage Calling Convention***

To log on to the Broker with a long password

1 Declare and initialize the RPC communication area as described under ***Both Copybooks `ERXCOMM` and `ERXVSTR` are Used*** under ***Step 1: Declare and Initialize the RPC Communication Area*** in section *Writing Standard Call Interface Clients*.

2 Log on to the broker with the logon function `LO` provided by the generic RPC services module, using the Call Interface.

```
* Set function broker logon
 MOVE "LO"   TO COMM-FUNCTION.
* Set broker user ID/kernelsecurity in RPC Communication Area
 MOVE "COB-USER" TO COMM-ETB-USER-ID.
 MOVE "Y"        TO COMM-KERNEL-SECURITY.
* set long broker password in RPC Variable String Area
 INSPECT ETBPWD TALLYING STR-LENGTH FOR CHARACTERS BEFORE SPACE.
 MOVE 1 TO STR-OFFSET.
 MOVE STR-OFFSET TO COMM-ETB-PASSWORD-OFFSET.
 MOVE STR-LENGTH TO COMM-ETB-PASSWORD-LENGTH.
 STRING ETBPWD DELIMITED BY SPACE INTO
          COMM-STRING-AREA WITH POINTER STR-OFFSET.
* Call generic RPC service module to call broker
  CALL "COBSRVI" USING ERX-COMMUNICATION-AREA
                       ERX-COMMUNICATION-VSTR.
 ON EXCEPTION
*       Perform error-handling
 NOT ON EXCEPTION
     IF (COMM-RETURN-CODE = 0) THEN
*       Perform success-handling
    ELSE
*       Perform error-handling (see Note 2)
    END-IF
 END-CALL.
  . . .
```

To log off from the Broker with a long password

- See the following code snippet:

```
  . . .
* Set function broker logoff
 MOVE "LF"   TO COMM-FUNCTION.
* Set broker user ID in RPC Communication Area
 MOVE "COB-USER" TO COMM-ETB-USER-ID.
* Call generic RPC service module to call broker  (see Note 1)
 CALL "COBSRVI" USING ERX-COMMUNICATION-AREA
                      ERX-COMMUNICATION-VSTR.
 ON EXCEPTION
*   Perform error-handling
 NOT ON EXCEPTION
    IF (COMM-RETURN-CODE = 0) THEN
*      Perform success-handling
    ELSE
*      Perform error-handling (see Note 2)
    END-IF
 END-CALL.
  . . .
```

Notes:

1. If both copybooks are used, you need to pass both addresses, first the address of `ERXCOMM`, then the address of `ERXVSTR` to the generic RPC service module.
2. The field `COMM-RETURN-CODE` in the RPC communication area contains the error provided by the COBOL Wrapper. For the error messages returned, see *Error Messages and Codes*.

12 Using Conversational RPC

This chapter explains how clients built with the COBOL Wrapper use conversational RPC.

RPC conversations are supported when communicating with an RPC server. It is further assumed that you are familiar with the concepts of conversational RPC and non-conversational RPC. To use conversational RPC, you need the following components:

- the *Delivered Modules* are provided to open, close or abort conversations
- the *RPC Communication Area*

The code you write depends on the interface type:

See *Client Interface Types*.

Call Interface

This interface type applies to the scenarios *CICS* | *Batch* | *IMS*.

To use conversational RPC

1 Declare and initialize the RPC communication area with one of the approaches described under *Step 1: Declare and Initialize the RPC Communication Area* in section *Writing Standard Call Interface Clients*. Here you can use **copybook** `ERXCOMM` **only**, or **both copybooks** `ERXCOMM` **and** `ERXVSTR`.

2 Open a conversation with the function Open Conversation `OC` provided by the generic RPC services module. The code snippet below illustrates the variant where only copybook `ERXCOMM` is used. If you are using both copybooks `ERXCOMM` and `ERXVSTR`, see Note 1.

```
 MOVE "OC"   TO COMM-FUNCTION.
* Call generic RPC service module to use conversational mode (Note 1)
 CALL "COBSRVI" USING ERX-COMMUNICATION-AREA
 ON EXCEPTION
*       Perform error-handling
 NOT ON EXCEPTION
    IF (COMM-RETURN-CODE = 0) THEN
*       Perform success-handling
    ELSE
*       Perform error-handling (See Note 2)
    END-IF
 END-CALL.
```

3 Issue your RPC requests as within non-conversational mode using the generated client interface objects. Different client interface objects can participate in the same RPC conversation.

To abort conversational RPC communication

- Abort an unsuccessful RPC conversation with the function Close Conversation `CB` provided by the generic RPC services module. The code snippet below illustrates the variant where only copybook `ERXCOMM` is used. If you are using both copybooks `ERXCOMM` and `ERXVSTR`, see Note 1.

```
 MOVE "CB"   TO COMM-FUNCTION.
* Call generic RPC service module to use conversational mode (Note 1)
 CALL "COBSRVI" USING ERX-COMMUNICATION-AREA
 ON EXCEPTION
*       Perform error-handling
 NOT ON EXCEPTION
    IF (COMM-RETURN-CODE = 0) THEN
*       Perform success-handling
    ELSE
*       Perform error-handling (See Note 2)
    END-IF
 END-CALL.
```

To close and commit a conversational RPC communication

- Close the RPC conversation successfully with the function Close Conversation and Commit `CE` provided by the generic RPC services module. The code snippet below illustrates the variant where only copybook `ERXCOMM` is used. If you are using both copybooks `ERXCOMM` and `ERXVSTR`, see Note 1.

```
 MOVE "CE"   TO COMM-FUNCTION.
* Call generic RPC service module to use conversational mode (Note 1)
 CALL "COBSRVI" USING ERX-COMMUNICATION-AREA
 ON EXCEPTION
*       Perform error-handling
 NOT ON EXCEPTION
    IF (COMM-RETURN-CODE = 0) THEN
*       Perform success-handling
    ELSE
*       Perform error-handling (See Note 2)
    END-IF
 END-CALL.
```

Notes:

1. If both copybooks `ERXCOMM` and `ERXVSTR` are used, you need to pass both parameters:

```
CALL "COBSRVI" USING ERX-COMMUNICATION-AREA
                     ERX-COMMUNICATION-VSTR.
```

2. The field `COMM-RETURN-CODE` in the RPC communication area contains the error provided by the COBOL Wrapper. For the error messages returned, see *Error Messages and Codes*.

EXEC CICS LINK Interface

This interface type applies to the scenario ***Using the COBOL Wrapper for CICS with*** `DFHCOMMAREA` ***Calling Convention (z/OS and z/VSE)***.

To use conversational RPC

1 Declare the RPC communication area as described under ***Step 1: Declare IDL Structures and RPC Communication Area*** in section *Writing* `EXEC CICS LINK` *Clients*.

2 Initialize the RPC communication area as described under ***Step 2: Initialize the RPC Communication Area*** under *Writing* `EXEC CICS LINK` *Clients*.

3 Open a conversation with the function Open Conversation `OC` provided by the generic RPC services module:

```
 MOVE "OC"   TO COMM-FUNCTION.
 EXEC CICS LINK PROGRAM  ("COBSRVI")
                RESP     (CICS-RESP1)
                RESP2    (CICS-RESP2)
                COMMAREA (ERX-COMMUNICATION-AREA)
                LENGTH   (LENGTH OF ERX-COMMUNICATION-AREA)
 END-EXEC.
 IF WORKRESP = DFHRESP(NORMAL)
    IF (COMM-RETURN-CODE = 0) THEN
*       Perform success-handling
    ELSE
*       Perform error-handling (See Note 1)
    END-IF
 ELSE
*    Perform error-handling
 END-IF.
```

4 Issue your RPC requests as within non-conversational mode using the generated client interface objects. Different client interface objects can participate in the same RPC conversation.

To abort conversational RPC communication

- Abort an unsuccessful RPC conversation with the function Close Conversation `CB` provided by the generic RPC services module:

```
 MOVE "CB"   TO COMM-FUNCTION.
 EXEC CICS LINK PROGRAM  ("COBSRVI")
                RESP     (CICS-RESP1)
                RESP2    (CICS-RESP2)
                COMMAREA (ERX-COMMUNICATION-AREA)
                LENGTH   (LENGTH OF ERX-COMMUNICATION-AREA)
 END-EXEC.
 IF WORKRESP = DFHRESP(NORMAL)
    IF (COMM-RETURN-CODE = 0) THEN
*       Perform success-handling
    ELSE
*       Perform error-handling (See Note 1)
    END-IF
 ELSE
*    Perform error-handling
 END-IF.
```

To close and commit a conversational RPC communication

- Close the RPC conversation successfully with the function Close Conversation and Commit `CE` provided by the generic RPC services module:

```
 MOVE "CE"   TO COMM-FUNCTION.
 EXEC CICS LINK PROGRAM  ("COBSRVI")
                RESP     (CICS-RESP1)
                RESP2    (CICS-RESP2)
                COMMAREA (ERX-COMMUNICATION-AREA)
                LENGTH   (LENGTH OF ERX-COMMUNICATION-AREA)
 END-EXEC.
 IF WORKRESP = DFHRESP(NORMAL)
    IF (COMM-RETURN-CODE = 0) THEN
*       Perform success-handling
    ELSE
*       Perform error-handling (See Note 1)
    END-IF
 ELSE
*    Perform error-handling
 END-IF.
```

Notes:

1. The field `COMM-RETURN-CODE` in the RPC communication area contains the error provided by the COBOL Wrapper. For the error messages returned, see *Error Messages and Codes*.

13 Using IDL Unbounded Groups or Arrays without Maximum

This chapter explains how clients built with the COBOL Wrapper use IDL unbounded groups or arrays without maximum upper bounds. For illustration of IDL unbounded arrays, see *Example of Arrays with Variable Upper-bounds* under *Software AG IDL Grammar* in the IDL Editor documentation.

Usage of IDL unbounded groups or arrays without maximum is supported for:

- operating system z/OS with IBM Enterprise COBOL compiler for z/OS version 6.1 and above
- RPC clients using a call interface to its client interface object, meaning one of the following ***Client Interface Types*** is selected:
  - ***CICS with Standard Linkage Calling Convention***
  - ***Batch with Standard Linkage Calling Convention***
  - ***IMS BMP with Standard Linkage Calling Convention***
  - ***IMS MPP with Standard Linkage Calling Convention***

See also ***Mapping Fixed and Unbounded Arrays***.

The example below illustrates how IDL unbounded groups without maximum (/V) are used from COBOL. The client interface objects are generated from the IDL as described in *Using the COBOL Wrapper* (CICS with Call Interfaces | Batch | IMS). Storage allocation and pointer usage for unbounded arrays without maximum (for example (A100/V)) are the same as for unbounded groups. Both are mapped to `OCCURS DEPENDING ON` with keyword `UNBOUNDED`.

For writing the RPC client programs, the steps described ***Writing Standard Call Interface Clients*** are valid. Additionally the COBOL group on level 1 containing `OCCURS DEPENDING ON` with keyword `UNBOUNDED` originating from an IDL unbounded group or array is

- defined in the `LINKAGE SECTION`. If no keyword `UNBOUNDED` is contained in the COBOL group on level 1, it is usually defined in the `WORKING STORAGE SECTION`. Compare `(050)` below and ***Step 2: Declare the IDL Data Structures for Client Interface Objects***

- allocated and freed manually, see (070) and (140) below. We strongly recommend using the IBM-specific COBOL ALLOCATE and FREE statements, because the storage is freed and reallocated inside the client interface object using same ALLOCATE and FREE statements
- passed with a pointer to the client interface object, see (110) below.

Sample IDL Program

```
program 'UnboundedTables' is
        define data parameter
        1 UT-TA1             (/V)    In Out
          2 UT-FST           (A12)
          2 UT-TA2           (/V)
            3 UT-ELE         (A05)
          2 UT-LST           (A12)
        end-define
```

Sample COBOL RPC Client and Explanation of Statements

```
(010)   IDENTIFICATION DIVISION.
        PROGRAM-ID. UNBNDCLT.

        DATA DIVISION.
            . . .
          WORKING-STORAGE SECTION.
            . . .
(020)     01 SIZE-IN-BYTES                          PIC 9(4) BINARY.
(030)     01 ITERATION1                             PIC 9(4) BINARY.
          01 ITERATION2                             PIC 9(4) BINARY.
(040)     01 UT-TA1A-PTR                            POINTER.
            . . .

          LINKAGE SECTION.
            . . .
(050)       01 UT-TA1A.
             02 UT-TA1-41                           PIC 9(8) BINARY.
             02 UT-TA2-61                           PIC 9(8) BINARY.
             02 UT-TA1X OCCURS 1 TO UNBOUNDED DEPENDING ON UT-TA1-41.
              03 UT-FST                             PIC X(12).
              03 UT-TA2X OCCURS 1 TO UNBOUNDED DEPENDING ON UT-TA2-61.
               04 UT-ELE                           PIC X(5).
              03 UT-LST                             PIC X(12).
            . . .

        PROCEDURE DIVISION.

            . . .
      *     upper bound is 4 for UT-TA1-41 and 6 for UT-TA1-61
(060)        COMPUTE SIZE-IN-BYTES = LENGTH OF UT-FST * 4
                                   + LENGTH OF UT-ELE * 4 * 6
                                   + LENGTH OF UT-LST * 4.
```

```
(070)        ALLOCATE SIZE-IN-BYTES CHARACTERS INITIALIZED RETURNING UT-TA1A-PTR.
(080)        SET ADDRESS OF UT-TA1A TO UT-TA1A-PTR.
(090)        MOVE 4 TO UT-TA1-41.
             MOVE 6 TO UT-TA1-61.
(100)        MOVE 0 TO ITERATION1.
             PERFORM UT-TA1-41 TIMES
                 ADD 1 TO ITERATION1
                 MOVE ... TO UT-FST(ITERATION1)
                 MOVE 0 TO ITERATION2
                 PERFORM UT-TA2-61 TIMES
                    ADD 1 TO ITERATION2
                    MOVE ... TO UT-ELE(ITERATION1 ITERATION2)
                 END-PERFORM
                 MOVE ... TO UT-LST(ITERATION1)
             END-PERFORM.
(110)        CALL "UNBNDTAB" USING UT-TA1A-PTR ERX-COMMUNICATION-AREA.
(120)        SET ADDRESS OF UT-TA1A TO UT-TA1A-PTR.
(130)        MOVE 0 TO ITERATION1.
             PERFORM UT-TA1-41 TIMES
                 ADD 1 TO ITERATION1
                 MOVE UT-FST(ITERATION1) TO ...
                 MOVE 0 TO ITERATION2
                 PERFORM UT-TA2-61 TIMES
                    ADD 1 TO ITERATION2
                    MOVE UT-ELE(ITERATION1 ITERATION2) TO ...
                 END-PERFORM
                 MOVE UT-LST(ITERATION1) TO ...
             END-PERFORM.
(140)        FREE UT-TA1A-PTR.
            . . .
        END PROGRAM UNBNDCLT.
```

Explanation of Statements

(010) COBOL RPC client UNBNDCLT to demonstrate IDL unbounded groups without maximum.

