webMethods EntireX

EntireX COBOL Wrapper

Version 9.10

April 2016

WEBMETHODS

Document ID: EXX-EEXXCOBWRAPPER-910-20181116

Table of Contents

EntireX COBOL Wrapper ... vii
1 About this Documentation ... 1
Document Conventions ... 2
Online Information and Support ... 2
Data Protection ... 3
2 Introduction to the COBOL Wrapper ... 5
Description ... 6
Generic RPC Services Module ... 7
COBOL Client Applications ... 7
COBOL Server Application ... 8
COBOL Server Interface Types ... 10
3 Using the COBOL Wrapper for the Client Side ... 17
Using the COBOL Wrapper for CICS with DFHCOMMAREA Calling Convention (z/OS and z/VSE) ... 19
Using the COBOL Wrapper for CICS with Call Interfaces (z/OS and z/VSE) ... 22
Using the COBOL Wrapper for Batch (z/OS, BS2000/OSD, z/VSE and IBM i) ... 24
Using the COBOL Wrapper for IMS (z/OS) ... 27
Using the COBOL Wrapper for IDMS/DC with Call Interfaces (z/OS) ... 28
Using the COBOL Wrapper for Micro Focus (UNIX and Windows) ... 30
4 Using the COBOL Wrapper for the Server Side ... 33
Using the COBOL Wrapper for CICS with DFHCOMMAREA Calling Convention (z/OS and z/VSE) ... 35
Using the COBOL Wrapper for CICS with Channel Container Calling Convention (z/OS) ... 38
Using the COBOL Wrapper for CICS with DFHCOMMAREA Large Buffer Interface (z/OS and z/VSE) ... 43
Using the COBOL Wrapper for Batch (z/OS, BS2000/OSD, z/VSE and IBM i) ... 46
Using the COBOL Wrapper for IMS BMP (z/OS) ... 50
Using the COBOL Wrapper for Micro Focus (UNIX and Windows) ... 54
5 Generating COBOL Source Files from Software AG IDL Files ... 57
Select an IDL File and Generate RPC Client or RPC Server ... 58
Generation Settings - Properties ... 62
Generation Settings - Preferences ... 73
6 Using the COBOL Wrapper in Command-line Mode ... 75
Command-line Options ... 76
Example Generating an RPC Client ... 79
Example Generating an RPC Server ... 80
Further Examples ... 80
7 Software AG IDL to COBOL Mapping ... 83
Mapping IDL Data Types to COBOL Data Types ... 84
Mapping Library Name and Alias ... 88
Mapping Program Name and Alias ... 89
Mapping Parameter Names ... 89

Mapping Fixed and Unbounded Arrays ... 90
Mapping Groups and Periodic Groups ... 91
Mapping Structures ... 91
Mapping the Direction Attributes In, Out, InOut ... 92
Mapping the ALIGNED Attribute ... 93
Calling Servers as Procedures or Functions ... 93
8 Writing Standard Call Interface Clients ... 95
Step 1: Declare and Initialize the RPC Communication Area ... 96
Step 2: Declare the Data Structures for RPC Stubs ... 96
Step 3: Required Settings in the RPC Communication Area ... 97
Step 4: Optional Settings in the RPC Communication Area ... 97
Step 5: Issue the RPC Request ... 98
Step 6: Examine the Error Code ... 98
9 Using the RPC Communication Area ... 99
Purpose of the RPC Communication Area ... 100
Using the RPC Communication Area with a Standard Call Interface ... 101
Using the RPC Communication Area with EXEC CICS LINK ... 103
10 Using the Generated Copybooks ... 107
IDL Interface Copybooks ... 108
COBINIT Copybook ... 109
COBEXIT Copybook ... 109
11 Using Broker Logon and Logoff ... 111
12 Using Conversational RPC ... 115
13 Using the COBOL Wrapper with Natural Security and Impersonation ... 119
14 Using the COBOL Wrapper with Non-secure Natural RPC Server ... 123
15 Reliable RPC for COBOL Wrapper ... 125
Introduction to Reliable RPC ... 126
Writing a Client ... 127
Writing a Server ... 132
Broker Configuration ... 132
16 Using the COBOL Wrapper with EntireX Security ... 133
17 Using SSL/TLS ... 135
z/OS ... 136
z/VSE ... 138
UNIX, Windows, BS2000/OSD ... 140
18 Client and Server Examples for Micro Focus (UNIX and Windows) ... 141
Basic RPC Client Examples - CALC, SQUARE ... 142
Basic RPC Server Examples - CALC, SQUARE ... 142
19 Client and Server Examples for z/OS Batch ... 145
Basic RPC Client Examples - CALC, SQUARE ... 146
Basic RPC Server Examples - CALC, SQUARE ... 148
20 Client and Server Examples for z/OS CICS ... 151
Basic RPC Client Examples - CALC, SQUARE ... 152
Basic RPC Server Examples - CALC, SQUARE ... 156
Advanced CICS Channel Container RPC Server Example ... 157

21 Client and Server Examples for z/OS IMS BMP 159
22 Server Examples for z/OS IMS MPP 161
CALC Server 162
SQUARE Server 162
23 Client and Server Examples for BS2000/OSD 165
Overview of Client and Server Examples for BS2000/OSD 166
Creating the Sample COBOL Client Programs 169
Creating the Sample COBOL Server Programs 170
Running the Sample COBOL Client Programs 170
24 Client and Server Examples for IBM i 173
Overview of Client and Server Examples for IBM i 174
Installing and Running the Client Examples for IBM i 175
Installing and Running the Server Examples for IBM i 175
25 Client and Server Examples for z/VSE Batch 177
Basic RPC Client Examples - CALC, SQUARE 178
Basic RPC Server Examples - CALC, SQUARE 180
26 Client and Server Examples for z/VSE CICS 183
Basic RPC CALC Example 184
Basic RPC SQUARE Example 186
27 COBOL Wrapper Reference 191
The RPC Communication Area (Reference) 192
Delivered Modules 196

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 Micro Focus, 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.
Reference	Provides reference material for the COBOL Wrapper.

1 About this Documentation

- Document Conventions 2
- Online Information and Support 2
- Data Protection 3

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

Software AG Documentation Website

You can find documentation on the Software AG Documentation website at **http://documentation.softwareag.com**. The site requires credentials for Software AG's Product Support site Empower. If you do not have Empower credentials, you must use the TECHcommunity website.

Software AG Empower Product Support Website

If you do not yet have an account for Empower, send an email to empower@softwareag.com with your name, company, and company email address and request an account.

Once you have an account, you can open Support Incidents online via the eService section of Empower at **https://empower.softwareag.com/**.

You can find product information on the Software AG Empower Product Support website at **https://empower.softwareag.com**.

To submit feature/enhancement requests, get information about product availability, and download products, go to **Products**.

To get information about fixes and to read early warnings, technical papers, and knowledge base articles, go to the **Knowledge Center**.

If you have any questions, you can find a local or toll-free number for your country in our Global Support Contact Directory at **https://empower.softwareag.com/public_directory.asp** and give us a call.

Software AG TECHcommunity

You can find documentation and other technical information on the Software AG TECHcommunity website at **http://techcommunity.softwareag.com**. You can:

- Access product documentation, if you have TECHcommunity credentials. If you do not, you will need to register and specify "Documentation" as an area of interest.
- Access articles, code samples, demos, and tutorials.
- Use the online discussion forums, moderated by Software AG professionals, to ask questions, discuss best practices, and learn how other customers are using Software AG technology.
- Link to external websites that discuss open standards and web technology.

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.

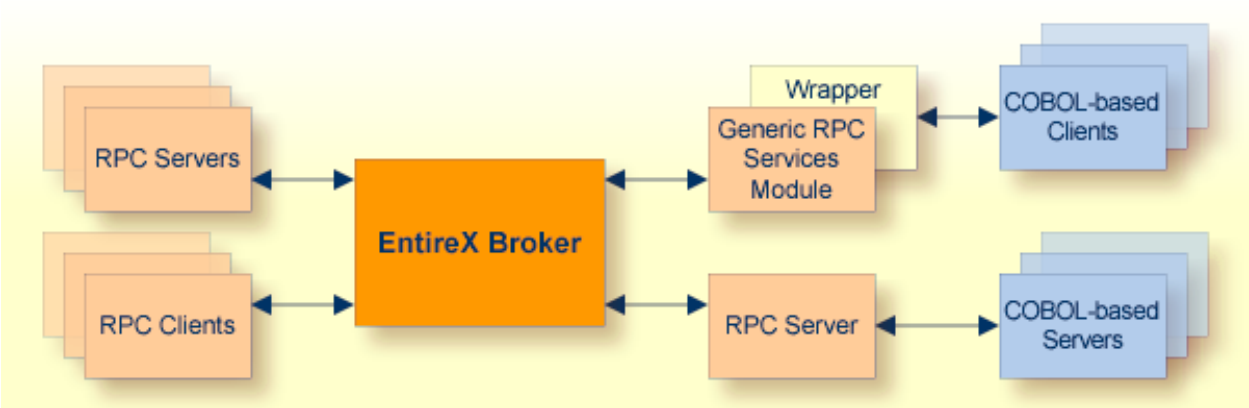
2 Introduction to the COBOL Wrapper

- Description 6
- Generic RPC Services Module 7
- COBOL Client Applications 7
- COBOL Server Application 8
- COBOL Server Interface Types 10

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 Workbench 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. Micro Focus, 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 an EntireX Workbench file with extension .svm or .cvm. See *Server Mapping Files for COBOL*.
- 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.

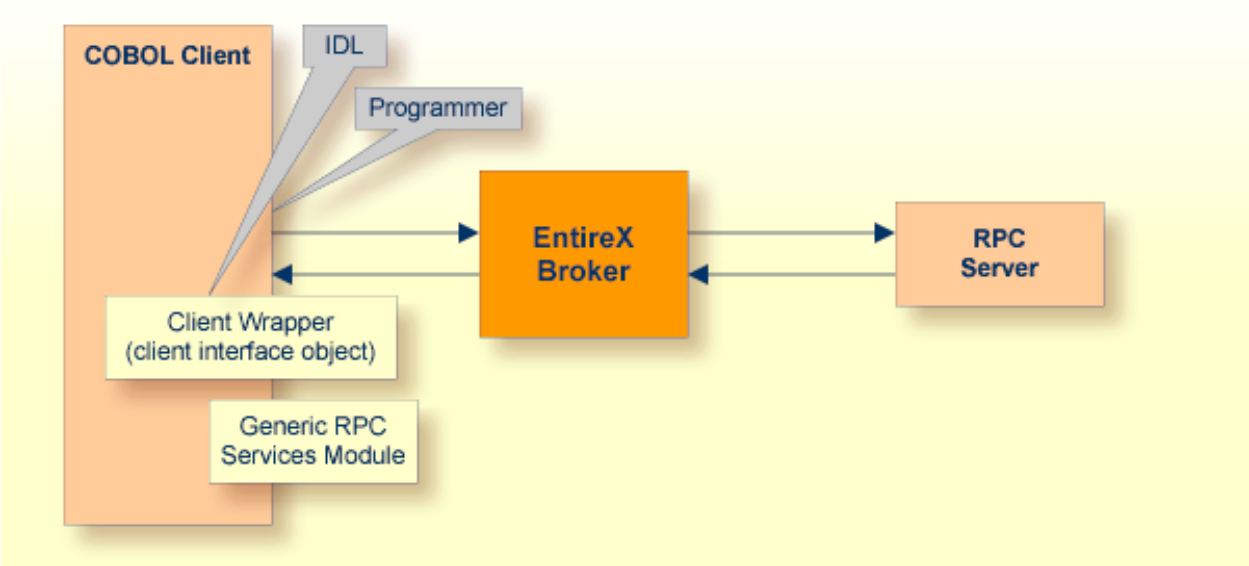
The generic RPC services module is used by RPC clients and contains the call to the broker stub, as well as other functions needed for RPC communication where an interface object is not needed, such as

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

For more information, see ***Generate 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.