(020) Variable to hold the result of the storage calculation in bytes for the COBOL structure (050).

(040) POINTER variable to access the COBOL structure (050) describing the IDL interface (010).

(050) COBOL structure describing the IDL interface (010) defined in LINKAGE SECTION.

(060) Storage calculation for COBOL structure (050) assuming upper bound is 4 for UT-TA1-41 and 6 for UT-TA1-61.

(070) Storage allocation using the calculated SIZE-IN-BYTES (060) with IBM-specific COBOL ALLOCATE statement; returned address is assigned to COBOL pointer UT-TA1A-PTR (040).

(080) COBOL structure (050) is set to the address of the allocated storage (070).

(090) Upper bounds are assigned to ODO objects of COBOL structure (050).

(100) ODO subjects of COBOL structure (050) are filled with data.

(110) Call to the client interface object; COBOL pointer UT-TA1A-PTR(040) is passed as parameter; The COBOL name of the client interface object UNBNDTAB is customized, see ***Customize Automatically Generated Client Names***.

(120) The client interface object may return a changed COBOL pointer UT-TA1A-PTR. So the COBOL structure (050) is set to the address returned from the client interface object.

(130) Processing of returned data.

(140) Storage allocated in (070) or returned by call to client interface object (110) is freed.

14 Using RPC Authentication (Natural Security, Impersonation, Integration Server)

Introduction

This section explains how clients built with the COBOL Wrapper can communicate with the following:

- Natural RPC Servers running under Natural Security
- RPC servers running with impersonation. See Impersonation in the respective RPC Server documentation.
- EntireX Adapter Listener with enabled **Execute Service with Client Credentials**, see *Configuring Listeners* in the EntireX Adapter documentation.

For this you will need the following components:

- the ***Delivered Modules*** which are provided to create and get a security token
- the copybook `ERXCOMM` if an 8-byte RPC user ID, an 8-byte RPC password and an 8 byte RPC library are sufficient. See `ERXCOMM`.
- the copybook `ERXVSTR` to use a long RPC user ID, a long RPC password and, if required, to override the IDL library with a long RPC library. See `ERXVSTR`.
- We strongly recommend using SSL/TLS if you send an authentication as described here with the COBOL Wrapper to a secure partner. See ***Using SSL/TLS*** in this section and also *SSL/TLS Parameters for SSL Clients* under *SSL/TLS, HTTP(S), and Certificates with EntireX* in the platform-independent Administration documentation.

RPC Authentication Using Short RPC User ID/RPC Password (all Interface Types)

This approach allows a maximum of 8 bytes for each of RPC user ID, RPC password and RPC library. The code you write depends on the interface type:

- Call Interface
- EXEC CICS LINK Interface

Call Interface

This interface type applies to the scenarios ***CICS*** | ***Batch*** | ***IMS***.

To use RPC authentication using short RPC user ID, RPC password and RPC library

1 Declare and initialize the RPC communication area as described under ***Only Copybook `ERXCOMM` is Used*** under ***Step 1: Declare and Initialize the RPC Communication Area*** in section *Writing Standard Call Interface Clients*.

2 Create a security token with the function Create Security Token `CT` provided by the generic RPC services module.

```
* Set function to create security token
 MOVE "CT"   TO COMM-FUNCTION.
* Set RPC userid and RPC password in RPC Communication Area
 MOVE "RPC-USER" TO COMM-USERID.
 MOVE "RPC-PSWD" TO COMM-PASSWORD.
* Optional set RPC library e.g. for Natural Security
 MOVE "RPC-LIB" TO COMM-LIBRARY.
* Call generic RPC service module to create security token (see Note 1)
 CALL "COBSRVI" USING ERX-COMMUNICATION-AREA
 ON EXCEPTION
*   Perform error-handling
 NOT ON EXCEPTION
    IF (COMM-RETURN-CODE = 0) THEN
*      Perform success-handling
    ELSE
*      Perform error-handling (See Note 2)
    END-IF
 END-CALL.
 . . .
```

After successful return from creating the security token, the authentication fields in the RPC communication area are properly set, so they can be used in subsequent RPC requests.

EXEC CICS LINK Interface

This interface type applies to the scenario ***Using the COBOL Wrapper for CICS with `DFHCOMMAREA` Calling Convention (z/OS and z/VSE)***.

To use RPC authentication using short RPC user ID, RPC password and RPC library

1 Declare the RPC communication area as described under ***Step 1: Declare IDL Structures and RPC Communication Area*** in section *Writing `EXEC CICS LINK` Clients*.

2 Initialize the RPC communication area as described under ***Step 2: Initialize the RPC Communication Area*** under *Writing `EXEC CICS LINK` Clients*.

3 Create a security token with the function Create Security Token `CT` provided by the generic RPC services module.

```
 MOVE "CT"   TO COMM-FUNCTION.
* Set RPC userid and RPC password in RPC Communication Area
 MOVE "RPC-USER" TO COMM-USERID.
 MOVE "RPC-PSWD" TO COMM-PASSWORD.
* Optional set RPC library e.g. for Natural Security
 MOVE "RPC-LIB" TO COMM-LIBRARY.
* Call generic RPC service module to create security token
 EXEC CICS LINK PROGRAM  ("COBSRVI")
                RESP     (CICS-RESP1)
                RESP2    (CICS-RESP2)
                COMMAREA (ERX-COMMUNICATION-AREA)
                LENGTH   (LENGTH OF ERX-COMMUNICATION-AREA)
 END-EXEC.
 IF WORKRESP = DFHRESP(NORMAL)
    IF (COMM-RETURN-CODE = 0) THEN
*      Perform success-handling
    ELSE
*      Perform error-handling (See Note 2)
    END-IF
 ELSE
*   Perform error-handling
 END-IF.
```

After successful return from creating the security token, the authentication fields in the RPC communication area are properly set, so they can be used in subsequent RPC requests.

Notes:

1. If you are only using copybook `ERXCOMM` only, pass only the address of `ERXCOMM` to the generic RPC service module.
2. The field `COMM-RETURN-CODE` in the RPC communication area contains the error provided by the COBOL Wrapper. For the error messages returned, see *Error Messages and Codes*.

RPC Authentication Using Long RPC User ID/RPC Password (z/OS with Call Interface)

This section applies to the scenarios *CICS*, *Batch* and *IMS* with the `CALL` interface.

With this approach you can use a long RPC user ID, RPC password and RPC library. It requires the `ERXVSTR` copybook.

The RPC communication area extension copybook `ERXVSTR` is generated for *Target Operating System* z/OS and RPC clients using a call interface to its client interface object, meaning one of the following *Client Interface Types* is selected:

- *CICS with Standard Linkage Calling Convention*
- *Batch with Standard Linkage Calling Convention*
- *IMS BMP with Standard Linkage Calling Convention*
- *IMS MPP with Standard Linkage Calling Convention*

To use RPC authentication with long RPC user ID, RPC password and RPC library

1 Declare and initialize the RPC communication area as described under *Both Copybooks `ERXCOMM` and `ERXVSTR` are Used* under *Step 1: Declare and Initialize the RPC Communication Area* in section *Writing Standard Call Interface Clients*.

2 Create a security token with the function Create Security Token `CT` provided by the generic RPC services module.

```
* Set function to create security token
 MOVE "CT"   TO COMM-FUNCTION.
* Set long RPC userid in RPC Variable String Area
 INSPECT RPCUID TALLYING STR-LENGTH FOR CHARACTERS BEFORE SPACE.
 MOVE 1 TO STR-OFFSET.
 MOVE STR-OFFSET TO COMM-RPC-USERID-OFFSET.
 MOVE STR-LENGTH TO COMM-RPC-USERID-LENGTH.
 STRING RPCUID DELIMITED BY SPACE INTO
        COMM-STRING-AREA WITH POINTER STR-OFFSET.
* Set long RPC password in RPC Variable String Area
 INSPECT RPCPWD TALLYING STR-LENGTH FOR CHARACTERS BEFORE SPACE.
 MOVE STR-OFFSET TO COMM-RPC-PASSWORD-OFFSET.
 MOVE STR-LENGTH TO COMM-RPC-PASSWORD-LENGTH.
 STRING RPCPWD DELIMITED BY SPACE INTO
        COMM-STRING-AREA WITH POINTER STR-OFFSET.
* Optional set long RPC library e.g. for Natural Security
 INSPECT RPCLIB TALLYING STR-LENGTH FOR CHARACTERS BEFORE SPACE.
 MOVE STR-OFFSET TO COMM-RPC-LIBRARY-OFFSET.
 MOVE STR-LENGTH TO COMM-RPC-LIBRARY-LENGTH.
```

```
 STRING RPCLIB DELIMITED BY SPACE INTO
        COMM-STRING-AREA WITH POINTER STR-OFFSET.
* Set CCSID for encoding of RPC userid/password and application data (Note 3)
 MOVE "37" TO COMM-CCSID.
* Call generic RPC service module to create security token (Note 1)
 CALL "COBSRVI" USING ERX-COMMUNICATION-AREA
                      ERX-COMMUNICATION-VSTR.
 ON EXCEPTION
 * Perform error-handling
 NOT ON EXCEPTION
   IF (COMM-RETURN-CODE = 0) THEN
 *    Perform success-handling
   ELSE
 *    Perform error-handling (See Note 2)
   END-IF
 END-CALL.
 . . .
```

After successful return from creating the security token with a long RPC user ID/RPC password:

- The authentication fields in the RPC communication area are properly set, so they can be used in subsequent RPC requests.
- The RPC protocol is forced to 2050 as a minium. You need an RPC server supporting this protocol level, see *Supported RPC Protocols*.

Notes:

1. If both copybooks are used, you need to pass both addresses, first the address of `ERXCOMM`, then the address of `ERXVSTR` to the generic RPC service module.
2. The field `COMM-RETURN-CODE` in the RPC communication area contains the error provided by the COBOL Wrapper. For the error messages returned, see *Error Messages and Codes*.
3. If a `CCSID` is provided:
   - It is used for conversion of the long RPC password and RPC user ID. If no `CCSID` is provided, the codepage active during compilation applies. Refer to your compiler documentation.
   - It is used as the codepage name to tell the broker the encoding of your application data. See ***Using Internationalization with the COBOL Wrapper***.

15 Using the COBOL Wrapper with Non-secure Natural RPC Server

This chapter explains how clients built with the COBOL Wrapper set the Natural library used to execute the RPC request programmatically when communicating to a non-secure Natural RPC Server (not running with Natural Security). If the Natural RPC Server is running with Natural Security, see ***Using RPC Authentication (Natural Security, Impersonation, Integration Server)***.

You will need the following components:

- the ***Delivered Modules***, which are provided to create and get a security token
- the RPC communication area copybook `ERXCOMM`

The code you write depends on the interface type, Call Interface or `EXEC CICS LINK` Interface:

Call Interface

This interface type applies to the scenarios ***CICS*** | ***Batch*** | ***IMS***.

To set the Natural library when communicating to a non-secure Natural RPC server

1 Declare and initialize the RPC communication area with one of the approaches described under ***Step 1: Declare and Initialize the RPC Communication Area*** in section *Writing Standard Call Interface Clients*. Here you can use **copybook `ERXCOMM` only**, or **both copybooks `ERXCOMM` and `ERXVSTR`**.

2 Set the library in RPC Communication Area and call generic RPC service module to create a security token with the function Create Security Token `CT` provided by the generic RPC services module, using the Call interface. The code snippet below illustrates the variant where only copybook `ERXCOMM` is used. If you are using both copybooks `ERXCOMM` and `ERXVSTR`, see Note 1.

```
  . . .
 MOVE "CT"   TO COMM-FUNCTION.
* Set library in RPC Communication Area
 MOVE "NAT-LIB"   TO COMM-LIBRARY.
* Call generic RPC service module to create security token (Note 1)
 CALL "COBSRVI" USING ERX-COMMUNICATION-AREA
*   Perform error-handling
 NOT ON EXCEPTION
   IF (COMM-RETURN-CODE = 0) THEN
*     Perform success-handling
   ELSE
*     Perform error-handling (See Note 2)
   END-IF
 END-CALL.
```

After successful return from the generic RPC services module, the required fields in the RPC communication area are properly set, so the non-secure Natural RPC server executes the RPC request in the library set.

Notes:

1. If both copybooks `ERXCOMM` and `ERXVSTR` are used, you need to pass both parameters:

```
CALL "COBSRVI" USING ERX-COMMUNICATION-AREA
                     ERX-COMMUNICATION-VSTR.
```

2. The field `COMM-RETURN-CODE` in the RPC communication area contains the error provided by the COBOL Wrapper. For the error messages returned, see *Error Messages and Codes*.

EXEC CICS LINK Interface

This interface type applies to the scenario ***Using the COBOL Wrapper for CICS with `DFHCOMMAREA` Calling Convention (z/OS and z/VSE)***.

To set the Natural library when communicating to a non-secure Natural RPC server

1 Declare the RPC communication area as described under ***Step 1: Declare IDL Structures and RPC Communication Area*** in section *Writing `EXEC CICS LINK` Clients*.

2 Initialize the RPC communication area as described under ***Step 2: Initialize the RPC Communication Area*** under *Writing `EXEC CICS LINK` Clients*.