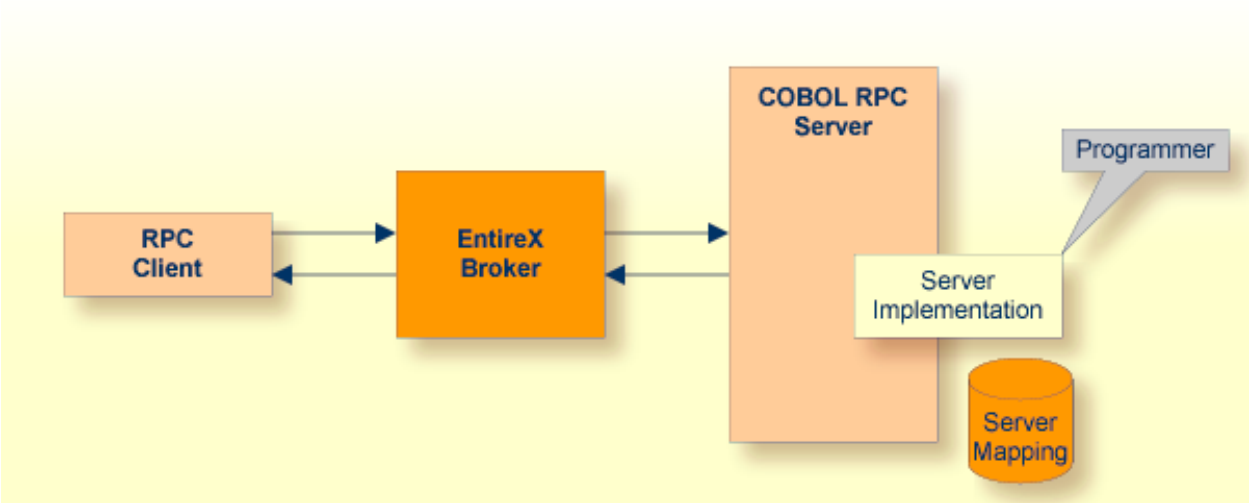


For more information, see ***Using the COBOL Wrapper for the Client Side***.

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. A server mapping file is an EntireX Workbench file with extension .svm or .cvm. See *When is a Server Mapping File Required?*

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. Client-side and server-side mapping files are handled differently. See *Server Mapping Files for COBOL* and ***Using the COBOL Wrapper for the Server Side***.


COBOL RPC
Server
Programmer
RPC
Client
EntireX
Broker
Server
Implementation
Server
Mapping

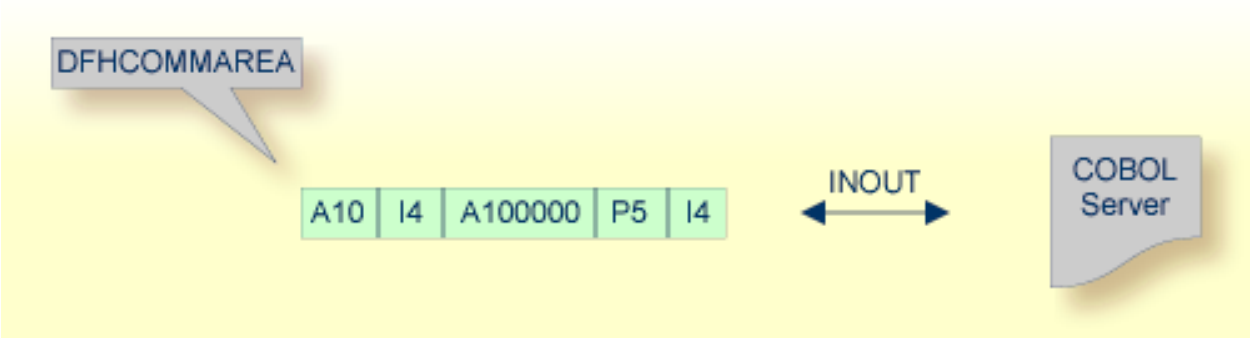
COBOL Server Interface Types

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
- Micro Focus with Standard Linkage Calling Convention
- Batch with Standard Linkage Calling Convention
- IMS BMP with Standard Linkage Calling Convention
- Compatibility between COBOL Interface Types and RPC Server

CICS with DFHCOMMAREA Calling Convention

CICS programs using the standard DFHCOMMAREA for parameter passing.



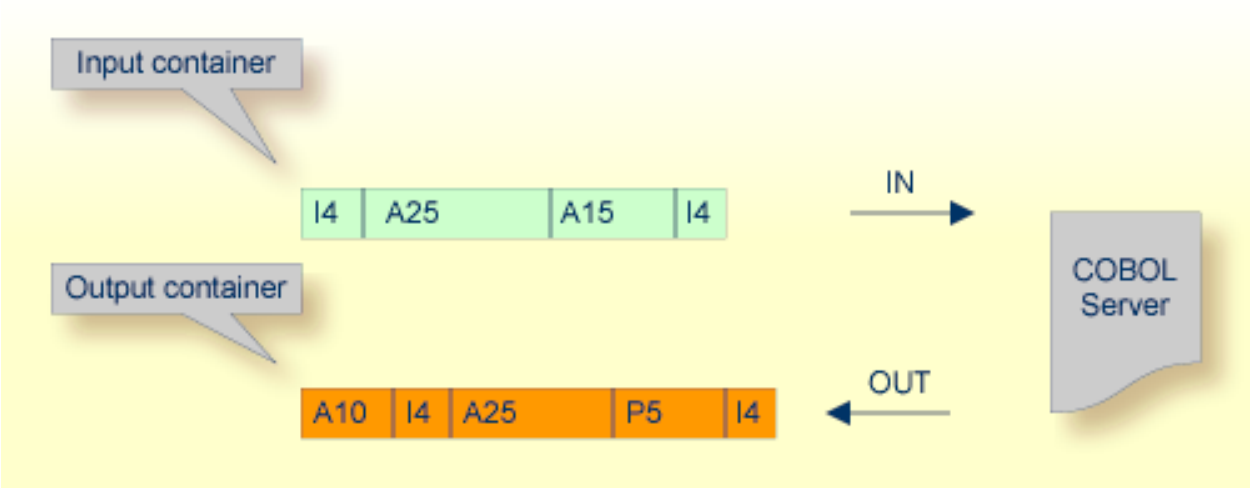
Technically, the generated COBOL server skeleton contains

- 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.



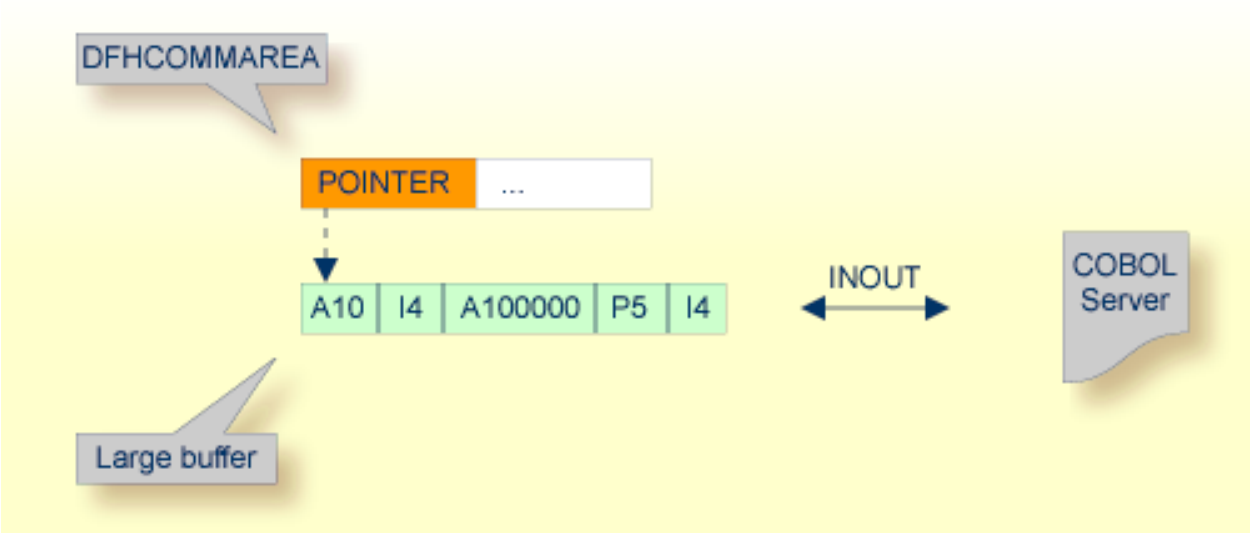
Technically, the generated COBOL server skeleton contains

- 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.