3 Set the library in RPC Communication Area and call generic RPC service module to create a security token with the function Create Security Token `CT` provided by the generic RPC services module, using `EXEC CICS LINK`.

```
  MOVE "CT"   TO COMM-FUNCTION.
* Set library in RPC Communication Area
  MOVE "NAT-LIB"   TO COMM-LIBRARY.
  EXEC CICS LINK PROGRAM  ("COBSRVI")
                 RESP     (CICS-RESP1)
                 RESP2    (CICS-RESP2)
                 COMMAREA (ERX-COMMUNICATION-AREA)
                 LENGTH   (LENGTH OF ERX-COMMUNICATION-AREA)
  END-EXEC.
  IF WORKRESP = DFHRESP(NORMAL)
     IF (COMM-RETURN-CODE = 0) THEN
*       Perform success-handling
     ELSE
*       Perform error-handling (See Note 1)
     END-IF
  ELSE
*    Perform error-handling
  END-IF.
```

After successful return from the generic RPC services module, the required fields in the RPC communication area are properly set, so the non-secure Natural RPC server executes the RPC request in the library set.

Notes:

1. The field `COMM-RETURN-CODE` in the RPC communication area contains the error provided by the COBOL Wrapper. For the error messages returned, see *Error Messages and Codes*.

16 Using SSL/TLS

RPC client applications can use Secure Sockets Layer/Transport Layer Security (SSL/TLS) as the transport medium. The term “SSL” in this chapter refers to both SSL and TLS. RPC-based clients are always SSL clients. The SSL server can be either the EntireX Broker, Broker SSL Agent, or Direct RPC in webMethods Integration Server (IS inbound). For an introduction see *SSL/TLS, HTTP(S), and Certificates with EntireX* in the platform-independent Administration documentation. This chapter describes using SSL with the COBOL Wrapper on z/OS and z/VSE.

z/OS

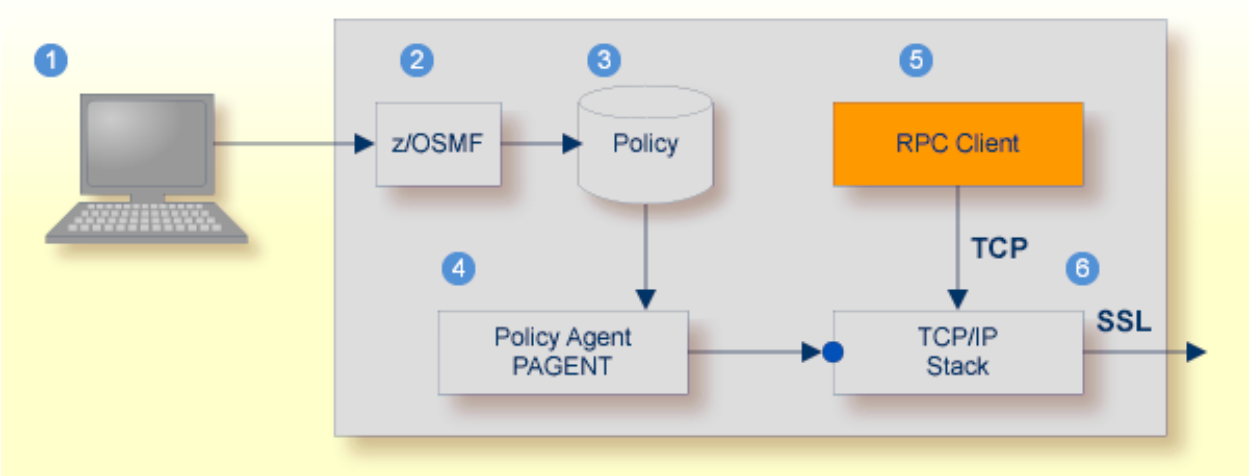
SSL delivered on a z/OS mainframe will typically use the Resource Access Control Facility (RACF) as the certificate authority (CA). Certificates managed by RACF can only be accessed through the RACF keyring container. A keyring is a collection of certificates that identify a networking trust relationship (also called a trust policy). In an SSL client/server network environment, entities identify themselves using digital certificates called through a keyring. Server applications on z/OS that wish to establish network connections to other entities can use keyrings and their certificate contents to determine the trustworthiness of the client or peer entity. Note that certificates can belong to more than one keyring, and you can assign different users to the same keyring. Because of the way RACF internally references certificates, they must be uniquely identifiable by owner and label, and also unique by serial number plus data set name (DSN).

For establishing an SSL connection on z/OS, IBM's Application Transparent Transport Layer Security (AT-TLS) can be used, where the establishment of the SSL connection is pushed down the stack into the TCP layer.

With the COBOL Wrapper you can use IBM's Application Transparent Transport Layer Security (AT-TLS), where the establishment of the SSL connection is pushed down the stack into the TCP layer.

Using IBM's Application Transparent Transport Layer Security (AT-TLS)

Configure the AT-TLS rules for the policy agent (`PAGENT`) 4 using an appropriate client 1 and the z/OS Management Facility (z/OSMF) 2. Together with SSL parameters (to provide certificates stored in z/OS as RACF keyrings) define AT-TLS rules, for example by using the application 5 job name and remote TCP port number. If the rules match, the TCP connection is turned into an SSL connection 6. Refer to your IBM documentation for more information, for example the IBM Redbook *Communications Server for z/OS VxRy TCP/IP Implementation Volume 4: Security and Policy-Based Networking*.



1. Client to interact with z/OS Management Facility (z/OSMF).
2. AT-TLS rules are defined with z/OSMF policy management.
3. Policy Repository with AT-TLS rules stored as z/OS files.
4. Policy Agent, MVS task `PAGENT`, provides AT-TLS rules through a policy enforcement point (PEP) to TCP/IP stack.
5. Application using TCP connection.
6. If AT-TLS rules match, the TCP connection is turned into an SSL connection.

Notes:

1. The client 1 may vary per operating system, for example a Web browser for z/OS 2.1.
2. z/OSMF 2 includes other administration and management tasks in addition to policy management.
3. Policy Management 3 includes other rules, such as IP filtering, network address translation etc.

To set up SSL with AT-TLS

1 To operate with SSL, certificates need to be provided and maintained. Depending on the platform, Software AG provides default certificates, but we strongly recommend that you create your own. See *SSL/TLS Sample Certificates Delivered with EntireX* in the EntireX Security documentation.

2 Set up the RPC component for a TCP/IP connection. On mainframe platforms, use *Transport-method-style Broker ID*. Example:

```
ETB024:1699:TCP
```

3 Configure AT-TLS to turn the TCP/IP connection to an SSL connection, using a client to interact with the z/OS Management Facility (z/OSMF). The outcome of this configuration is a Policy Repository with AT-TLS rules stored as z/OS files. This file is the configuration file for the Policy Agent, MVS task `PAGENT`.

4 Make sure the SSL server to which the RPC component connects is prepared for SSL connections as well. The SSL server can be EntireX Broker, Broker SSL Agent, or Direct RPC in webMethods Integration Server (IS inbound). See:

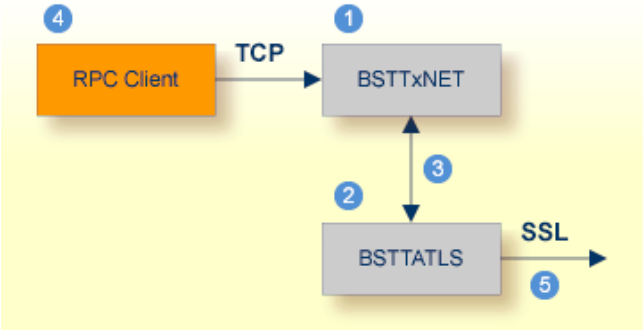
- *Running Broker with SSL/TLS Transport* in the platform-specific Administration documentation
- Broker SSL Agent in the platform-specific Administration documentation
- *Support for SSL/TLS* in the EntireX Adapter documentation (for Direct RPC)

z/VSE

Establishing an SSL connection on z/VSE requires BSI's Automatic Transport Layer Security (ATLS). This facility is similar to z/OS Application Transparent - Transport Layer Security (AT-TLS). ATLS is supported by the BSI stack only.

Using BSI's Automatic Transport Layer Security (ATLS)

Together with SSL parameters (to provide certificates), define ATLS rules for socket interception in the ATLS daemon startup job `BSTTATLS` (2). If the rules match, the socket connection is turned into an SSL connection (5). Refer to your IBM documentation for further information. For an overview, refer to the IBM Redbook *Enhanced Networking on IBM z/VSE*; for a more detailed description, refer to *BSI SSL Installation, Programming and User's Guide*.



(1) BSI TCP/IP Stack, either BSTTINET (IPv4) or BSTT6NET (IPv6).

2. ATLS rules are defined manually. See Sample ATLS Daemon Configuration below.
3. BSTTATLS is associated with a TCP/IP stack.
4. Application using TCP connection.
5. BSTTATLS intercepts outbound TCP connection and converts it to SSL connection. For inbound, SSL connections can also be intercepted and converted to TCP connections.

To set up SSL with ATLS

1. To operate with SSL, certificates need to be provided and maintained. Depending on the platform, Software AG provides default certificates, but we strongly recommend that you create your own. See *SSL/TLS Sample Certificates Delivered with EntireX* in the EntireX Security documentation.
2. Set up the RPC component for a TCP/IP connection. On mainframe platforms, use *Transport-method-style Broker ID*. Example:

```
ETB024:1699:TCP
```

3. Configure ATLS to turn the TCP/IP connection to an SSL connection, see above.
4. Make sure the SSL server to which the RPC component connects is prepared for SSL connections as well. The SSL server can be EntireX Broker, Broker SSL Agent, or Direct RPC in webMethods Integration Server (IS inbound). See:

   - *Running Broker with SSL/TLS Transport* in the platform-specific Administration documentation
   - Broker SSL Agent in the platform-specific Administration documentation
   - *Support for SSL/TLS* in the EntireX Adapter documentation (for Direct RPC)

Sample ATLS Daemon Configuration

```
* Converting inbound EntireX Broker connection
* Converts listen port 1971 to SSL listen port 1972
OPTION SERVER
ATTLS 1971 AS 2071 SSL
*
* Converting outbound client connection
* Converts connect to 192.168.2.100:1972:TCP to 192.168.2.100:2072:SSL
OPTION CLIENT
ATTLS 1972 TO 192.168.2.100 AS 2072 SSL
```

Note: We recommend setting `SETPARM` value `SUBTASK` to a value greater than 0 in the ATLS daemon startup job (valid values 0-16, default=0). For example:

```
// SETPARM SUBTASK=8
```

See also *BSI SSL Installation, Programming and User's Guide*.

UNIX, Windows, BS2000

RPC client applications built with the COBOL Wrapper do not support Secure Sockets Layer/Transport Layer Security (SSL/TLS) as the transport medium under UNIX, Windows or BS2000.

17 Using Internationalization with the COBOL Wrapper

RPC clients generated with the COBOL Wrapper do *not* convert your application data (in RPC IDL type A, K, AV and KV fields) before it is sent to the broker. The application's data is shipped as given by the RPC client program.

- **For Operating System z/OS**
  - By default, no codepage is transferred to the broker. It is assumed the broker's locale string defaults match. See *Broker's Locale String Defaults*.
  - You can provide the CCSID in the field `COMM-CCSID` of copybook `ERXCOMM` to tell the broker the encoding of your application data. Do this before issuing broker and RPC calls, for example in Step *Optional Settings in the RPC Communication Area* (Call Interface | CICS).

    Example:

    ```
    MOVE 37 TO COMM-CCSID.
    ```

  - If a `CCSID` is provided, it is sent as `CP<number>` to the broker. It must be a codepage supported by the broker and follow the rules described under *Locale String Mapping*.
- **For Operating System Windows**
  - The Generic RPC Service module assumes the data is given in the encoding of the Windows ANSI codepage configured for your system. A codepage identifier of this Windows ANSI codepage is automatically transferred to tell the broker how the data is encoded.
  - If you want to adapt the Windows ANSI codepage, see the Regional Settings in the Windows Control Panel and your Windows documentation.
- **For all other Operating System**
  - No codepage is transferred to the broker. It is assumed the broker's locale string defaults match. See *Broker's Locale String Defaults*.

Enable character conversion in the broker by setting the service-specific attribute `CONVERSION` to "`SAGTRPC`". See also *Configuring ICU Conversion* under *Configuring Broker for Internationalization* in the platform-specific Administration documentation. More information can be found under *Internationalization with EntireX*.

IV Reliable RPC for COBOL Wrapper

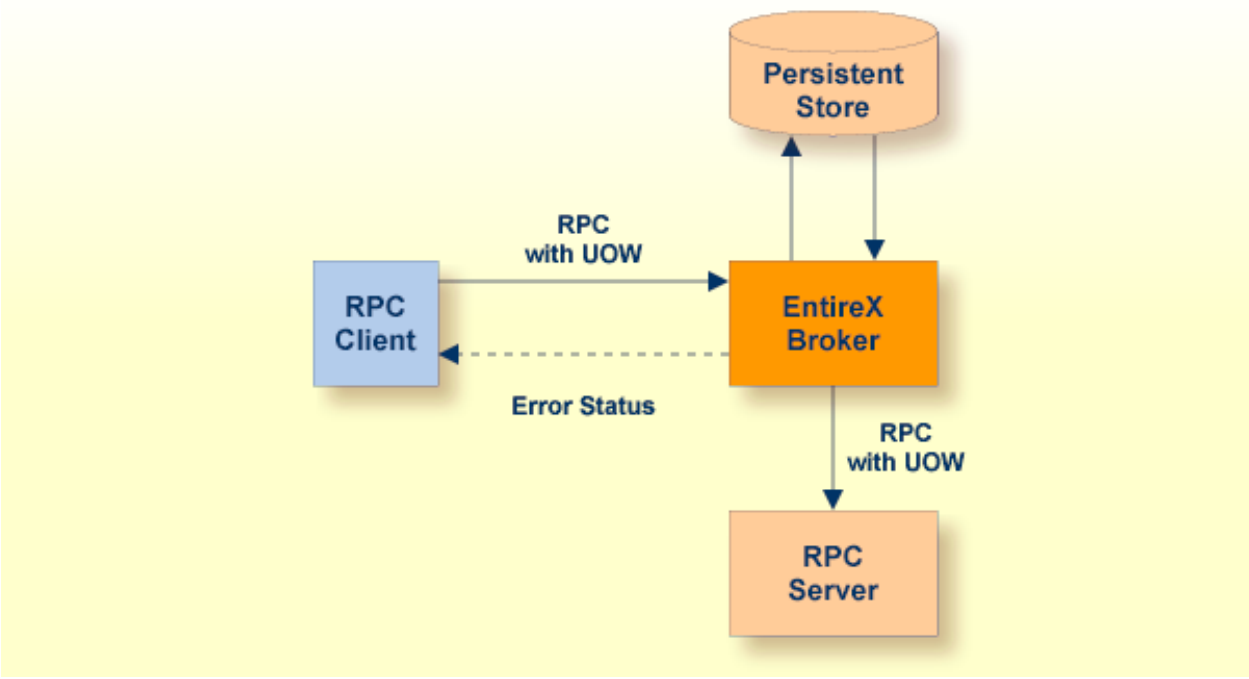
18 Reliable RPC for COBOL Wrapper

Introduction to Reliable RPC

In the architecture of modern e-business applications (such as SOA), loosely coupled systems are becoming more and more important. Reliable messaging is one important technology for this type of system.