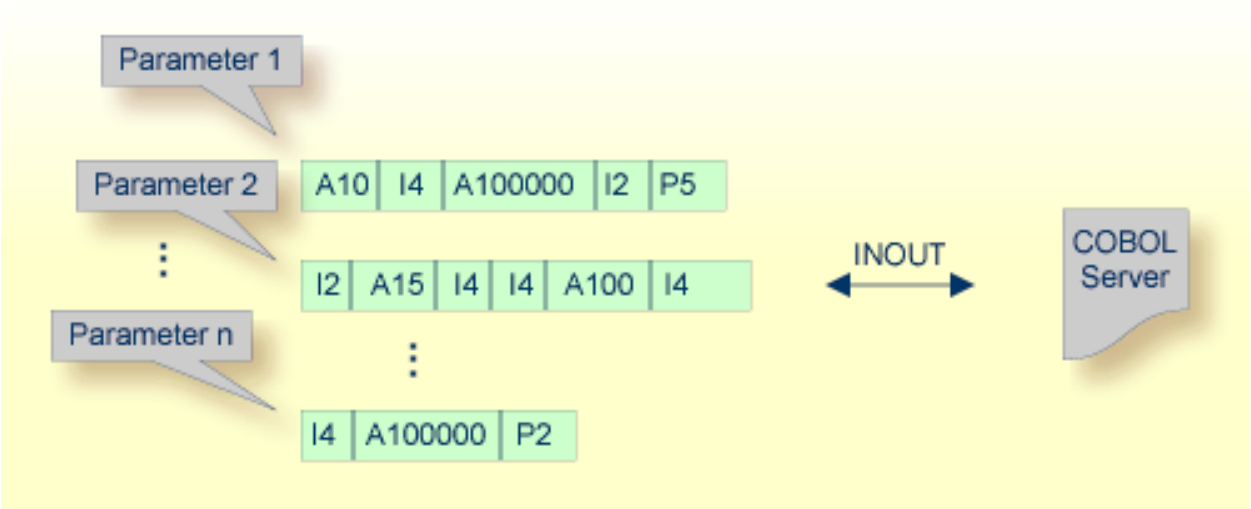
Technically,

- 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 COBOL's `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.

Micro Focus with Standard Linkage Calling Convention

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



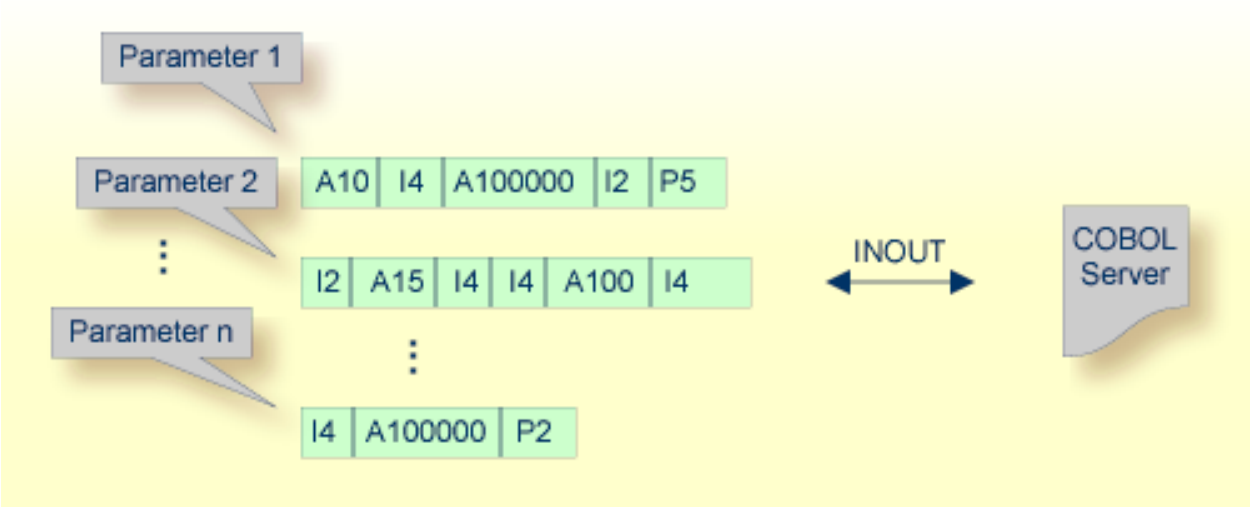
Technically, the generated COBOL server skeleton contains

- a parameter list `PROCEDURE DIVISION USING PARM1 PARM2 ... PARMn`
- 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.

Batch with Standard Linkage Calling Convention

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



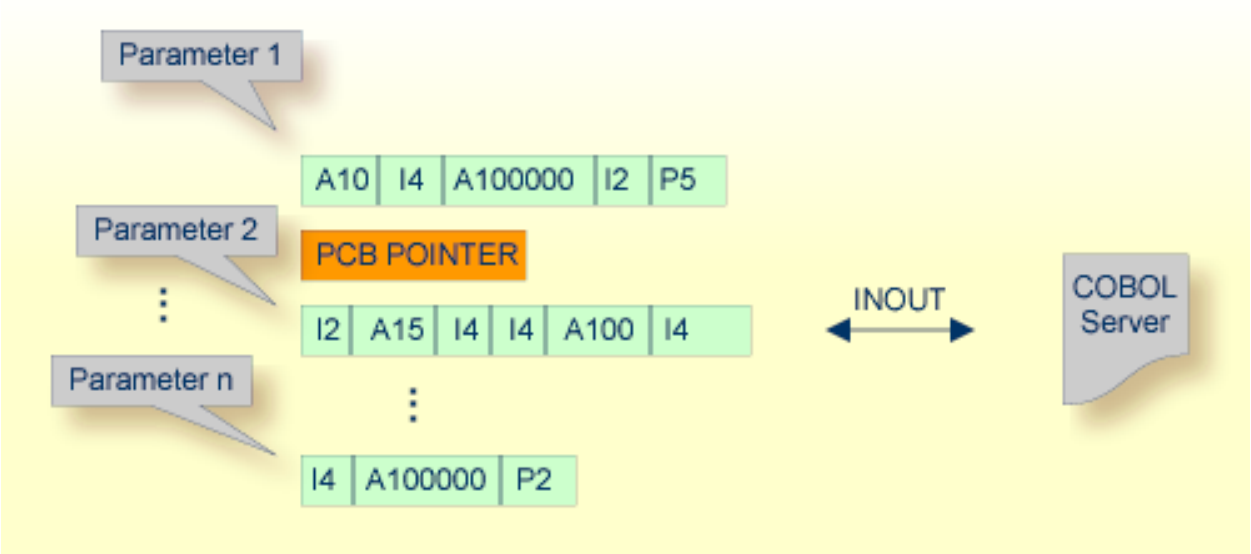
Technically, the generated COBOL server skeleton contains

- a parameter list `PROCEDURE DIVISION USING PARM1 PARM2 ... PARMn`
- 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 a server successfully, the RPC server used must support the interface type of the COBOL server. The table below gives an overview of possible combinations of an interface type and a supporting RPC server:

Interface Type	Supported by EntireX Component		Supported by EntireX Adapter	Supported by RPC Server								
				z/OS			UNIX/Windows			BS2000/ OSD	z/VSE	
				CICS	Batch	IMS	CICS ECI	Micro Focus	IMS Connect	Batch	CICS	Batch
CICS with `DFHCOMMAREA` Calling Convention	Extractor	Wrapper	x	x			x				x	
CICS with `DFHCOMMAREA` Large Buffer Interface	Extractor	Wrapper		x							x	
CICS with Channel Container Calling Convention	Extractor	Wrapper		x								
Batch with Standard Linkage Calling Convention	Extractor	Wrapper			x	x				x		x
Micro Focus with Standard Linkage Calling Convention	Extractor	Wrapper						x				
IMS BMP with Standard Linkage Calling Convention	Extractor	Wrapper				x						
IMS MPP Message Interface (IMS Connect)	Extractor		x						x			

3 Using the COBOL Wrapper for the Client Side

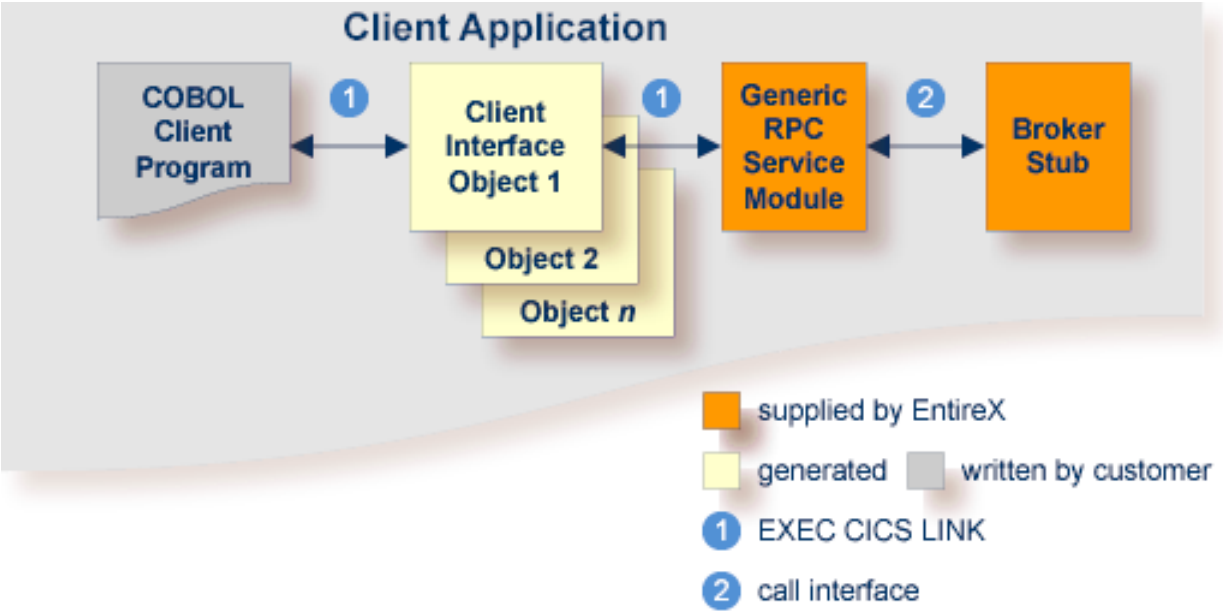
- Using the COBOL Wrapper for CICS with DFHCOMMAREA Calling Convention (z/OS and z/VSE) 19
- Using the COBOL Wrapper for CICS with Call Interfaces (z/OS and z/VSE) 22
- Using the COBOL Wrapper for Batch (z/OS, BS2000/OSD, z/VSE and IBM i) 24
- Using the COBOL Wrapper for IMS (z/OS) 27
- Using the COBOL Wrapper for IDMS/DC with Call Interfaces (z/OS) 28
- Using the COBOL Wrapper for Micro Focus (UNIX and Windows) 30

The COBOL Wrapper provides access to RPC-based components from COBOL applications and enables you to develop both clients and servers. This section 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 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 and the broker stub are linked together to a CICS program. The COBOL client program, every generated client interface object and the generic RPC services module together with the broker stub 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).
- The restriction of the 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.
- 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).

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. You may need to adapt the broker stub that supports the required transport (TCP, NET). See *Adapting the Used Broker Stub*.
4. Write your COBOL client program. See *Writing Applications with the COBOL Wrapper*, in particular the section *Using the RPC Communication Area with a Standard Call Interface*, and take into consideration the information given in *Software AG IDL to COBOL Mapping*.
5. 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.
6. 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.

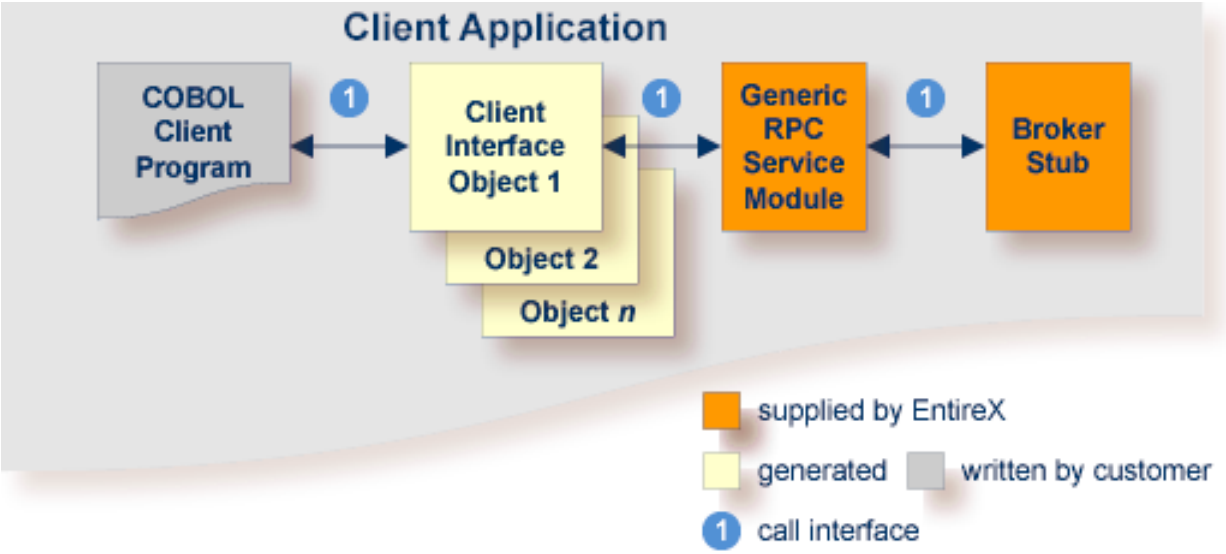
7. Install every client interface object, if required the CICS RPC service module COBSRVI and your COBOL client program as separate CICS programs.
8. Make sure the correct broker stub is used and can be called dynamically by the CICS generic RPC service module COBSRVI.

   - **z/OS**
     See the broker installation documentation and use a broker stub for CICS (for example CICSETB) from the common load library EXX910.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 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 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/OSD, z/VSE and IBM i)*.

Using a call interface within CICS may be useful if

- the restriction of the COMMAREA length (about 31 KB) prevents you from using the *Using the COBOL Wrapper for CICS with* `DFHCOMMAREA` *Calling Convention (z/OS and z/VSE)* scenario (see above)
- 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.

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 You may need to adapt the broker stub that supports the required transport (TCP, NET). See ***Adapting the Used Broker Stub***.

4 Write your COBOL client program. See ***Writing Applications with the COBOL Wrapper***, in particular the section ***Using the RPC Communication Area with a Standard Call Interface***, and take into consideration the information given in ***Software AG IDL to COBOL Mapping***.

5 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.

6 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.

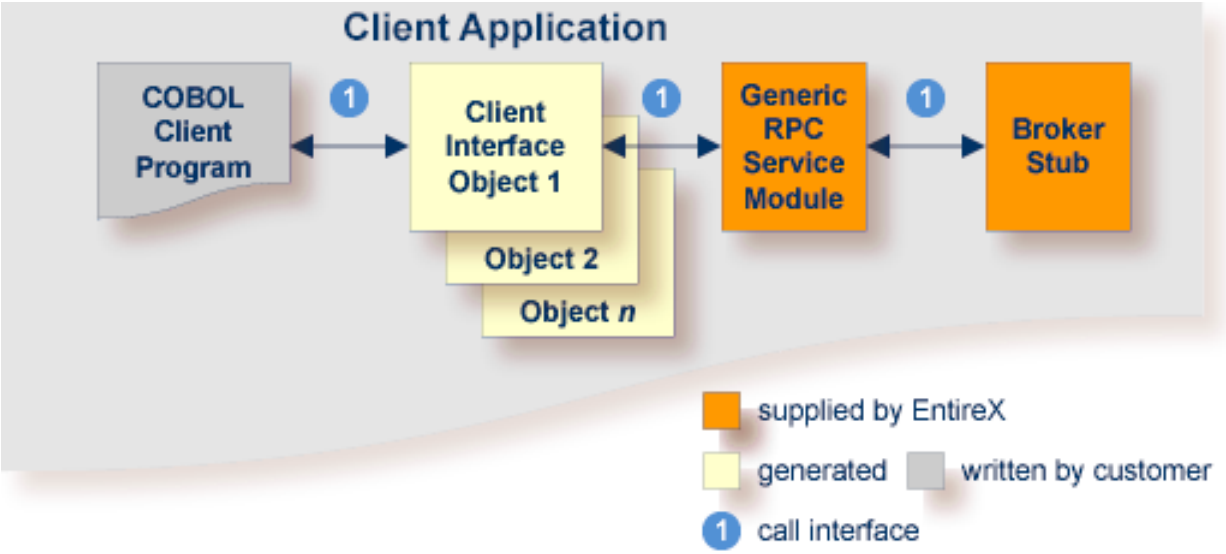
7 Install the client application within CICS.

8 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 EXX910.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/OSD, z/VSE and IBM i)

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



In this scenario, the COBOL client program, every generated client interface object, generic RPC services module and the broker stub are linked together to the client application.

Use the COBOL Wrapper for batch if you need to embed the client interface object into your application with a standard linkage calling convention.

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. You may need to adapt the broker stub that supports the required transport (TCP, NET). See *Adapting the Used Broker Stub*.
4. Write your COBOL client program. See *Writing Applications with the COBOL Wrapper*, in particular the section *Using the RPC Communication Area with a Standard Call Interface*, and take into consideration the information given in *Software AG IDL to COBOL Mapping*.
5. 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.

- **BS200/OSD**
  The IDL types U or UV require a compiler that supports COBOL data type NATIONAL. See *BS2000/OSD Prerequisites* for more information on supported compilers.
- **IBM i**
  - Use the command CRTCBLMOD (create COBOL module) and compile all modules above to ILE modules.
  - Use the IBM i compiler command with the options shown below:

```
CRTCBLMOD
OPTION(*NOMONOPRC) EXTDSOPT(*NODFRWRT) LINKLIT(*PRC)
```

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

6 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.
  To link the main program, use the following create program command with the options shown:

```
CRTPGM
   MODULE(*LIB/myapplication mystub1 mystub2 ..)
   BNDSRVPGM(EXX/EXA) ...
```

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

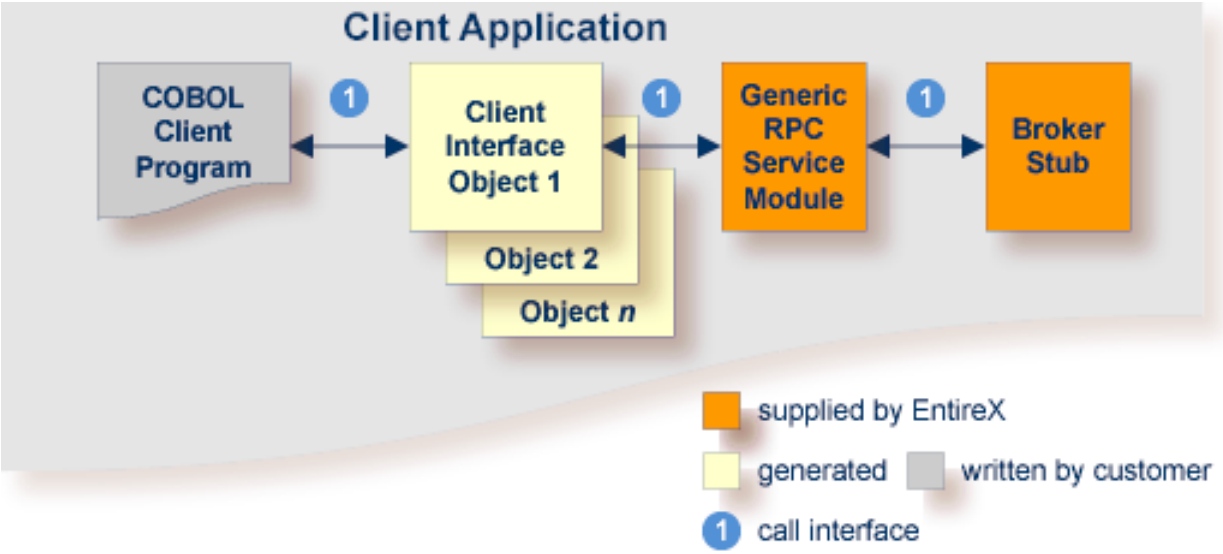
- **Other Platforms**
  Refer to your standard linker (binder) documentation.

7 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 EXX910.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 BKIMBTIA is located in the broker LMS load library.
- **IBM i**
  The broker stub EXA is located by default in the EntireX product library EXX.

Using the COBOL Wrapper for IMS (z/OS)

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



In this scenario, the COBOL client program, every generated client interface object, the generic RPC services module and the broker stub are linked together to the client application.

Use the COBOL Wrapper for IMS if you need to embed the client interface object into your IMS BMP or IMS MPP application with a standard linkage calling convention.

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. You may need to adapt the broker stub that supports the required transport (TCP, NET). See *Adapting the Used Broker Stub*.
4. Write your COBOL client program. See *Writing Applications with the COBOL Wrapper*, in particular the section *Using the RPC Communication Area with a Standard Call Interface*, and take into consideration the information given in *Software AG IDL to COBOL Mapping*.
5. 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.

6 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.