Reliable RPC is the EntireX implementation of a reliable messaging system. It combines EntireX RPC technology and persistence, which is implemented with units of work (UOWs).

- Reliable RPC allows asynchronous calls (“fire and forget”)
- Reliable RPC is supported by most EntireX wrappers
- Reliable RPC messages are stored in the Broker's persistent store until a server is available
- Reliable RPC clients are able to request the status of the messages they have sent



Reliable RPC is used to send messages to a persisted Broker service. The messages are described by an IDL program that contains only `IN` parameters. The client interface object and the server interface object are generated from this IDL file, using the EntireX COBOL Wrapper.

Reliable RPC is enabled at runtime. The client has to set one of two different modes before issuing a reliable RPC request:

- `AUTO_COMMIT`
- `CLIENT_COMMIT`

While AUTO_COMMIT commits each RPC message implicitly after sending it, a series of RPC messages sent in a unit of work (UOW) can be committed or rolled back explicitly using CLIENT_COMMIT mode.

The server is implemented and configured in the same way as for normal RPC.

Writing a Client

The following steps describe how to write a COBOL reliable RPC client program with the scenario *Using the COBOL Wrapper for CICS with DFHCOMMAREA Calling Convention (z/OS and z/VSE)*:

- Step 1: Declare the Data Structures and RPC Communication Area
- Step 2: Initialize the RPC Communication Area
- Step 3: Required Settings in the RPC Communication Area
- Step 4: Perform a Broker Logon
- Step 5: Enable Reliable RPC with CLIENT_COMMIT
- Step 6: Send the RPC Message
- Step 7: Check the Reliable RPC Message Status
- Step 8: Send a Second RPC Message
- Step 9: Check the Reliable RPC Message Status
- Step 10: Commit both Reliable RPC Messages
- Step 11: Send a Third RPC Message
- Step 12: Check the Reliable RPC Message Status
- Step 13: Roll Back the Third RPC Message
- Step 14: Check the Reliable RPC Message Status
- Step 15: Perform a Broker Logoff

Note: Reliable RPC requires an explicit broker logon. See *Using Broker Logon and Logoff*.

Step 1: Declare the Data Structures and RPC Communication Area

The customer data structures (all below COBOL label SM-COMA) must match the interfaces of the generated client interface objects with regard to format and lengths, otherwise unpredictable results will occur. See the following code snippet:

```
* Declare the customer data of the generated RPC interface (See Note 1)
 01 SENDMAIL.
    02 SM-COMA.
       03 SM-TOADDRESS          PIC  X(60).
       03 SM-SUBJECT            PIC  X(20).
       03 SM-TEXT               PIC X(100).
* Declare RPC communication area
    02 ERX-COMMUNICATION-AREA.
       COPY ERXCOMM.
```

Notes:

1. As an alternative the generated IDL Interface copybooks can be used. See ***Step 1: Declare IDL Structures and RPC Communication Area*** in section *Writing `EXEC CICS LINK` Clients* for usage examples.

Step 2: Initialize the RPC Communication Area

See the following code snippet and refer to ***Step 2: Initialize the RPC Communication Area*** under *Writing `EXEC CICS LINK` Clients* for additional hints and information.

```
* Initialize RPC communication area
 INITIALIZE ERX-COMMUNICATION-AREA.
 MOVE "2000"           to COMM-VERSION.
```

Step 3: Required Settings in the RPC Communication Area

The following settings to the RPC communication area are required as a minimum to use the COBOL Wrapper. These settings have to be applied in your RPC client program. It is not possible to generate any defaults into your client interface objects:

```
* assign the broker to talk with
 MOVE "localhost:1971" to COMM-ETB-BROKER-ID.

* assign the server to talk with
 MOVE "RPC"            to COMM-ETB-SERVER-CLASS.
 MOVE "SRV1"           to COMM-ETB-SERVER-NAME.
 MOVE "CALLNAT"        to COMM-ETB-SERVICE-NAME.
* assign the user ID for Broker logon
 MOVE "ERXUSER"        to COMM-USERID.
```

Step 4: Perform a Broker Logon

```
MOVE "LO" TO COMM-FUNCTION.
EXEC CICS LINK
  PROGRAM  ("COBSRVI")
  COMMAREA (ERX-COMMUNICATION-AREA)
  LENGTH   (LENGTH OF ERX-COMMUNICATION-AREA)
  RESP     (CICS-RESP1)
  RESP2    (CICS-RESP2)
END-EXEC.
IF WORKRESP = DFHRESP(NORMAL)
    IF (COMM-RETURN-CODE = 0) THEN
*       Perform success-handling
    ELSE
*       Perform error-handling (See Note 1)
    END-IF
 ELSE
```

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

Notes:

1. The field COMM-RETURN-CODE in the RPC communication area contains the error provided by the COBOL Wrapper. For the error messages returned, see *Error Messages and Codes*.

Step 5: Enable Reliable RPC with CLIENT_COMMIT

Before reliable RPC can be used, the reliable state must be set to either ERX_RELIABLE_CLIENT_COMMIT or ERX_RELIABLE_AUTO_COMMIT.

- "C" - CLIENT_COMMIT
- "A" - AUTO_COMMIT

```
* Set the reliable RPC mode
 MOVE "C" TO COMM-RELIABLE-STATE.
```

Step 6: Send the RPC Message

The RPC message is sent using the EXEC CICS LINK interface.

```
* Send the RPC message
 MOVE DFHRESP(NORMAL) TO CICS-RESP1.
 MOVE DFHRESP(NORMAL) TO CICS-RESP2.
 MOVE ZEROES          TO COMM-RETURN-CODE.
 EXEC CICS LINK
   PROGRAM  ("SENDMAIL")
   RESP     (CICS-RESP1)
   RESP2    (CICS-RESP2)
   COMMAREA (SENDMAIL)
   LENGTH   (LENGTH OF SENDMAIL)
 END-EXEC.
 IF WORKRESP = DFHRESP(NORMAL)
   IF (COMM-RETURN-CODE = 0) THEN
*      Perform success-handling (See Note 1)
   ELSE
*      Perform error-handling (See Note 2)
   END-IF
 ELSE
*  Perform error-handling
 END-IF.
```

Notes:

1. After successful call the UOWID is available in the RPC communication area field COMM-ETB-UOW-ID. See *The RPC Communication Area (Reference)*.

2. The field COMM-RETURN-CODE in the RPC communication area contains the error provided by the COBOL Wrapper. For the error messages returned, see *Error Messages and Codes*.

Step 7: Check the Reliable RPC Message Status

To determine that reliable RPC messages are delivered, the reliable RPC message status can be queried. See *Understanding UOW Status* and *Broker UOW Status Transition* for more information.

```
 MOVE DFHRESP(NORMAL) TO CICS-RESP1.
 MOVE DFHRESP(NORMAL) TO CICS-RESP2.
 MOVE "RS" TO COMM-FUNCTION.
 MOVE ZEROES TO COMM-RETURN-CODE.
 EXEC CICS LINK
   PROGRAM  ("COBSRVI")
   RESP     (CICS-RESP1)
   RESP2    (CICS-RESP2)
   COMMAREA (ERX-COMMUNICATION-AREA)
   LENGTH   (LENGTH OF ERX-COMMUNICATION-AREA)
 END-EXEC.
 IF WORKRESP = DFHRESP(NORMAL)
    IF (COMM-RETURN-CODE = 0) THEN
*       Perform success-handling (See Note 1)
    ELSE
*       Perform error-handling (See Note 2)
    END-IF
  ELSE
*   Perform error-handling
  END-IF.
```

Notes:

1. After successful call the UOW status is available in the RPC communication area field COMM-ETB-UOW-STATUS. See ***The RPC Communication Area (Reference)***.
2. The field COMM-RETURN-CODE in the RPC communication area contains the error provided by the COBOL Wrapper. For the error messages returned, see *Error Messages and Codes*.

Step 8: Send a Second RPC Message

Send a second reliable RPC message. See **Step 6**.

Step 9: Check the Reliable RPC Message Status

Check the reliable RPC message before the commit call. See **Step 7**.

Step 10: Commit both Reliable RPC Messages

Now both reliable RPC messages are committed. This will deliver all reliable RPC messages to the server if it is available.

```
 MOVE DFHRESP(NORMAL) TO CICS-RESP1.
 MOVE DFHRESP(NORMAL) TO CICS-RESP2.
 MOVE "RC" TO COMM-FUNCTION.
 MOVE ZEROES TO COMM-RETURN-CODE.
 EXEC CICS LINK
   PROGRAM  ("COBSRVI")
   RESP     (CICS-RESP1)
   RESP2    (CICS-RESP2)
   COMMAREA (ERX-COMMUNICATION-AREA)
   LENGTH   (LENGTH OF ERX-COMMUNICATION-AREA)
 END-EXEC.
 IF WORKRESP = DFHRESP(NORMAL)
   IF (COMM-RETURN-CODE = 0) THEN
*      Perform success-handling (See Note 1)
   ELSE
*      Perform error-handling (See Note 2)
   END-IF
 ELSE
*  Perform error-handling
 END-IF.
```

Notes:

1. After successful call, both reliable RPC messages are committed. This will deliver all reliable RPC messages to the server if it is available.
2. The field `COMM-RETURN-CODE` in the RPC communication area contains the error provided by the COBOL Wrapper. For the error messages returned, see *Error Messages and Codes*.

Step 11: Send a Third RPC Message

Send a third reliable RPC message. See **Step 6**.

Step 12: Check the Reliable RPC Message Status

Check the reliable RPC message before the rollback call. See Step 7.

Step 13: Roll Back the Third RPC Message

Roll back the current reliable RPC message.

```
 MOVE DFHRESP(NORMAL) TO CICS-RESP1.
 MOVE DFHRESP(NORMAL) TO CICS-RESP2.
 MOVE "RR" TO COMM-FUNCTION.
 MOVE ZEROES TO COMM-RETURN-CODE.
 EXEC CICS LINK
   PROGRAM  ("COBSRVI")
   RESP     (CICS-RESP1)
   RESP2    (CICS-RESP2)
   COMMAREA (ERX-COMMUNICATION-AREA)
   LENGTH   (LENGTH OF ERX-COMMUNICATION-AREA)
 END-EXEC.
 IF WORKRESP = DFHRESP(NORMAL)
   IF (COMM-RETURN-CODE = 0) THEN
*      Perform success-handling
   ELSE
*      Perform error-handling (See Note 1)
   END-IF
 ELSE
*  Perform error-handling
 END-IF.
```

Step 14: Check the Reliable RPC Message Status

When the rollback call is returned, check whether it was successful or not.

```
 MOVE DFHRESP(NORMAL) TO CICS-RESP1.
 MOVE DFHRESP(NORMAL) TO CICS-RESP2.
 MOVE "RS" TO COMM-FUNCTION.
 MOVE ZEROES TO COMM-RETURN-CODE.
 EXEC CICS LINK
   PROGRAM  ("COBSRVI")
   RESP     (CICS-RESP1)
   RESP2    (CICS-RESP2)
   COMMAREA (ERX-COMMUNICATION-AREA)
   LENGTH   (LENGTH OF ERX-COMMUNICATION-AREA)
 END-EXEC.
 IF WORKRESP = DFHRESP(NORMAL)
   IF (COMM-RETURN-CODE = 0) THEN
*      Perform success-handling
   ELSE
*      Perform error-handling (See Note 1)
   END-IF
```

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

Notes:

1. The field `COMM-RETURN-CODE` in the RPC communication area contains the error provided by the COBOL Wrapper. For the error messages returned, see *Error Messages and Codes*.

Step 15: Perform a Broker Logoff

```
MOVE "LF" TO COMM-FUNCTION.
EXEC CICS LINK
  PROGRAM  ("COBSRVI")
  COMMAREA (ERX-COMMUNICATION-AREA)
  LENGTH   (LENGTH OF ERX-COMMUNICATION-AREA)
  RESP     (CICS-RESP1)
  RESP2    (CICS-RESP2)
END-EXEC.
IF WORKRESP = DFHRESP(NORMAL)
    IF (COMM-RETURN-CODE = 0) THEN
*       Perform success-handling
    ELSE
*       Perform error-handling (See Note 1)
    END-IF
  ELSE
*   Perform error-handling
  END-IF.
```

Notes:

1. The field `COMM-RETURN-CODE` in the RPC communication area contains the error provided by the COBOL Wrapper. For the error messages returned, see *Error Messages and Codes*.

Writing a Server

There are no server-side methods for reliable RPC. The server does not send back a message to the client. The server can run deferred, thus client and server do not necessarily run at the same time. If the server fails, it returns an error code greater than zero. This causes the transaction (unit of work inside the Broker) to be cancelled, and the error code is written to the user status field of the unit of work. For writing reliable RPC servers, see ***Using the COBOL Wrapper for the Server Side***.

To execute a reliable RPC service with an RPC server, the parameter `logon` (`LOGN` under CICS) must be set to `YES`. See `logon` under z/OS (CICS | Batch | IMS) | BS2000 in the respective RPC Server documentation.