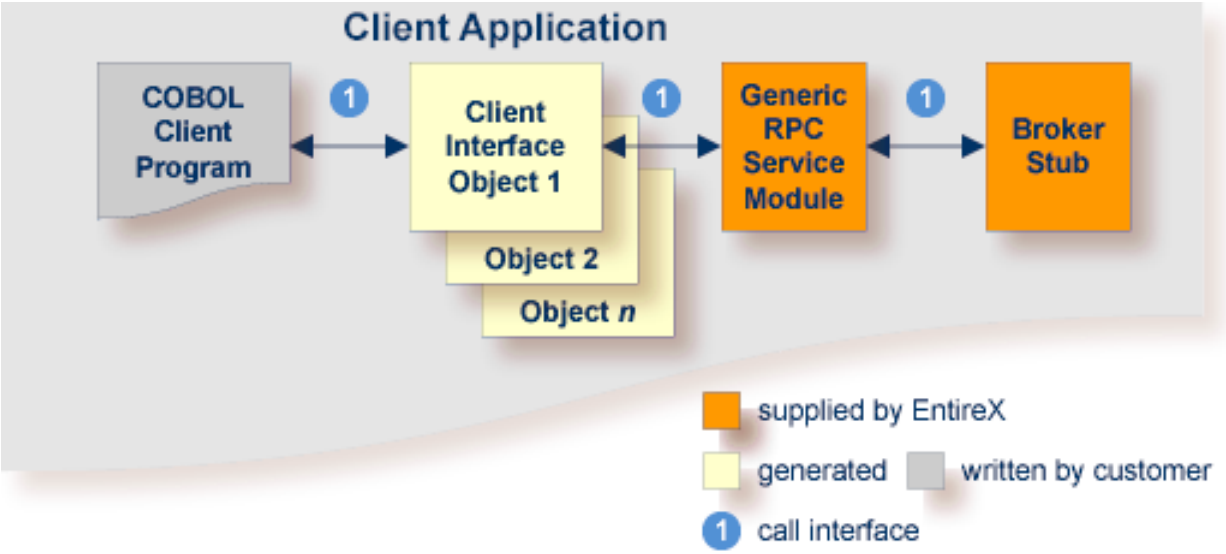
7 Make sure the correct broker stub is used and can be called dynamically. In the common load library EXX910.LOAD you can find broker stubs that can be used for

- 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/OSD, 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. You may need to adapt the broker stub that supports the required transport (TCP, NET). See *Adapting the Used Broker Stub*.
4. Write your COBOL client program. See *Writing Applications with the COBOL Wrapper*, in particular the section *Using the RPC Communication Area with a Standard Call Interface*, and take into consideration the information given in *Software AG IDL to COBOL Mapping*.
5. Using the IDMS/DC translator for COBOL provided with your IDMS/DC installation and a COBOL compiler supported by the COBOL Wrapper, translate and compile:

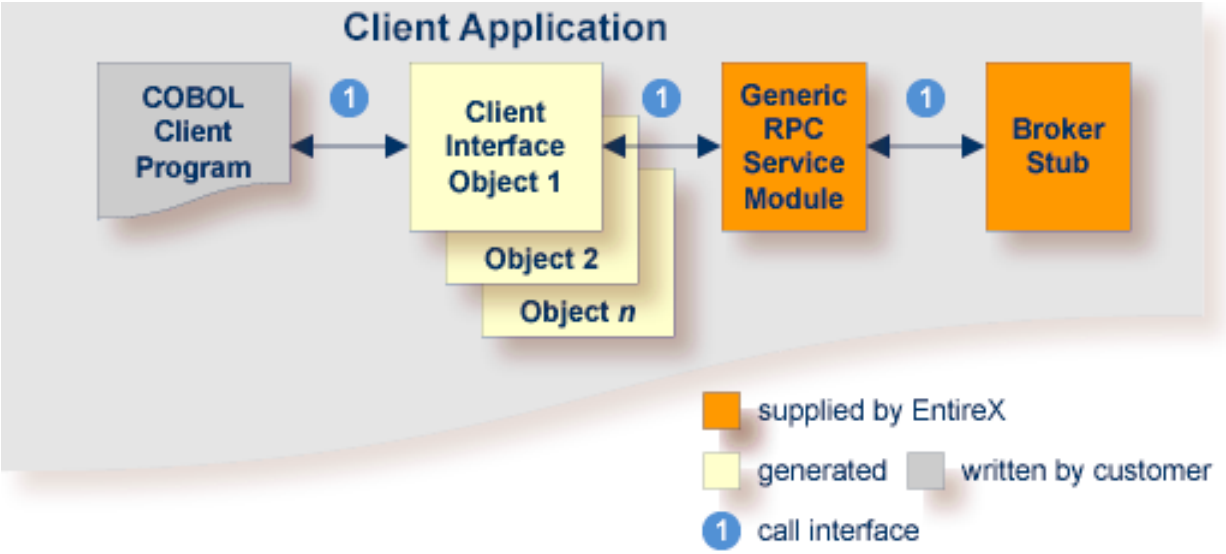
   - the generated client interface object(s)
   - if required, the generic RPC service module COBSRVI
   - your COBOL client program

   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.
6. Using the standard linker (binder) of the target platform, link (bind) all translated and compiled modules, and, if required, the broker stub, together to a IDMS/DC program, using the standard linker (binder) of the target platform.
7. Install the IDMS/DC program within IDMS/DC.
8. 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 EXX910.LOAD. See also *Administering Broker Stubs*.

Using the COBOL Wrapper for Micro Focus (UNIX and Windows)

This mode applies to UNIX and Windows.



In this scenario, the COBOL client program, every generated client interface object, generic RPC services module and the broker stub are linked together to the client application.

Use the COBOL Wrapper for Micro Focus if you need to embed the client interface object into your client application with a standard linkage calling convention.

To use the COBOL Wrapper for Micro Focus

1. Generate the client interface object(s) for the target operating system, for example "Windows", and use interface type "Micro Focus with standard linkage calling convention". See *Generating COBOL Source Files from Software AG IDL Files*. Check the option **Generic the 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. Import the modules into your Micro Focus IDE. The file names of the generated copybooks (see *Using the Generated Copybooks*) are derived from the IDL program name or its alias if present. The file names are the same as the file names of the client interface objects. They are distinguished by their extension, ".cbl" for the client interface objects and ".cpy" for the copybooks. If you import the generated copybooks and client interface objects into your Micro Focus development environment, take care the copybooks are accessed correctly by the compiler and not confused with the client interface objects. This may happen if you copy the

generated copybooks and the client interface objects into one directory. See your Micro Focus documentation for more information.

4 Write your COBOL client program. See ***Writing Applications with the COBOL Wrapper***, in particular the section ***Using the RPC Communication Area with a Standard Call Interface***, and take into consideration the information given in ***Software AG IDL to COBOL Mapping***.

5 Compile and link (bind) all modules together to an executable program:

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

For target operating system **UNIX** (i.e. the modules are generated for UNIX):

- The broker library from the EntireX UNIX installation must be linked to your client application, e.g. by defining the symbol "broker" as a linker option and linking the module *broker.o* from the EntireX UNIX installation.
- See your Micro Focus documentation for more information.

For target operating system **Windows** (i.e. the modules are generated for Windows):

- no additional compiler directives and linker options are required

6 Make sure the broker stub module can be called dynamically.

- **UNIX**
  The broker stub shared library or object *libbroker.so|sl* is accessible according to the rules of the UNIX system used, e.g. the directory of the library is defined in the `LD_LIBRARY_PATH` environment variable.;
- **Windows**
  The broker stub DLL *broker.dll* is accessible, for example with the `PATH` environment variable.

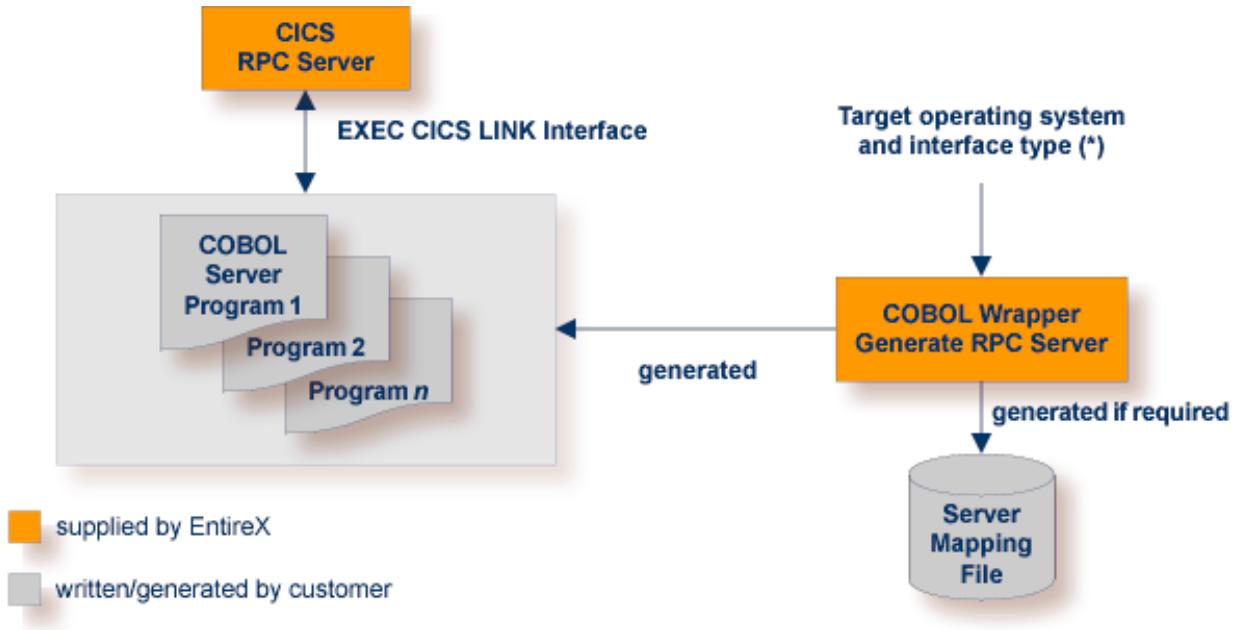
4 Using the COBOL Wrapper for the Server Side

- Using the COBOL Wrapper for CICS with DFHCOMMAREA Calling Convention (z/OS and z/VSE) 35
- Using the COBOL Wrapper for CICS with Channel Container Calling Convention (z/OS) 38
- Using the COBOL Wrapper for CICS with DFHCOMMAREA Large Buffer Interface (z/OS and z/VSE) 43
- Using the COBOL Wrapper for Batch (z/OS, BS2000/OSD, z/VSE and IBM i) 46
- Using the COBOL Wrapper for IMS BMP (z/OS) 50
- Using the COBOL Wrapper for Micro Focus (UNIX and Windows) 54

The COBOL Wrapper provides access to RPC-based components from COBOL applications and enables you to develop both clients and servers. This section 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 mode applies to z/OS and z/VSE. See also *COBOL Scenarios* under in the CICS RPC Server documentation.



(*) 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)*

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 an EntireX Workbench file with extension .svm or .cvm. See *Server Mapping Files for COBOL*.

   - *Server*-side mapping files (.svm): Deploy these to the RPC server. See *Deploying Server-side Mapping Files to the RPC Server* in the CICS RPC Server (z/OS, z/VSE, CICS ECI) sections of the documentation, except for CICS ECI connections with the webMethods EntireX Adapter, where you need to update your Adapter connection. See *Step 3: Select the Connection Type* in the Integration Server Wrapper documentation.
   - *Client*-side mapping files (.cvm): These 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 EntireX Workbench documentation) and re-generate the client interface objects. For connections with the webMethods EntireX Adapter you need to update your Adapter connection. See *Step 3: Select the Connection Type* in the Integration Server Wrapper documentation.