Broker Configuration

A Broker configuration with PSTORE is recommended. This enables the Broker to store the messages for more than one Broker session. These messages are still available after Broker restart. The attributes STORE, PSTORE, and PSTORE-TYPE in the Broker attribute file can be used to configure this feature. The lifetime of the messages and the status information can be configured with the attributes UOW-DATA-LIFETIME and UOW-STATUS-LIFETIME. Other attributes such as MAX-MESSAGES-IN-UOW, MAX-UOWS and MAX-UOW-MESSAGE-LENGTH may be used in addition to configure the units of work. See *Broker Attributes*.

The result of the generic RPC function call "RS" - get reliable status depends on the configuration of the unit of work status lifetime in the EntireX Broker configuration. See COMM-FUNCTION. If the status is not stored longer than the message, the function call returns the error code 00780305 (no matching UOW found).

V Delivered Examples for the COBOL Wrapper

This chapter describes the following examples provided for the COBOL Wrapper:

- ***Client and Server Examples for z/OS Batch***
- ***Client and Server Examples for z/OS CICS***
- ***Client and Server Examples for z/OS IMS BMP***
- ***Server Examples for z/OS IMS MPP***
- ***Client and Server Examples for BS2000***
- *Client and Server Examples for z/VSE Batch*
- *Client and Server Examples for z/VSE CICS*

19 Client and Server Examples for z/OS Batch

This chapter describes the RPC examples provided. After installation of the *EntireX Development Tools* package, all examples here can be found in the EntireX directory *examples/RPC*. They are also available as a z/OS data set, see *Installing RPC Examples for z/OS*.

Basic RPC Client Examples - CALC, SQUARE

CALC Client

For z/OS Batch, the `CALC` client is built with COBOL Wrapper "Batch with standard linkage calling convention" interface type. See ***Client Interface Types*** for more information.

Name	Type	Data Set	Description	Notes
`CALC`	COBOL source code	EXP108.CCCO	Client interface object for IDL program CALC.	1
`CALCCLT`	COBOL source code	EXP108.CCCO	A client application calling the remote procedure (RPC service) CALC, with associated *example.idl*.	2
`CALCIGY`	JCL	EXP108.CCCO	Job (JCL) to build the RPC client CALCCLT.	3
`CALCRUN`	JCL	EXP108.CCCO	Job (JCL) to execute the RPC client CALCCLT.	3
`CALC`	COBOL copybook	EXP108.CICO	Client interface object copybook for IDL program CALC.	1
`COBSRVI`	COBOL source code	EXP108.CCCO	Generic RPC service module for Batch.	4

Notes:

1. Under z/OS, client interface objects are delivered with the installation; under UNIX and Windows, generate these objects with the Designer.
2. Application built according to the client-side build instructions, see ***Using the COBOL Wrapper for Batch (z/OS, BS2000, z/VSE and IBM i)***.
3. Adapt the JCL to your needs.
4. See ***Generation and Usage of Generic RPC Service Module COBSRVI***.

For more information refer to the readme file in EntireX directory *examples/RPC/CobolClient/zosBatch* under UNIX or Windows.

SQUARE Client

For batch under operating system z/OS, the SQUARE client is built with COBOL Wrapper "Batch with standard linkage calling convention" interface type. See *Client Interface Types* for more information.

Name	Type	Data Set	Description	Notes
COBSRVI	COBOL source code	EXP108.CCCO	Generic RPC service module for Batch.	4
SQRECLT	COBOL source code	EXP108.CCCO	A client application calling the remote procedure (RPC service) SQUARE, with associated example.idl.	1
SQREIGY	JCL	EXP108.CCCO	Job (JCL) to build the RPC client SQRECLT.	2
SQRERUN	JCL	EXP108.CCCO	Job (JCL) to execute the RPC client SQRECLT.	2
SQUARE	COBOL source code	EXP108.CCCO	Client interface object for IDL program SQUARE.	3
SQUARE	COBOL copybook	EXP108.CICO	Client interface object copybook for IDL program SQUARE.	3

Notes:

1. Application built according to the client-side build instructions, see *Using the COBOL Wrapper for Batch (z/OS, BS2000, z/VSE and IBM i)*.
2. Adapt the JCL to your needs.
3. Under z/OS, client interface objects are delivered with the installation; under UNIX and Windows, generate these objects with the Designer.
4. See *Generation and Usage of Generic RPC Service Module COBSRVI*.

For more information, see the readme file in EntireX directory *examples/RPC/CobolClient/zosBatch* under UNIX or Windows.

Basic RPC Server Examples - CALC, SQUARE

CALC Server

For batch under operating system z/OS, the CALC server is built with COBOL Wrapper "Batch with standard linkage calling convention" interface type. See *Server Interface Types* for more information.

Name	Type	Data Set	Description	Notes
CALC	COBOL source code	EXP108.CVCO	A server application providing the remote procedure CALC (RPC service), with associated *example.idl*.	1
CALCIGY	JCL	EXP108.CVCO	Job (JCL) to build the remote procedure CALC (RPC service).	2

Notes:

1. Application built according to the server-side build instructions, see *Using the COBOL Wrapper for Batch (z/OS, BS2000 and IBM i)*.
2. Adapt the JCL to your needs.

For more information, refer to the readme file in EntireX directory *examples/RPC/CobolServer/zosBatch* under UNIX or Windows.

SQUARE Server

For batch on operating system z/OS, the SQUARE server is built with COBOL Wrapper "Batch with standard linkage calling convention" interface type. See *Client Interface Types* for more information.

Name	Type	Data Set	Description	Notes
SQREIGY	JCL	EXP108.CVCO	Job (JCL) to build the remote procedure SQUARE (RPC service)	2
SQUARE	COBOL source code	EXP108.CVCO	a server application providing the remote procedure SQUARE (RPC service), with associated *example.idl*	1

Notes:

1. Application built according to the server-side build instructions, see *Using the COBOL Wrapper for Batch (z/OS, BS2000, z/VSE and IBM i)*.

2. Adapt the JCL to your needs.

For more information, refer to the readme file in EntireX directory *examples/RPC/CobolServer/zosBatch* under UNIX or Windows.

20 Client and Server Examples for z/OS CICS

This chapter describes the RPC examples provided. After installation of the *EntireX Development Tools* package, all examples here can be found in the EntireX directory *examples/RPC*. They are also available for z/OS, if this is installed. See *Installing RPC Examples for z/OS*.

Basic RPC Client Examples - CALC, SQUARE

CALC Client using Call Interface

For CICS under operating system z/OS, the following CALC client is implemented with interface type "CICS with standard linkage calling convention". See ***Client Interface Types*** for more information.

Name	Type	Data Set	Description	Notes
CALC	COBOL source code	EXP108.DCCO	Client interface object for IDL program CALC.	1
CALCCLT	COBOL source code	EXP108.DCCO	An RPC client application calling the remote procedure (RPC service) CALC.	2
CALCDFH	CICS CSD	EXP108.DCCO	CSD Definition for RPC client CALCCLT.	
CALCIGY	JCL	EXP108.DCCO	Job (JCL) to build the RPC client CALCCLT.	3
CALCMAP	CICS Map	EXP108.DCCO	CICS Map definition for RPC client CALCCLT.	
CALC	COBOL copybook	EXP108.DICO	Client interface object copybook for IDL program CALC.	1
CALCMAP	COBOL copybook	EXP108.DICO	Description of input and output fields of map CALCMAP.	
COBSRVI	COBOL source code	EXP108.DICO	Generic RPC service module for CICS with call interface.	4

Notes:

1. Under z/OS, client interface objects are delivered with the installation; under UNIX and Windows, generate these objects with the Designer.
2. Application
   a. built according to the client-side build instructions, see ***Using the COBOL Wrapper for CICS with Call Interfaces (z/OS and z/VSE)***
   b. associated with IDL file *example.idl*
   c. CALCCLT uses CICS Map definition CALCMAP

d. CALCCLT and client interface object CALC are linked together

e. CALCCLT installed as single CICS program

3. Adapt the JCL to your needs.
4. See *Generation and Usage of Generic RPC Service Module COBSRVI*.

For more information, refer to the readme file in EntireX directory *examples/RPC/CobolClient/zos-CICS/Callinterface* under UNIX or Windows.

CALC Client using DFHCOMMAREA

For CICS under operating system z/OS, the following CALC client is implemented with interface type "CICS with DFHCOMMAREA calling convention". See *Client Interface Types* for more information.

Name	Type	Data Set	Description	Notes
CALC1DFH	CICS CSD	EXP108.DCCO	CSD Definition for RPC client CALC1CLT.	
CALC1IGY	JCL	EXP108.DCCO	Job (JCL) to build the RPC client CALC1CLT.	2
CALC1MAP	CICS Map	EXP108.DCCO	CICS Map definition for RPC client and CALC1CLT.	
CALC1	COBOL source code	EXP108.DCCO	Client interface object for IDL program CALC1, alias of CALC.	1
CALC1CLT	COBOL source code	EXP108.DCCO	An RPC client application calling the remote procedure (RPC service) CALC.	3
CALC1MAP	COBOL copybook	EXP108.DICO	Description of input and output fields of map CALC1MAP.	
CALC1	COBOL copybook	EXP108.DICO	Client interface object copybook for IDL program CALC1, alias of CALC.	1

Notes:

1. Under z/OS, client interface objects are delivered with the installation; under UNIX and Windows, generate these objects with the Designer.
2. Adapt the JCL to your needs.
3. Application

   a. built according to the client-side build instructions, see *Using the COBOL Wrapper for CICS with DFHCOMMAREA Calling Convention (z/OS and z/VSE)*.

   b. associated with IDL file *exampleWithPgmAlias.idl*, delivered under UNIX and Windows in EntireX directory *examples/RPC/CobolClient/zosCICS/DFHCOMMAREA*.

   c. client interface object name CALC1 different from remote procedure name CALC (RPC service).

   d. CALC1CLT and client interface objects CALC1 installed as separate CICS programs.

For more information, refer to the readme file in EntireX directory *examples/RPC/CobolClient/zos-CICS/DFHCOMMAREA* under UNIX or Windows.

SQUARE Client using Call Interface

For CICS on operating system z/OS, the following SQUARE client is implemented with interface type "CICS with standard linkage calling convention". See ***Client Interface Types*** for more information.

Name	Type	Data Set	Description	Notes
COBSRVI	COBOL source code	EXP108.DCCO	Generate RPC service module for CICS with call interface.	4
SQRECLT	COBOL source code	EXP108.DCCO	An RPC client application calling the remote procedure (RPC service) SQUARE.	2
SQREDFH	CICS CSD	EXP108.DCCO	CSD Definition for RPC client SQRECLT.	
SQREIGY	JCL	EXP108.DCCO	Job (JCL) to build the RPC client SQRECLT.	3
SQREMAP	CICS Map	EXP108.DCCO	CICS Map definition for RPC client SQRECLT.	
SQUARE	COBOL source code	EXP108.DCCO	Client interface object for IDL program SQUARE.	1
SQREMAP	COBOL copybook	EXP108.DICO	Description of input and output fields of map SQREMAP.	
SQUARE	COBOL copybook	EXP108.DICO	Client interface object copybook for IDL program SQUARE.	1

Notes:

1. Under z/OS, client interface objects are delivered with the installation; under UNIX and Windows, generate these objects with the Designer.
2. Application
   a. built according to the client-side build instructions, see ***Using the COBOL Wrapper for CICS with Call Interfaces (z/OS and z/VSE)***.
   b. associated with IDL file *example.idl*.
   c. SQRECLT uses CICS Map definition SQREMAP.
   d. SQRECLT and client interface object SQUARE are linked together.
   e. SQRECLT installed as single CICS program.
3. Adapt the JCL to your needs.
4. See ***Generation and Usage of Generic RPC Service Module COBSRVI***.

For more information, refer to the readme file in EntireX directory *examples/RPC/CobolClient/zos-CICS/CallInterface* under UNIX or Windows.

SQUARE Client using DFHCOMMAREA

For CICS on operating system z/OS, the following SQUARE client is implemented with interface type "CICS with DFHCOMMAREA calling convention". See *Client Interface Types* for more information.

Name	Type	Data Set	Description	Notes
SQRE1DFH	CICS CSD	EXP108.DCCO	CSD Definition for RPC client SQRE1CLT.	
SQREI1GY	JCL	EXP108.DCCO	Job (JCL) to build the RPC client SQRE1CLT.	2
SQRE1MAP	CICS Map	EXP108.DCCO	CICS Map definition for RPC clients SQRE1CLT.	
SQRE1	COBOL source code	EXP108.DCCO	Client interface object for IDL program SQRE1, alias of SQUARE.	1
SQRE1CLT	COBOL source code	EXP108.DCCO	An RPC client application calling the remote procedure (RPC service) SQUARE.	3
SQRE1MAP	COBOL copybook	EXP108.DICO	Description of input and output fields of map SQRE1MAP.	
SQRE1	COBOL copybook	EXP108.DICO	Client interface object copybook for IDL program SQRE1, alias of SQUARE.	1

Notes:

1. Under z/OS, client interface objects are delivered with the installation; under UNIX and Windows, generate these objects with the Designer.
2. Adapt the JCL to your needs.
3. Application
   a. built according to the client-side build instructions, see *Using the COBOL Wrapper for CICS with DFHCOMMAREA Calling Convention (z/OS and z/VSE)*.
   b. associated with IDL *exampleWithPgmAlias.idl*.
   c. client interface object name SQRE1 different from remote procedure name SQUARE (RPC service).
   d. SQRE1CLT and client interface object SQRE1 installed as separate CICS programs.

For more information, refer to the readme file in EntireX directory *examples/RPC/CobolClient/zos-CICS/DFHCOMMAREA* under UNIX or Windows.

Basic RPC Server Examples - CALC, SQUARE

CALC Server

For CICS under operating system z/OS, the CALC server is built with COBOL Wrapper "CICS with DFHCOMMAREA calling convention" interface type. See *Server Interface Types* for more information.

Name	Type	Data Set	Description	Notes
CALC	COBOL source code	EXP108.DVCO	A server application providing the remote procedure CALC (RPC service), with associated *example.idl*.	1
CALCDFH	CICS CSD	EXP108.DVCO	CSD Definition for remote procedure CALC (RPC service).	
CALCIGY	JCL	EXP108.DVCO	Job (JCL) to build the remote procedure CALC (RPC service).	2

Notes:

1. Application built according to the server-side build instructions, see *Using the COBOL Wrapper for CICS with DFHCOMMAREA Calling Convention (z/OS and z/VSE)*.
2. Adapt the JCL to your needs.