   See *How to Set the Type of Server Mapping Files* for how to define use of server-side or client-side mapping.
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 *Aborting RPC Server Customer Code and Returning Error to RPC Client* in the CICS RPC Server 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 CICS RPC server, EntireX Adapter, or CICS ECI RPC server:

   - Install your server(s) as separate CICS program(s).
   - If you are using a *server*-side mapping file, a concatenation of the `program-name` and the `library-name` given in the IDL is used to locate the server mapping file. See `program-definition` and `library-definition` under *Software AG IDL Grammar* in the IDL Editor documentation. Example: If a client performs an RPC request that is based on the IDL program name `CALC` and the IDL library `EXAMPLE`, the RPC server will dynamically try to locate logically the server mapping file `EXAMPLECALC` and execute the program with the

COBOL name defined in the server mapping. See ***Customize Automatically Generated Server Names***. If no corresponding program can be found, the access will fail.

- If you are using a *client*-side mapping file, the server mapping is taken from 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.
- If neither a server-side nor client-side mapping file is used - for example it is not required 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.

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

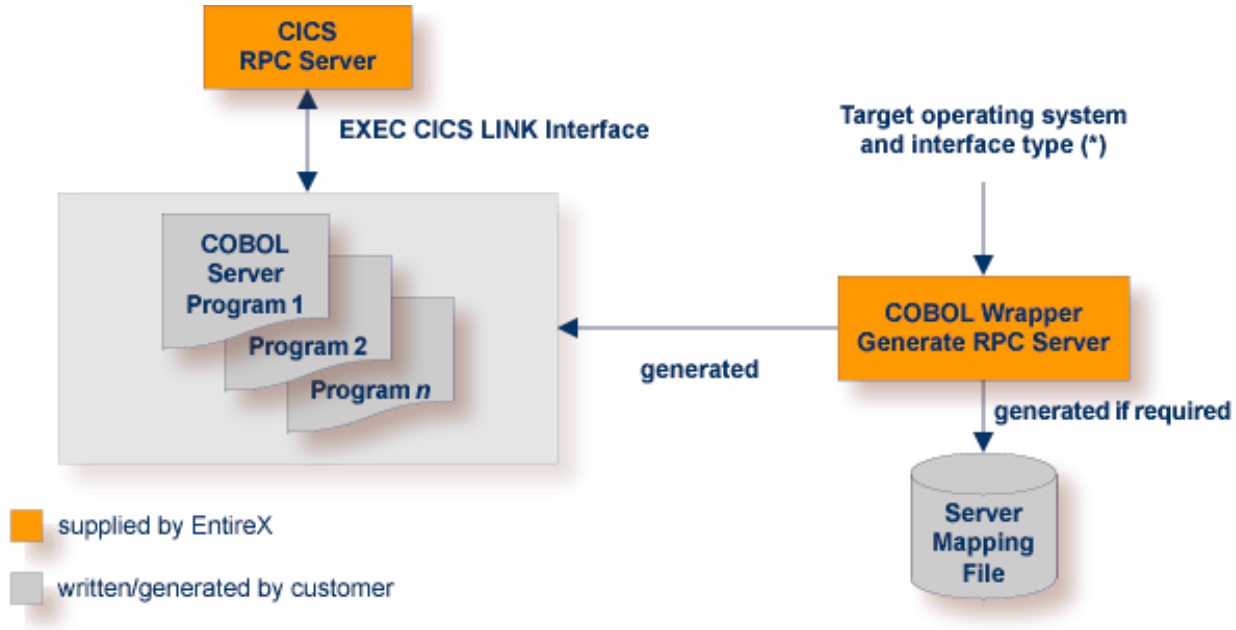
Using the COBOL Wrapper for CICS with Channel Container Calling Convention (z/OS)

This section covers the following topics:

- Introduction
- CICS Channel Container IDL Rules
- Restrictions
- Example 1: Same Container for Direction In and Out
- Example 2: Different Container for Direction In and Out
- Example 3: Multiple Containers
- Example 4: Variable Number of Containers (Direction Out Only)
- Steps

Introduction

This mode applies to z/OS. See also *COBOL Scenarios* in the CICS RPC Server documentation.



(*) 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 an EntireX Workbench file with extension .svm or .cvm. See *Server Mapping Files for COBOL*.
   - *Server*-side mapping files (.svm): Deploy these to the RPC server. See Deploying Server-side Mapping Files to the RPC Server under *Server-side Mapping Files* in the CICS RPC Server documentation.
   - *Client*-side mapping files (.cvm): These 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 EntireX Workbench documentation) and re-generate the client interface objects. For connections with the webMethods EntireX Adapter you need to update your Adapter connection. See *Step 3: Select the Connection Type* in the Integration Server Wrapper documentation.

See *How to Set the Type of Server Mapping Files* for how to define use of server-side or client-side mapping.

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 *Aborting RPC Server Customer Code and Returning Error to RPC Client* in the CICS RPC Server 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.

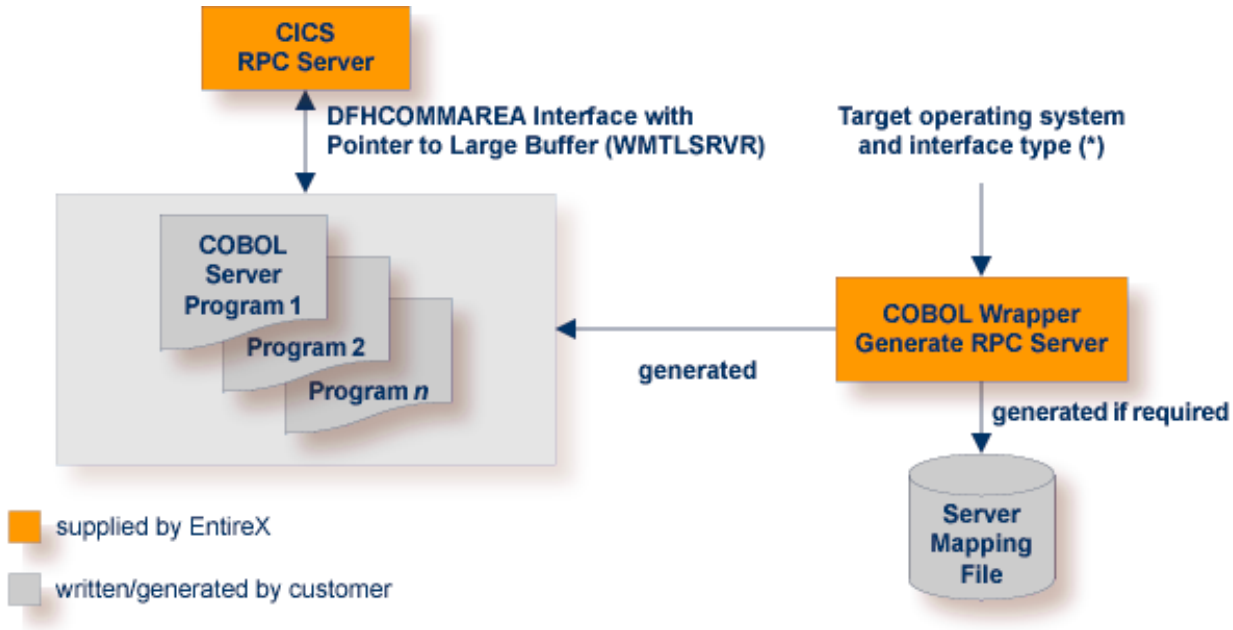
6 Link (bind) the server to an executable program, using the standard linker (binder) of the target platform. Give your server a CICS program name that is the same as the `program-name` in the IDL file (see `program-definition` under *Software AG IDL Grammar* in the IDL Editor documentation).

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

- Install your server(s) as separate CICS program(s).
- If you are using a *server*-side mapping file, a concatenation of the `program-name` and the `library-name` given in the IDL is used to locate the server mapping file. See `program-definition` and `library-definition` under *Software AG IDL Grammar* in the IDL Editor documentation. Example: If a client performs an RPC request that is based on the IDL program name `CALC` and the IDL library `EXAMPLE`, the RPC server will dynamically try to locate logically the server mapping file `EXAMPLECALC` and execute the program with the COBOL name defined in the server mapping. See ***Customize Automatically Generated Server Names***. If no corresponding program can be found, the access will fail.
- If you are using a *client*-side mapping file, the server mapping is taken from 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.
- If you are using the CICS RPC Server, before using your server(s), check if you need to alter
  - CICS settings - for example `TWASIZE` - before using your server(s); see CICS Settings under *Administering the EntireX RPC Server* in the CICS RPC Server documentation
  - *IBM LE Runtime Options* - for example AMODE24, how to trap ABENDS etc.

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

This mode applies to z/OS and z/VSE. See also *COBOL Scenarios* under in the CICS RPC Server documentation.



(*) 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 CICS RPC server.
- 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.

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 an EntireX Workbench file with extension .svm or .cvm. See *Server Mapping Files for COBOL*.
   - *Server*-side mapping files (.svm): Deploy these to the RPC server. See *Deploying Server-side Mapping Files to the RPC Server* in the CICS RPC Server (z/OS, z/VSE) sections of the documentation.
   - *Client*-side mapping files (.cvm): These 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 EntireX Workbench documentation) and re-generate the client interface objects. For connections with the webMethods EntireX Adapter you need to update your Adapter connection. See *Step 3: Select the Connection Type* in the Integration Server Wrapper documentation.