For more information, refer to the readme file in EntireX directory *examples/RPC/CobolServer/zos-CICS/DFHCOMMAREA* under UNIX or Windows.

SQUARE Server

For CICS under operating system z/OS, the SQUARE server is built with COBOL Wrapper "CICS with DFHCOMMAREA calling convention" interface type. See *Client Interface Types* for more information.

Name	Type	Data Set	Description	Notes
SQREDFH	CICS CSD	EXP108.DVCO	CSD Definition for remote procedure SQUARE (RPC service).	
SQREIGY	JCL	EXP108.DVCO	Job (JCL) to build the remote procedure SQUARE (RPC service).	2
SQUARE	COBOL source code	EXP108.DVCO	A server application providing the remote procedure SQUARE (RPC service), with associated *example.idl*.	1

Notes:

1. Application built according to the server-side build instructions, see ***Using the COBOL Wrapper for CICS with DFHCOMMAREA Calling Convention (z/OS and z/VSE)***.
2. Adapt the JCL to your needs.

For more information, refer to the readme file in EntireX directory *examples/RPC/CobolServer/zos-CICS/DFHCOMMAREA* under UNIX or Windows.

Advanced CICS Channel Container RPC Server Example

For CICS on operating system z/OS, the TWOC server is built with COBOL Wrapper "CICS with Channel Container calling convention" interface type. See ***Server Interface Types*** for more information.

Name	Type	Data Set	Description	Notes
TWOC	COBOL source code	EXP108.DVCO	A server application providing the remote procedure TWOC (RPC service), with associated *CICSChannelContainer.idl*.	1
TWOCDFH	CICS CSD	EXP108.DVCO	CSD Definition for remote procedure TWOC (RPC service).	
TWOCIGY	JCL	EXP108.DVCO	Job (JCL) to build remote procedure TWOC (RPC service).	2

1. Application built according to the server-side build instructions. See ***Using the COBOL Wrapper for CICS with Channel Container Calling Convention (z/OS)***.
2. Adapt the JCL to your needs.

For more information, see the readme file in EntireX directory *examples/RPC/CobolServer/zos-CICS/ChannelContainer* under UNIX or Windows.

21 Client and Server Examples for z/OS IMS BMP

The delivered *client* examples for z/OS batch can be used as a basis for use in BMP mode, but they have to be adapted.

The delivered *server* examples for z/OS batch can also be used in BMP mode. See ***Client and Server Examples for z/OS Batch***. Using IMS PCB pointers to access IMS databases in this context is described in ***IMS PCB Pointer IDL Rules*** under ***Using the COBOL Wrapper for IMS BMP (z/OS)***.

22 Server Examples for z/OS IMS MPP

This chapter describes the RPC examples provided. After installation of the *EntireX Development Tools* package, all examples here can be found in the EntireX directory *examples/RPC*. They are also available as a z/OS data set, see *Installing RPC Examples for z/OS*.

CALC Server

The CALC server is an IMS message processing program (MPP) for the TP system IMS under operating system z/OS. It is accessible with IMS Connect using *RPC Server for IMS Connect* or the *EntireX Adapter*.

Name	Type	Data Set	Description	Notes
CALC	COBOL source code	EXP108.MVCO	A server application providing the remote procedure CALC (RPC service) with associated *example.idl*.	
CALCIGY	JCL	EXP108.MVCO	Job (JCL) to build the remote procedure CALC (RPC service).	1
CALCSTG	IMS definition	EXP108.MVCO	IMS first stage generation definition for TNCALCP transaction.	1

Notes:

1. Adapt the JCL to your needs.

For more information, refer to the readme file in EntireX directory *examples/RPC/CobolServer/zosIMS-MPP* under UNIX or Windows.

SQUARE Server

The SQUARE server is an IMS message processing program (MPP) for the TP system IMS under operating system z/OS. It is accessible with IMS Connect using the *RPC Server for IMS Connect* or the *EntireX Adapter*.

Name	Type	Data Set	Description	Notes
SQUARE	COBOL source code	EXP108.MVCO	A server application providing the remote procedure SQUARE (RPC service), with associated *example.idl*.	
SQREIGY	JCL	EXP108.MVCO	Job (JCL) to build the remote procedure SQUARE (RPC service).	1
SQRESTG	IMS definition	EXP108.MVCO	IMS first stage generation definition for TNSQREP transaction.	1

Notes:

1. Adapt the JCL to your needs.

For more information, refer to the readme file in EntireX directory *examples/RPC/CobolServer/zosIMS-MPP* under UNIX or Windows.

23 Client and Server Examples for BS2000

This chapter describes the RPC examples provided. After installation of the *EntireX Development Tools* package, all examples here can be found in the EntireX directory *examples/RPC*. The basic RPC server example *CALC* is also delivered on BS2000 in the LMS library EXP103.COBS.

Basic RPC Client Examples - CALC, SQUARE

CALC

Element	Type	Comment	Notes
CREATE-CALC-CLIENT	J	S-procedure to generate the CALC COBOL sample client application. It makes use of RUN-COBOL-COMPILER and BIND-CALC-CLIENT.	2
BIND-CALC-CLIENT	J	S-procedure to bind the CALC COBOL sample client application.	
RUN-COBOL-COMPILER	J	S-procedure to run the COBOL2000 / COBOL85 compiler.	2
RUN-CALC-CLIENT	J	S-procedure to run the CALC COBOL sample client application.	7
CALCCLT.COB	S	Main program source of the CALC COBOL example.	1
CALC.COB	S	COBOL RPC client interface object.	3
CALC	S	COBOL RPC interface copybook.	3
COBSRVI.COB	S	Generic RPC service.	4
ERXCOMM	S	Layout of the RPC communication area. See *The RPC Communication Area (Reference)*.	3
CLIENT-ADAPARM	S	Adabas ADALNK IDTNAME parameter required when using the NET transport method. It is shared by all clients.	5
CLIENT-INPARM-CALC	S	CALC client input parameters.	6

1. For applications built according to the client-side build instructions, see *Using the COBOL Wrapper for Batch (z/OS, BS2000, z/VSE and IBM i)*.
2. The default configuration expects a COBOL2000 environment. Depending on your installation it might be necessary to change the COMPILER parameter within the parameter declaration section of the procedures. The delivered procedures support both COBOL2000 and COBOL85 syntax.
3. Generate these objects with the Designer.
4. See *Generation and Usage of Generic RPC Service Module COBSRVI*.
5. Optional. If NET is chosen as transport method, specify the name of the ID table to which the broker is connected: ADALNK IDTNAME=ADA*xxxxx*.
6. Set up the BROKER-ID in one of two formats depending on the transport method:

- **TCP Transport Method**

```
ip:port:TCP
```

where *ip* is the address or DNS host name,
port is the port number that EntireX Broker is listening on, and
TCP is the protocol name

- **NET Transport Method**

```
ETBnnn:SVCmmm:NET
```

where *nnn* is the ID under which EntireX Broker is connected to the Adabas ID table,
mmm is the SVC number under which the Adabas ID table can be accessed, and
NET is the protocol name

7. Enter the following command to run the CALC COBOL sample client:

```
/CALL-PROCEDURE *LIB(LIB=EXP103.COBC,ELE=RUN-CALC-CLIENT)
```

For more information refer to the file *README.TXT* in the EntireX directory *examples/RPC/CobolServer/bs2000* under UNIX or Windows.

SQUARE

Element	Type	Comment	Notes
CREATE-SQUARE-CLIENT	J	S-procedure to generate the SQUARE COBOL sample client application. It makes use of RUN-COBOL-COMPILER and BIND-SQUARE-CLIENT.	2
BIND-SQUARE-CLIENT	J	S-procedure to bind the SQUARE COBOL sample client application.	
RUN-COBOL-COMPILER	J	S-procedure to run the COBOL2000 / COBOL85 compiler.	2
RUN-SQUARE-CLIENT	J	S-procedure to run the SQUARE COBOL sample client application.	7
SQRECLT.COB	S	Main program source of the SQUARE COBOL example.	1
SQUARE.COB	S	COBOL RPC client interface object.	3
SQUARE	S	COBOL RPC interface copybook.	3
COBSRVI.COB	S	Generic RPC service.	4
ERXCOMM	S	Layout of the RPC communication area. See *The RPC Communication Area (Reference)*.	3
CLIENT-ADAPARM	S	Adabas ADALNK IDTNAME parameter required when using the NET transport method. It is shared by all clients.	5

Element	Type	Comment	Notes
CLIENT-INPARM-SQUARE	S	SQUARE client input parameters.	6

1. For applications built according to the client-side build instructions, see ***Using the COBOL Wrapper for Batch (z/OS, BS2000, z/VSE and IBM i)***.
2. The default configuration expects a COBOL2000 environment. Depending on your installation it might be necessary to change the COMPILER parameter within the parameter declaration section of the procedures. The delivered procedures support both COBOL2000 and COBOL85 syntax.
3. Generate these objects with the Designer.
4. See ***Generation and Usage of Generic RPC Service Module COBSRVI***.
5. Optional. If NET is chosen as transport method, specify the name of the ID table to which the broker is connected: ADALNK IDTNAME=ADA*xxxxx*.
6. Set up the BROKERID in one of two formats depending on the transport method:
   - **TCP Transport Method**

     ```
     ip:port:TCP
     ```

     where *ip* is the address or DNS host name,
     port is the port number that EntireX Broker is listening on, and
     TCP is the protocol name
   - **NET Transport Method**

     ```
     ETBnnn:SVCmmm:NET
     ```

     where *nnn* is the ID under which EntireX Broker is connected to the Adabas ID table,
     mmm is the SVC number under which the Adabas ID table can be accessed, and
     NET is the protocol name
7. Enter the following command to run the SQUARE COBOL sample client:

```
/CALL-PROCEDURE *LIB(LIB=EXP103.COBC,ELE=RUN-SQUARE-CLIENT)
```

For more information refer to the file *README.TXT* in the EntireX directory *examples/RPC/CobolServer/bs2000* under UNIX or Windows.

Basic RPC Server Examples - CALC, SQUARE

CALC

Element	Type	Comment	Notes
CREATE-CALC-SERVER	J	S-procedure to generate the CALC COBOL example server. It makes use of RUN-COBOL-COMPILER.	2
RUN-COBOL-COMPILER	J	S-procedure to run the COBOL2000 / COBOL85 compiler.	2
CALC.COB	S	Server program source of CALC COBOL example.	1,3

1. For applications built according to the server-side build instructions, see ***Using the COBOL Wrapper for Batch (z/OS, BS2000 and IBM i)***.
2. The default configuration expects a COBOL2000 environment. Depending on your installation it might be necessary to change the COMPILER parameter within the parameter declaration section of the procedures. The delivered procedures support both COBOL2000 and COBOL85 syntax.
3. When executing:
   - make sure the RPC server runs as COBOL RPC server (refer to parameter *marshalling* in the RPC-CONFIG S-element in library *EXP103.JOBS*)
   - make sure that library *EXP103.COBS* is included as PROGRAM-LIB in the startup procedure START-RPC-SERVER

For more information refer to the file *README.TXT* in the EntireX directory *examples/RPC/CobolServer/bs2000* under UNIX or Windows.

SQUARE

Element	Type	Comment	Notes
CREATE-SQUARE-SERVER	J	S-procedure to generate the SQUARE COBOL example server. It makes use of RUN-COBOL-COMPILER.	2
RUN-COBOL-COMPILER	J	S-procedure to run the COBOL2000 / COBOL85 compiler.	2
SQUARE.COB	S	Server program source of SQUARE COBOL example.	1,3

1. For applications built according to the server-side build instructions, see ***Using the COBOL Wrapper for Batch (z/OS, BS2000 and IBM i)***.
2. The default configuration expects a COBOL2000 environment. Depending on your installation it might be necessary to change the COMPILER parameter within the parameter declaration section of the procedures. The delivered procedures support both COBOL2000 and COBOL85 syntax.

3. When executing:
   - make sure the RPC server runs as COBOL RPC server (refer to parameter *marshalling* in the RPC-CONFIG S-element in library *EXP103.JOBS*)
   - make sure that library *EXP103.COBS* is included as PROGRAM-LIB in the startup procedure START-RPC-SERVER

For more information refer to the file *README.TXT* in the EntireX directory *examples/RPC/CobolServer/bs2000* under UNIX or Windows.

24 Client and Server Examples for z/VSE Batch

This chapter describes the RPC examples provided. After installation of the *EntireX Development Tools* package, all examples here can be found in the EntireX directory *examples/RPC*.

Basic RPC Client Examples - CALC, SQUARE

CALC Client

The `CALC` client is built with COBOL Wrapper interface type "Batch with standard linkage calling convention". See ***Client Interface Types*** for more information.

Name	Type	Sublibrary[3]	Description	Notes
`README1.TXT`	Text document	`EXAMPLE.COBCLTB`	Client build instructions and description.	
`CALCCLT.C`	COBOL source code	`EXAMPLE.COBCLTB`	A client application calling the remote procedure (RPC service) CALC, with associated example.idl.	2
`CALC.C`	COBOL source code	`EXAMPLE.COBCLTB`	Client interface object for IDL program `CALC`.	1
`CALC.C`	COBOL copybook	`EXAMPLE.COBCPYB`	Client interface object copybook for IDL program CALC.	1
`ERXCOMM.C`	COBOL copybook	`EXAMPLE.COBCPY`	RPC Communication Area copybook.	1
`COBSRVIB.C`	COBOL source code	`EXAMPLE.COBCLTB`	Generic RPC Service for Batch.	4
`CALCCLT.J`	JCL	`EXAMPLE.COBCLTB`	Job control to build the RPC client `CALCCLT`.	3
`CALCRUN.J`	JCL	`EXAMPLE.COBCLTB`	Job control to execute the RPC client `CALCCLT`.	3

Notes:

1. Generate these objects with the Designer.
2. Application built according to the client-side build instructions, see ***Using the COBOL Wrapper for Batch (z/OS, BS2000, z/VSE and IBM i)***.
3. The delivered JCL requires the sources, copybooks etc. to be placed in the documented sublibrary. Adapt the JCL to your needs.
4. See ***Generation and Usage of Generic RPC Service Module COBSRVI***.

For more information refer to the file `README1.TXT` in EntireX directory *examples/RPC/CobolClient/vseBatch* under UNIX or Windows.

SQUARE Client

For batch under operating system z/VSE, the SQUARE client is built with COBOL Wrapper interface type "Batch with standard linkage calling convention". See *Client Interface Types* for more information.

Name	Type	Sublibrary (2)	Description	Notes
README1.TXT	Text document	EXAMPLE.COBCLTB	Client build instructions and description	
SQRECLT.C	COBOL source code	EXAMPLE.COBCLTB	A client application calling the remote procedure (RPC service) SQUARE, with associated example.idl.	1
SQUARE.C	COBOL source code	EXAMPLE.COBCLTB	Client interface object for IDL program SQUARE.	3
SQUARE.C	COBOL copybook	EXAMPLE.COBCPYB	Client interface object copybook for IDL program SQUARE.	3
ERXCOMM.C	COBOL copybook	EXAMPLE.COBCPY	RPC Communication Area copybook.	3
COBSRVIB.C	COBOL source code	EXAMPLE.COBCLTB	Generic RPC Service for Batch.	4
SQRECLT.J	JCL	EXAMPLE.COBCLTB	Job control to build the RPC client SQRECLT.	2
SQRERUN.J	JCL	EXAMPLE.COBCLTB	Job control to execute the RPC client SQRECLT.	2

Notes:

1. Application built according to the client-side build instructions, see *Using the COBOL Wrapper for Batch (z/OS, BS2000, z/VSE and IBM i)*.
2. The delivered JCL requires the sources, copybooks etc. to be placed in the documented sublibrary. Adapt the JCL to your needs.
3. Generate these objects with the Designer.
4. See *Generation and Usage of Generic RPC Service Module COBSRVI*.

For more information, refer to the file README1.TXT in EntireX directory *examples/RPC/CobolClient/vseBatch* under UNIX or Windows.

Basic RPC Server Examples - CALC, SQUARE

CALC Server

For batch under operating system z/VSE, the CALC server is built with COBOL Wrapper "Batch with standard linkage calling convention" interface type. See *Server Interface Types* for more information.

Name	Type	Sublibrary (2)	Description	Notes
`README1.TXT`	Text file	`EXAMPLE.COBSRVB`	CALC server build instructions and description	
`CALC.C`	COBOL source code	`EXAMPLE.COBSRVB`	A server application providing the remote procedure CALC (RPC service), with associated example.idl.	1
`CALC.J`	JCL	`EXAMPLE.COBSRVB`	Job control to build the remote procedure CALC (RPC service).	2

Notes:

1. Application built according to the server-side build instructions, see *Using the COBOL Wrapper for Batch (z/OS, BS2000 and IBM i)*.
2. The delivered JCL requires the sources, copybooks etc. to be placed in the documented sublibrary. Adapt the JCL to your needs.

For more information refer to the file `README1.TXT` in EntireX directory *examples/RPC/CobolServer/vseBatch* under UNIX or Windows.

SQUARE Server

For Batch on operating system z/VSE, the `SQUARE` server is built with COBOL Wrapper interface type "Batch with standard linkage calling convention". See *Client Interface Types* for more information.

Name	Type	Sublibrary[2]	Description	Notes
README1.TXT	Text file	EXAMPLE.COBSRVB	SQUARE server build instructions and description	
SQUARE.C	COBOL source code	EXAMPLE.COBSRVB	A server application providing the remote procedure SQUARE (RPC service), with associated example.idl	1
SQUARE.J	JCL	EXAMPLE.COBSRVB	Job control to build the remote procedure SQUARE (RPC service)	2

Notes:

1. Application built according to the server-side build instructions, see ***Using the COBOL Wrapper for Batch (z/OS, BS2000, z/VSE and IBM i)***.
2. The delivered JCL requires the sources, copybooks etc. to be placed in the documented sublibrary. Adapt the JCL to your needs.

For more information refer to the file README1.TXT in EntireX directory *examples/RPC/CobolServer/vseBatch* under UNIX or Windows.

25 Client and Server Examples for z/VSE CICS

This chapter describes the RPC examples provided. After installation of the *EntireX Development Tools* package, all examples here can be found in the EntireX directory *examples/RPC*. This chapter covers the following topics.

Basic RPC CALC Example

CALC Client using Call Interface (CALCCLT)

The `CALC` CICS client example `CALCCLT` is implemented with interface type "CICS with standard linkage calling convention". See ***Client Interface Types*** for more information.

Name	Type	Sublibrary [4]	Description	Notes
`README1.TXT`	Text file	`EXAMPLE.COBCLTC`	Client build instructions and description.	
`CALCCLT.C`	COBOL source code	`EXAMPLE.COBCLTC`	An RPC client application calling the remote procedure (RPC service) `CALC`.	1
`CALC.C`	COBOL source code	`EXAMPLE.COBCLTC`	Client interface object for IDL program `CALC`.	2
`CALC.C`	COBOL copybook	`EXAMPLE.COBCPYC`	Client interface object copybook for IDL program `CALC`.	2
`ERXCOMM.C`	COBOL copybook	`EXAMPLE.COBCPY`	RPC Communication Area copybook.	2
`COBSRVID.C`	COBOL source code	`EXAMPLE.COBCLTC`	Generic RPC Service module for CICS with call interface.	5
`CALCMAP.A`	CICS map	`EXAMPLE.COBCLTC`	CICS map for RPC client `CALCCLT`.	
`CALCMAP.C`	COBOL copybook	`EXAMPLE.COBCPYC`	Generated CICS Map COBOL Definitions.	3
`CALCLT.J`	JCL	`EXAMPLE.COBCLTC`	Job control to build the RPC client `CALCCLT`.	4
`CALCDFH.J`	JCL	`EXAMPLE.COBCLTC`	CICS CSD definitions job control for RPC client `CALCCLT`.	

Notes:

1. Built according to the client-side build instructions, see ***Using the COBOL Wrapper for CICS with Call Interfaces (z/OS and z/VSE)***.
2. Generate these objects with the Designer.
3. Generated from `CALCMAP.A` during execution of `CALCCLT.J`.
4. The delivered JCL requires the sources, copybooks etc. to be placed in the documented sublibrary. Adapt the JCL to your needs.

5. See *Generation and Usage of Generic RPC Service Module COBSRVI*.

For more information, refer to the `README1.TXT` file in EntireX directory *examples/RPC/CobolClient/vseCICS/Callinterface* under UNIX or Windows.

CALC Client using DFHACOMMAREA (CALC1CLT)

The `CALC` CICS client example `CALC1CLT` is implemented with interface type "CICS with DFHCOMMAREA calling convention". See *Client Interface Types* for more information.

Name	Type	Sublibrary [4]	Description	Notes
`README3.TXT`	Text file	`EXAMPLE.COBCLTC`	Client build instructions and description.	
`CALC1CLT.C`	COBOL source code	`EXAMPLE.COBCLTC`	An RPC client application calling the remote procedure (RPC service) `CALC`.	1
`CALC1.C`	COBOL source code	`EXAMPLE.COBCLTC`	Client interface object for IDL program `CALC`.	2
`CALC1.C`	COBOL copybook	`EXAMPLE.COBCPYC`	Client interface object copybook for IDL program `CALC`.	2
`ERXCOMM.C`	COBOL copybook	`EXAMPLE.COBCPY`	RPC Communication Area copybook.	2
`COBSRVIC.C`	COBOL source code	`EXAMPLE.COBCLTC`	Generic RPC Service with `EXEC CICS LINK` interface.	5
`CALC1MAP.A`	CICS map	`EXAMPLE.COBCLTC`	CICS map for RPC client `CALC1CLT`.	
`CALC1MAP.C`	COBOL copybook	`EXAMPLE.COBCPYC`	Generated CICS Map COBOL Definitions.	3
`CALC1CLT.J`	JCL	`EXAMPLE.COBCLTC`	Job control to build the RPC client `CALC1CLT`.	4
`CALC1DFH.J`	JCL	`EXAMPLE.COBCLTC`	CICS CSD definitions job control for RPC client `CALC1CLT`.	

Notes:

1. Built according to the client-side build instructions, see *Using the COBOL Wrapper for CICS with `DFHCOMMAREA` Calling Convention (z/OS and z/VSE)*.
2. Generate these objects with the Designer.
3. Generated from `CALC1MAP.A` during execution of `CALC1CLT.J`.
4. The delivered JCL requires the sources, copybooks etc. to be placed in the documented sublibrary. Adapt the JCL to your needs.
5. Built as `COBSRVI.PHASE` by `CALC1CLT.J`. See *Generation and Usage of Generic RPC Service Module COBSRVI*.

For more information, refer to the `README3.TXT` file in EntireX directory *examples/RPC/CobolClient/vseCICS/Callinterface* under UNIX or Windows.

CALC Server (CALC)

The CALC CICS server example is built with COBOL Wrapper interface type "CICS with DFHCOMMAREA calling convention". See *Server Interface Types* for more information.

Name	Type	Sublibrary (2)	Description	Notes
README1.TXT	Text file	EXAMPLE.COBSRVC	CALC server build instructions and description.	
CALC.C	COBOL source code	EXAMPLE.COBSRVC	A server application providing the remote procedure CALC (RPC service), with associated example.idl.	1
CALC.J	JCL	EXAMPLE.COBSRVC	Job control to build the remote procedure CALC (RPC service).	2
CALCDFH.J	JCL	EXAMPLE.COBSRVC	CICS CSD definitions job control for remote procedure CALC (RPC service).	

Notes:

1. Application built according to the server-side build instructions, see *Using the COBOL Wrapper for CICS with DFHCOMMAREA Calling Convention (z/OS and z/VSE)*.
2. The delivered JCL requires the sources, copybooks etc. to be placed in the documented sublibrary. Adapt the JCL to your needs.

For more information, refer to the README1.TXT file in EntireX directory *examples/RPC/CobolServer/vseCICS* under UNIX or Windows.

Basic RPC SQUARE Example

- SQUARE Client using Call Interface (SQRECLT)
- SQUARE Client using DFHACOMMAREA (SQRE1CLT)
- SQUARE Server (SQUARE)

SQUARE Client using Call Interface (SQRECLT)

The SQUARE CICS client example SQRECLT is implemented with interface type "CICS with standard linkage calling convention". See *Client Interface Types* for more information.

Name	Type	Sublibrary [4]	Description	Notes
README1.TXT	Text file	EXAMPLE.COBCLTC	Client build instructions and description.	
SQRECLT.C	COBOL source code	EXAMPLE.COBCLTC	An RPC client application calling the remote procedure (RPC service) SQUARE.	1
SQUARE.C	COBOL source code	EXAMPLE.COBCLTC	Client interface object for IDL program SQUARE.	2
SQUARE.C	COBOL copybook	EXAMPLE.COBCPYC	Client interface object copybook for IDL program SQUARE.	2
ERXCOMM.C	COBOL copybook	EXAMPLE.COBCPY	RPC Communication Area copybook.	2
COBSRVID.C	COBOL source code	EXAMPLE.COBCLTC	Generic RPC Service for CICS with call interface.	2,5
SQREMAP.A	CICS map	EXAMPLE.COBCLTC	CICS map for RPC client SQRECLT.	
SQREMAP.C	COBOL copybook	EXAMPLE.COBCPYC	Generated CICS Map COBOL Definitions.	3
SQRECLT.J	JCL	EXAMPLE.COBCLTC	Job control to build the RPC client SQRECLT.	4
SQREDFH.J	JCL	EXAMPLE.COBCLTC	CICS CSD definitions job control for RPC client SQRECLT.	

Notes:

1. Built according to the client-side build instructions, see ***Using the COBOL Wrapper for CICS with Call Interfaces (z/OS and z/VSE)***.
2. Generate these objects with the Designer.
3. Generated from SQREMAP.A during execution of SQRECLT.J.
4. The delivered JCL requires the sources, copybooks etc. to be placed in the documented sublibrary. Adapt the JCL to your needs.
5. See ***Generation and Usage of Generic RPC Service Module COBSRVI***.

For more information, refer to the README1.TXT file in EntireX directory *examples/RPC/CobolClient/vseCICS/Callinterface* under UNIX or Windows, or the downloaded example sublibrary EXAMPLE.COBCLTC.

SQUARE Client using DFHACOMMAREA (SQRE1CLT)

The SQUARE CICS client example SQRE1CLT is implemented with interface type "CICS with DFHCOMMAREA calling convention". See ***Client Interface Types*** for more information.

Name	Type	Sublibrary [4]	Description	Notes
README3.TXT	Text file	EXAMPLE.COBCLTC	Client build instructions and description.	
SQRE1CLT.C	COBOL source code	EXAMPLE.COBCLTC	An RPC client application calling the remote procedure (RPC service) SQUARE.	1
SQRE1.C	COBOL source code	EXAMPLE.COBCLTC	Client interface object for IDL program SQUARE.	2
SQRE1.C	COBOL copybook	EXAMPLE.COBCPYC	Client interface object copybook for IDL program SQUARE.	2
ERXCOMM.C	COBOL copybook	EXAMPLE.COBCPY	RPC Communication Area copybook.	2
COBSRVIC.C	COBOL source code	EXAMPLE.COBCLTC	Generic RPC Service.	2,5
SQRE1MAP.A	CICS map	EXAMPLE.COBCLTC	CICS map for RPC client SQRE1CLT.	
SQRE1MAP.C	COBOL copybook	EXAMPLE.COBCPYC	Generated CICS Map COBOL Definitions.	3
SQRE1CLT.J	JCL	EXAMPLE.COBCLTC	Job control to build the RPC client SQRE1CLT.	4
CALC1DFH.J	JCL	EXAMPLE.COBCLTC	CICS CSD definitions job control for RPC client SQRE1CLT.	

Notes:

1. Built according to the client-side build instructions, see ***Using the COBOL Wrapper for CICS with DFHCOMMAREA Calling Convention (z/OS and z/VSE)***.
2. Generate these objects with the Designer.
3. Generated from SQRE1MAP.A during execution of SQRE1CLT.J.
4. The delivered JCL requires the sources, copybooks etc. to be placed in the documented sublibrary. Adapt the JCL to your needs.
5. Built as COBSRVI.PHASE by SQRE1CLT.J. See ***Generation and Usage of Generic RPC Service Module COBSRVI***.