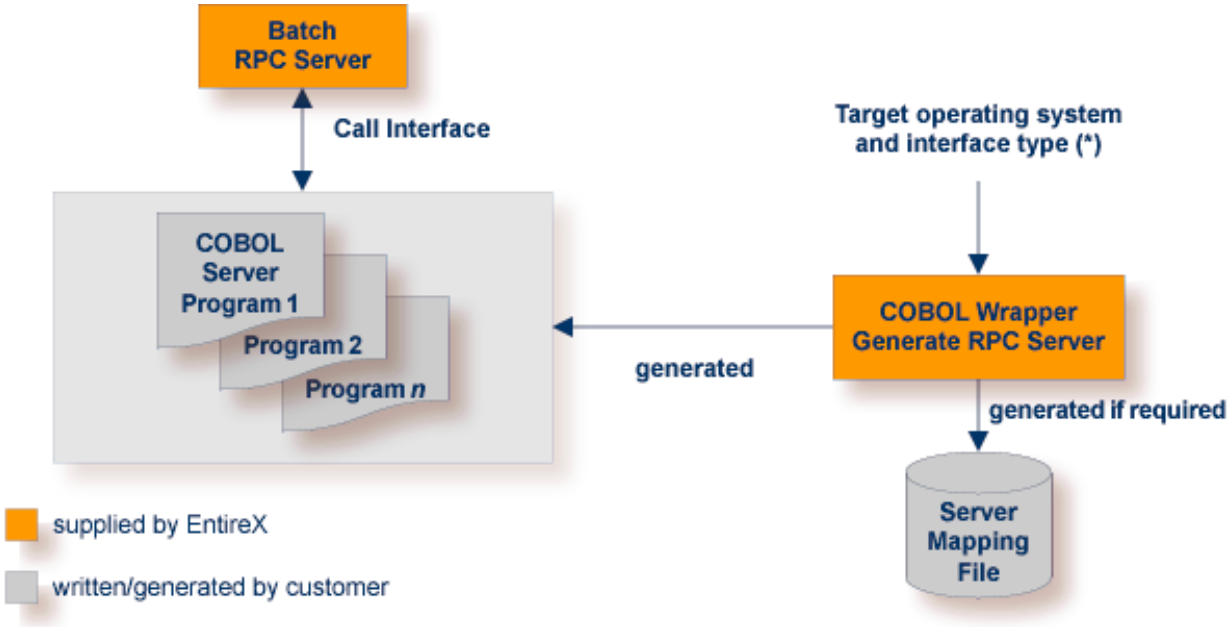
   See *How to Set the Type of Server Mapping Files* for how to define use of server-side or client-side mapping.
3. If necessary, use FTP to transfer the server (skeleton(s)) to the target platform where you write your server.
4. Use the generated server (skeleton(s)) and complete it by applying your application logic. Note the information given in ***Software AG IDL to COBOL Mapping*** and *Aborting RPC Server Customer Code and Returning Error to RPC Client* in the CICS RPC Server 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 CICS RPC server.
   - Install your server(s) as separate CICS program(s).
   - If you are using a *server*-side mapping file, a concatenation of the `program-name` and the `library-name` given in the IDL is used to locate the server mapping file. See `program-definition` and `library-definition` under *Software AG IDL Grammar* in the IDL Editor documentation. Example: If a client performs an RPC request that is based on the IDL program name `CALC` and the IDL library `EXAMPLE`, the RPC server will dynamically try

to locate logically the server mapping file `EXAMPLECALC` and execute the program with the COBOL name defined in the server mapping. See ***Customize Automatically Generated Server Names***. If no corresponding program can be found, the access will fail.

- If you are using a *client*-side mapping file, the server mapping is taken from 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.
- If you are using the CICS RPC Server, before using your server(s), check if you need to alter
  - CICS settings - for example `TWASIZE`; see *CICS Settings* in the z/OS or z/VSE RPC Server documentation
  - for z/OS additionally *IBM LE Runtime Options* - for example AMODE24, how to trap ABENDS etc.

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

This mode applies to z/OS, BS20000/OSD, z/VSE and IBM i. See also *COBOL Scenarios* in the Batch RPC Server documentation.



(*) 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 Batch RPC server.

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. If a server mapping file is required, it has to be provided. A server mapping file is an EntireX Workbench file with extension .svm or .cvm. See *Server Mapping Files for COBOL*.
   - *Server*-side mapping files (.svm): Deploy these to the RPC server. See *Deploying Server-side Mapping Files to the RPC Server* in the respective sections of the documentation.

- *Client*-side mapping files (.cvm): These 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 EntireX Workbench documentation) and re-generate the client interface objects. For connections with the webMethods EntireX Adapter you need to update your Adapter connection. See *Step 3: Select the Connection Type* in the Integration Server Wrapper documentation.

See *How to Set the Type of Server Mapping Files* for how to define use of server-side or client-side mapping.

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 *Aborting RPC Server Customer Code and Returning Error to RPC Client* in the Batch RPC Server documentation.
- **IBM i**
  Consider multithreading issues:
  - Your server has to be implemented as an ILE COBOL program of type *PGM.
  - The RPC server is running in a multithreaded environment. Therefore your server must be thread-safe. This implies that all commands and subprograms accessed in your servers must allow multithreads.
  - Please note that some COBOL statements do not support multithreads. Using statements that are not thread-safe (e.g. `STOP RUN`) can result in the RPC server ending abnormally. Therefore the server programs have to be terminated with a thread-safe statement, for example `EXIT PROGRAM`. For details, see the IBM documentation *Language Restrictions under THREAD and Preparing ILE COBOL Programs for Multithreading*.

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

- **BS2000/OSD**
  - The IDL types U or UV require a compiler that supports COBOL data type NATIONAL. See *BS2000/OSD 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/OSD**
  There is no need to link the server modules with the BS2000/OSD 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`.
  - 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`.
  - When linking/binding servers, the CRTPGM parameter `ACTGRP (*CALLER)` must be specified. This guarantees that the server application runs in the same activation group as the calling RPC server.
- **Other Platforms**
  Use the standard linker (binder) of the target platform.

7 Provide your server to the Batch RPC Server.

- **IBM i**
  - Put the server into a library whose name corresponds to the library name in the IDL file (see `library-definition` under *Software AG IDL Grammar* in the IDL Editor documentation).
  - If you put the server program into a library other than the library name given in the IDL (e.g. *MyLib*), you must tell this to the RPC server, using the server parameter `Library=Fix(MyLib)`. In this case, the library name sent with the client request is ignored.

    Example: If a client performs an RPC request that is based on the IDL program name CALC in the IDL library EXAMPLE, the remote RPC server will dynamically try to execute the ILE program CALC in the IBM i library EXAMPLE. If no corresponding program can be found, the access will fail.
- **Other Platforms**
  - Add the server to the Batch RPC Server STEPLIB chain.
  - If you are using a *server*-side mapping file, a concatenation of the `program-name` and the `library-name` given in the IDL is used to locate the server mapping file. See

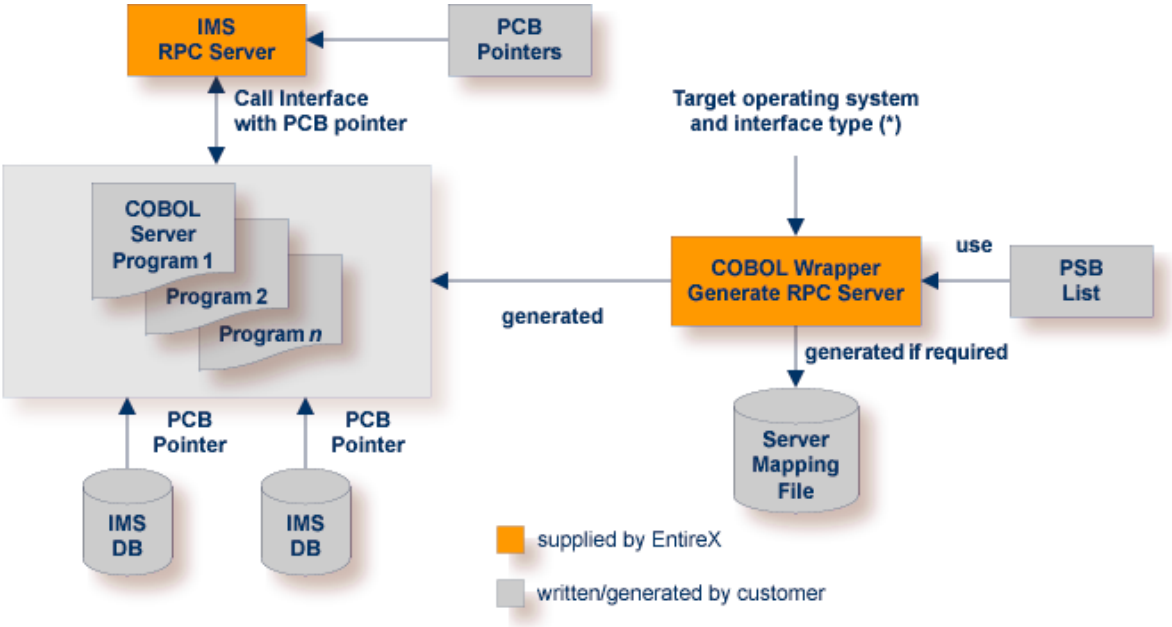
`program-definition` and `library-definition` under *Software AG IDL Grammar* in the IDL Editor documentation. Example: If a client performs an RPC request that is based on the IDL program name `CALC` and the IDL library `EXAMPLE`, the RPC server will dynamically try to locate logically the server mapping file `EXAMPLECALC` and execute the program with the COBOL name defined in the server mapping. See ***Customize Automatically Generated Server Names***. If no corresponding program can be found, the access will fail.

- If you are using a *client*-side mapping file, the server mapping is taken from 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.
- If neither a server-side nor client-side mapping file is used - for example it is not required 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.

  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 IMS BMP (z/OS)

This mode applies to z/OS IMS mode BMP. See also *COBOL Scenarios* in the IMS RPC Server documentation.