For more information, refer to the README3.TXT file in EntireX directory *examples/RPC/CobolClient/vseCICS/Callinterface* under UNIX or Windows, or the downloaded example sublibrary EXAMPLE.COBCLTC.

SQUARE Server (SQUARE)

The SQUARE CICS server example is built with COBOL Wrapper interface type "CICS with DFHCOMMAREA calling convention". See *Server Interface Types* for more information.

Name	Type	Sublibrary [2]	Description	Notes
README1.TXT	Text file	EXAMPLE.COBSRVC	CALC server build instructions and description.	
SQUARE.C	COBOL source code	EXAMPLE.COBSRVC	A server application providing the remote procedure SQUARE (RPC service), with associated example.idl.	1
SQUARE.J	JCL	EXAMPLE.COBSRVC	Job control to build the remote procedure SQUARE (RPC service).	2
SQREDFH.J	JCL	EXAMPLE.COBSRVC	CICS CSD definitions job control for remote procedure SQUARE (RPC service).	

Notes:

1. Application built according to the server-side build instructions, see *Using the COBOL Wrapper for CICS with DFHCOMMAREA Calling Convention (z/OS and z/VSE)*.
2. The delivered JCL requires the sources, copybooks etc. to be placed in the documented sublibrary. Adapt the JCL to your needs.

For more information, refer to the README1.TXT file in EntireX directory *examples/RPC/CobolServer/vseCICS* under UNIX or Windows.

VI

26 The RPC Communication Area (Reference)

The RPC communication area is used to specify parameters that are needed to communicate with the broker and are not specific to client interface objects. These are, for example, the Broker ID, client parameters such as user ID, password and the server address such as class/servername/service etc. See the tables below for a complete listing.

The RPC communication area is provided with the generated copybook `ERXCOMM` and optionally with its extension copybook `ERXVSTR` in the folder *include* for RPC client generation. See ***Generating COBOL Source Files from Software AG IDL Files***.

Copybook ERXCOMM

The `ERXCOMM` copybook enables an RPC client to specify and retrieve data for RPC communication. For usage refer to *`ERXCOMM` **Copybook*** under *Using the Generated Copybooks*.

RPC Communication Area Field	Explanation	Req/ Opt/ Auto	In/ Out	Notes
`ERXCOMM-HEADER`	Label.	-	-	-
`COMM-REQUEST`	Label.	-	-	-
`COMM-VERSION`	Version of RPC communication area. Possible values: 2000 or 4000.	R	I	7
`COMM-FUNCTION`	`LO` - log on to the Broker	O	I	1
	`LF` - log off from the Broker			1
	`OC` - open conversation			2
	`CE` - close conversation with commit			2
	`CB` - close conversation with backout			2
	`CT` - create Natural Security token			3
	`RC` - do reliable RPC commit			5
	`RR` - do reliable RPC rollback			5
	`RS` - get reliable status			5
	`EC` - end of conversation			-
`COMM-RETURN-CODE`	Message class and message code returned by COBOL Wrapper.	-	O	-
`COMM-MESSAGE-TEXT-EX`	Message text provided by COBOL Wrapper (long versions).	-	O	-
`COMM-MESSAGE-TEXT`	Deprecated. Use `COMM-MESSAGE-TEXT-EX` instead.	-	O	-
`ERXCOMM-AREA1`	Label.	-	-	-
`COMM-USERID`	Label.	-	-	-
`COMM-USERID1`	RPC user ID (8 characters) for RPC authentication.	O	I	3

RPC Communication Area Field	Explanation	Req/ Opt/ Auto	In/ Out	Notes
COMM-USERID2	Deprecated - do not use.	O	I	-
COMM-PASSWORD	RPC password (8 characters) for RPC authentication.	O	I	3
COMM-LIBRARY	RPC library name (8 characters) for RPC authentication or to log on to a specific library.	O	I	3,4
COMM-SECURITY-TOKEN-LENGTH	Length of Natural Security token.	A	I/O	6
COMM-SECURITY-TOKEN	Natural Security token.	A	I/O	6
COMM-IN-CONVERSATION	Control variable used internally by generic RPC services and client interface objects. If set to Y, RPC requests will use COMM-ETB-CONV-ID for conversationality.	A	I/O	6
COMM-IN-ACTIVE-UOW	Control variable used internally by generic RPC services and client interface objects for reliable RPC. If set to Y, RPC requests will use COMM-ETB-UOW-ID for reliability.	A	I/O	6
COMM-RELIABLE-STATE	Control variable used by the application to determine whether standard RPC requests or reliable RPC messages are used. Valid values: ' ' (blank) normal RPC requests A reliable RPC in AUTO-COMMIT mode C reliable RPC in CLIENT-COMMIT mode	R	I/O	5
COMM-RELIABLE-STATUS	Result of a "get reliable status" call to generic RPC services, see field COMM-FUNCTION above. Values correspond to broker ACI field UOWSTATUS.	A	O	5
COMM-KERNEL-SECURITY	Corresponds to Broker ACI field KERNELSECURITY.	O	I	1
COMM-CCSID	This field is available for operating system z/OS only. Specify the CCSID for the following tasks: ■ to convert the codepage of the long RPC password and RPC user ID; see *Using RPC Authentication (Natural Security, Impersonation, Integration Server)* ■ to tell the broker the encoding of your application data; see *Using Internationalization with the COBOL Wrapper*	O	I	-
COMM-ETB-BROKER-ID	Corresponds to Broker ACI field BROKER-ID.	R	I	-
COMM-ETB-SERVER-CLASS	Corresponds to Broker ACI field SERVER-CLASS.	R	I	-
COMM-ETB-SERVER-NAME	Corresponds to Broker ACI field SERVER-NAME.	R	I	-
COMM-ETB-SERVICE-NAME	Corresponds to Broker ACI field SERVICE.	R	I	-
COMM-ETB-USER-ID	Corresponds to Broker ACI field USER-ID.	O	I	1

RPC Communication Area Field	Explanation	Req/ Opt/ Auto	In/ Out	Notes
COMM-ETB-PASSWORD	Corresponds to Broker ACI field PASSWORD.	O	I	1
COMM-ETB-TOKEN	Corresponds to Broker ACI field TOKEN.	O	I/O	-
COMM-ETB-SECURITY-TOKEN	Corresponds to Broker ACI field SECURITY-TOKEN.	A	I/O	6
COMM-ETB-CONV-ID	Corresponds to Broker ACI field CONV-ID.	A	I/O	6
COMM-ETB-WAIT	Corresponds to Broker ACI field WAIT. Default: 60 seconds.	O	I	-
COMM-ETB-APIVERS	Corresponds to Broker ACI field API-VERSION.	A	I/O	6
COMM-ETB-UOW-ID	Corresponds to Broker ACI field UOWID.	O	I/O	5
COMM-ETB-STORE	Corresponds to Broker ACI field STORE.	O	I/O	5
COMM-ETB-PROGRAM-OFFSET	Fields are used internally to support the separate Application Monitoring documentation and for accounting purposes. See *Accounting in EntireX Broker* in the platform-specific Administration documentation.	A	I/O	6
COMM-ETB-LIBRARY-OFFSET		A	I/O	6
COMM-ETB-MESSAGE-ID	Corresponds to Broker ACI field MESSAGE-ID.	A	O	-
COMM-ETB-CORRELATION-ID	Corresponds to Broker ACI field CORRELATION-ID.	A	O	-
APPMON-SUPPORT	Fields are used internally to support the separate Application Monitoring documentation	A	I/O	6
APPMON-VERIFY		A	I/O	6
APPMON-TIMEVALUE		A	I/O	6
APPMON-TRANSPORT-BUFFER		A	I/O	6
APPMON-LEN-TRANSPORT-BUFFER		A	I/O	6
APPMON-RECEIVE-BUFFER		A	I/O	6
APPMON-LEN-RECEIVE-BUFFER		A	I/O	6
APPMON-LEN-DATA		A	I/O	6
APPMON-RETURN-CODE		A	I/O	6

Key

Req/Opt/Auto

Indicates for input fields whether they have to be given by the RPC client (required) or may be given by the user (optional). Fields marked with "Auto" are managed internally by the *Delivered Modules* themselves.

In/Out

Indicates whether the field is an input field (to be given by the RPC client) or an output field (returned to your RPC client).

Notes:

1. See *Using Broker Logon and Logoff*.

2. For RPC conversations. See *Using Conversational RPC*.
3. See *Using RPC Authentication (Natural Security, Impersonation, Integration Server)*.
4. If you are communicating with a non-secure Natural RPC Server you can set the Natural library. See *Using the COBOL Wrapper with Non-secure Natural RPC Server*.
5. See *Reliable RPC for COBOL Wrapper*.
6. Control variable used internally by generic RPC services and client interface objects. This means the field is managed internally by the *Delivered Modules* themselves.
7. - For a standard call interface client, see *Step 1: Declare and Initialize the RPC Communication Area* in section *Writing Standard Call Interface Clients*.
   - For an `EXEC CICS LINK` client refer to *Step 1: Declare IDL Structures and RPC Communication Area* in section *Writing `EXEC CICS LINK` Clients*.

Copybook ERXVSTR

The optional `ERXVSTR` copybook is an extension to the `ERXCOMM` copybook. It enables an RPC client to specify long data strings (e.g. passwords). For usage see *`ERXVSTR` Copybook* under *Using the Generated Copybooks*.

This table describes the fields in the RPC Variable String Area.

RPC Variable String Area Field	Explanation	Req/ Opt/ Auto	In/ Out	Notes
`ERXCOMM-AREA2`	Label.	-	-	-
`COMM-STRING-HEADER`	Label.	-	-	-
`COMM-ETB-PASSWORD-OFFSET`		O	I	1
`COMM-ETB-PASSWORD-LENGTH`		O	I	1
`COMM-RPC-USERID-OFFSET`		O	I	2
`COMM-RPC-USERID-LENGTH`		O	I	2
`COMM-RPC-PASSWORD-OFFSET`		O	I	2
`COMM-RPC-PASSWORD-LENGTH`		O	I	2
`COMM-RPC-LIBRARY-OFFSET`		O	I	2
`COMM-RPC-LIBRARY-LENGTH`		O	I	2
`COMM-STRING-AREA`		O	I	1,2

Notes:

1. See *Using Broker Logon and Logoff*.

2. See *Using RPC Authentication (Natural Security, Impersonation, Integration Server)*.

27 Delivered Modules

This chapter covers the following topics:

Delivered Modules for z/OS

Module	Data Set	Description	Notes
COBSRVI	EXP108.SRCE	CICS generic RPC services with EXEC CICS LINK interface.	2
COBDFH	EXP108.SRCE	CICS CSD definitions of CICS generic RPC services COBSRVI with EXEC CICS LINK interface.	2
ERXCOMM	EXP108.INCL	RPC communication area.	1
ERXVSTR	EXP108.INCL	RPC communication area extension copybook.	4
ERXRCSRV	EXP108.SRCE	C main module for application errors.	3
ERXRCSRV	EXP108.LD00	Ready-to-use ERXRCSRV module for application errors.	3
EXPCSRVI	EXX108.JOBS	JCL to compile the CICS generic RPC service module COBSRVI with EXEC CICS LINK interface.	2

- **EXP108.INCL**
  The Generic RPC *include* data set may be delivered as a patch with a different name EXP108.IN*nn*, where *nn* is the patch level number. Make sure you install the highest patch level available. The data set is required to SYSLIB input for the COBOL compiler.
- **EXP108.SRCE**
  The Generic RPC *source* data set may be delivered as a patch with a different name EXP108.S0*nn*, where *nn* is the patch level number. Make sure you install the highest patch level available. The data set is required to SYSLIB input for the COBOL compiler.

Notes:

1. The ERXCOMM copybook enables an RPC client to specify and retrieve data for RPC communication. For usage refer to *ERXCOMM* ***Copybook*** under *Using the Generated Copybooks*.
2. See ***Generation and Usage of Generic RPC Service Module COBSRVI***.
3. See *Using RETURN-CODE Special Register (COBOL only)* under *Scenarios and Programmer Information* in the RPC Server for CICS documentation.
4. The optional ERXVSTR copybook is an extension to the ERXCOMM copybook. It enables an RPC client to specify long data strings (e.g. passwords). For usage see *ERXVSTR* ***Copybook*** under *Using the Generated Copybooks*.

Delivered Modules for z/VSE

File	Sublibrary	Description	Notes
ERXCOMM	EXP960	RPC Communication area.	1,3
COBSRVIB.C	EXP960	Batch generic RPC services with call interface (source).	2,3
COBSRVIB.OBJ	EXP960	Batch generic RPC services with call interface (object).	2,3
COBSRVIC.C	EXP960	CICS generic RPC services with EXEC CICS LINK interface (source).	2,3
COBSRVIC.OBJ	EXP960	CICS generic RPC services with EXEC CICS LINK interface (object).	2,3
COBSRVID.C	EXP960	CICS generic RPC services with call interface (source).	2,3
COBSRVID.OBJ	EXP960	CICS generic RPC services with call interface (object).	2,3

Notes:

1. The ERXCOMM copybook enables an RPC client to specify and retrieve data for RPC communication. For usage refer to *ERXCOMM Copybook* under *Using the Generated Copybooks*.
2. See *Generation and Usage of Generic RPC Service Module COBSRVI*.
3. Do not use the modules delivered with your z/VSE installation. Use the modules generated by the Designer instead. Refer to *Generating COBOL Source Files from Software AG IDL Files*.

Delivered Modules for BS2000

Module	Data Set	Description	Notes
ERXCOMM	EXP103.COBC	RPC communication area.	1,3
COBSRVI.COB	EXP103.COBC	Batch generic RPC services with call interface.	2, 3

Notes:

1. The ERXCOMM copybook enables an RPC client to specify and retrieve data for RPC communication. For usage refer to *ERXCOMM Copybook* under *Using the Generated Copybooks*.
2. See *Generation and Usage of Generic RPC Service Module COBSRVI*.
3. Do not use the modules delivered with your BS2000 installation. Use the modules generated by the Designer instead. Refer to *Generating COBOL Source Files from Software AG IDL Files*.