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

In IMS BMP, the IMS RPC server 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 IMS RPC server (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 IMS RPC server 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
```

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 If a server mapping file is required, it has to be provided. A server mapping file is an EntireX Workbench file with extension .svm or .cvm. See *Server Mapping Files for COBOL*.

- *Server*-side mapping files (.svm): Deploy these to the RPC server. See *Deploying Server-side Mapping Files to the RPC Server*.
- *Client*-side mapping files (.cvm): These 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 EntireX Workbench documentation) and re-generate the client interface objects. For connections with the webMethods EntireX Adapter you need to update your Adapter connection. See *Step 3: Select the Connection Type* in the Integration Server Wrapper documentation.

See *How to Set the Type of Server Mapping Files* for how to define use of server-side or client-side mapping.

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 *Aborting RPC Server Customer Code and Returning Error to RPC Client* in the IMS RPC Server 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 IMS RPC server.

- Add the server to the IMS RPC server STEPLIB chain.
- If you are using a *server*-side mapping file, a concatenation of the `program-name` and the `library-name` given in the IDL is used to locate the server mapping file. See `program-definition` and `library-definition` under *Software AG IDL Grammar* in the IDL Editor documentation. Example: If a client performs an RPC request that is based on the IDL program name `CALC` and the IDL library `EXAMPLE`, the RPC server will dynamically try to locate logically the server mapping file `EXAMPLECALC` and execute the program with the

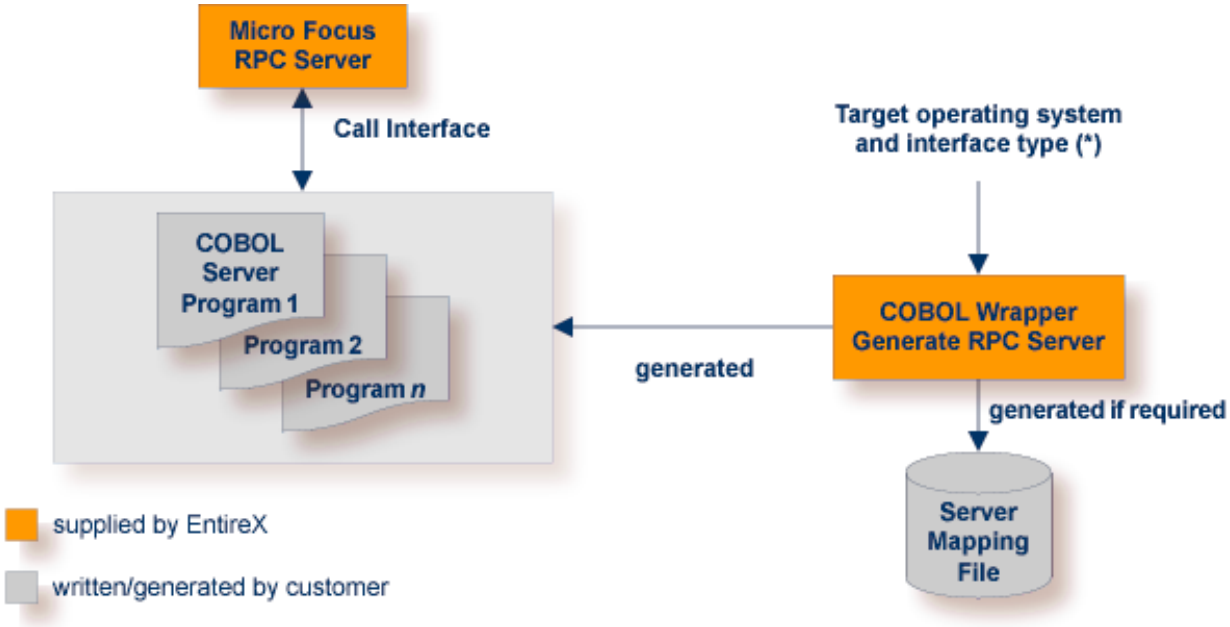
COBOL name defined in the server mapping. See ***Customize Automatically Generated Server Names***. If no corresponding program can be found, the access will fail.

- If you are using a *client*-side mapping file, the server mapping is taken from 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.
- If neither a server-side nor client-side mapping file is used - for example it is not required 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.

  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 Micro Focus (UNIX and Windows)

This mode applies to UNIX and Windows. See also *Scenarios and Programmer Information* in the Micro Focus RPC Server documentation.



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

The Micro Focus 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 Wrapperfor Micro Focus to build servers for the Micro Focus RPC server.

To use the COBOL Wrapper for Micro Focus

1. Generate a server (skeleton(s)) for the target operating system, for example "Windows", and use interface type "Micro Focus with standard linkage calling convention". See ***Generating COBOL Source Files from Software AG IDL Files*** for details.
2. If a server mapping file is required, it has to be provided. A server mapping file is an EntireX Workbench file with extension .svm or .cvm. See *Server Mapping Files for COBOL*.
   - *Server*-side mapping files (.svm): Deploy these to the RPC server. See *Deploying Server-side Mapping Files to the RPC Server*.

- *Client*-side mapping files (.cvm): These 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 EntireX Workbench documentation) and re-generate the client interface objects. For connections with the webMethods EntireX Adapter you need to update your Adapter connection. See *Step 3: Select the Connection Type* in the Integration Server Wrapper documentation.

See *How to Set the Type of Server Mapping Files* for how to define use of server-side or client-side mapping.

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

4 Import the modules into your Micro Focus IDE.

5 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***.

6 Compile and - if the format requires it - link (bind) and package your server(s) to one of the following formats:

- Micro Focus intermediate code (int) or generated code (gnt). These formats can also be packaged into a Micro Focus library file (lbr). In this case the `program-name` (see `program-definition` under *Software AG IDL Grammar* in the IDL Editor documentation) given in the IDL file must match the library file name. The `library-name` (`library-definition` under *Software AG IDL Grammar* in the IDL Editor documentation) given in the IDL file is ignored and not used.
- Under Windows to a DLL, and under UNIX to a shared library (so/sl). The `library-name` (`library-definition` under *Software AG IDL Grammar* in the IDL Editor documentation) given in the IDL file must match the executables file name, and the `program-name` (see `program-definition` under *Software AG IDL Grammar* in the IDL Editor documentation) given in the IDL file must match an entry point.

7 Provide your server to the Micro Focus RPC server.

- Make sure your server(s) are accessible by the Micro Focus RPC server:
  - under UNIX, for example with the `LD_LIBRARY_PATH` environment variable
  - under Windows, for example with the `PATH` environment variable.
- If you are using a *server*-side mapping file, a concatenation of the `program-name` and the `library-name` given in the IDL is used to locate the server mapping file. See `program-definition` and `library-definition` under *Software AG IDL Grammar* in the IDL Editor documentation. Example: If a client performs an RPC request that is based on the IDL program name `CALC` and the IDL library `EXAMPLE`, the RPC server will dynamically try to locate logically the server mapping file `EXAMPLECALC` and execute the program with the COBOL name defined in the server mapping. See ***Customize Automatically Generated Server Names***. If no corresponding program can be found, the access will fail.

- If you are using a *client*-side mapping file, the server mapping is taken from 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.
- If neither a server-side nor client-side mapping file is used - for example it is not required 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.

  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.

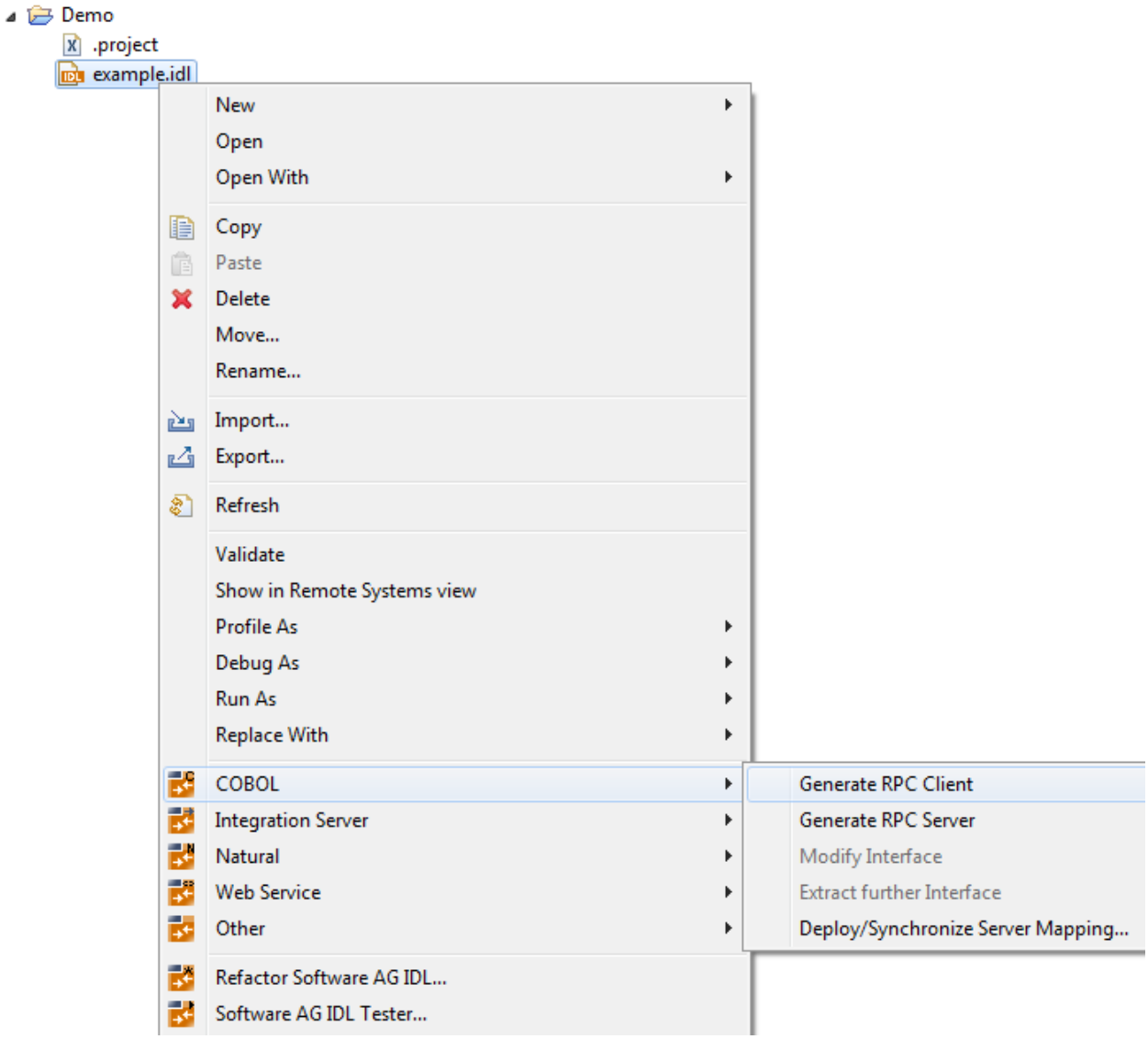
5 Generating COBOL Source Files from Software AG IDL Files

- Select an IDL File and Generate RPC Client or RPC Server 58
- Generation Settings - Properties 62
- Generation Settings - Preferences 73

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 overwritten.

Results for **RPC client**:

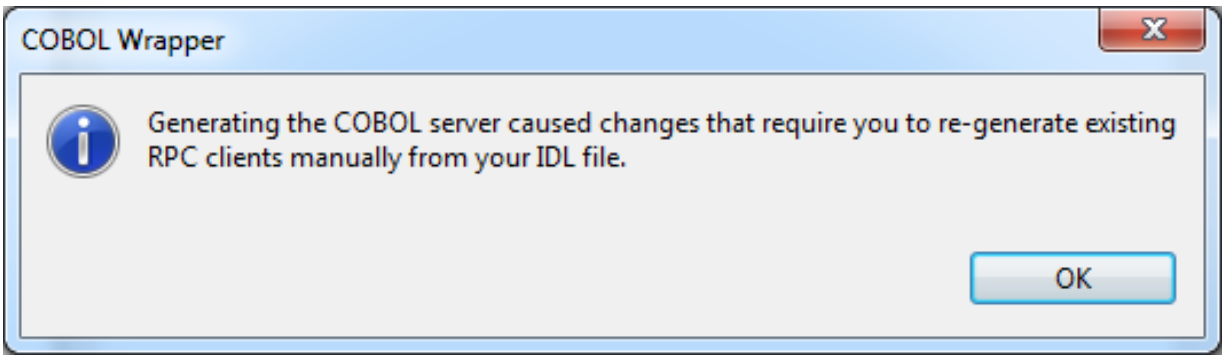
- The folders *client* and *include* are created as subfolders to the **IDL-specific Output Folder** defined in the ***Generation Settings - Properties***.
- The *client* folder contains the client interface objects, and optionally the generic RPC service module. See ***Delivered Modules***.
- The folder *include* contains the associated copybooks, the RPC communication area copybook `ERXCOMM` and optionally the copybooks `COBINIT` and `COBEXIT`.

Notes:

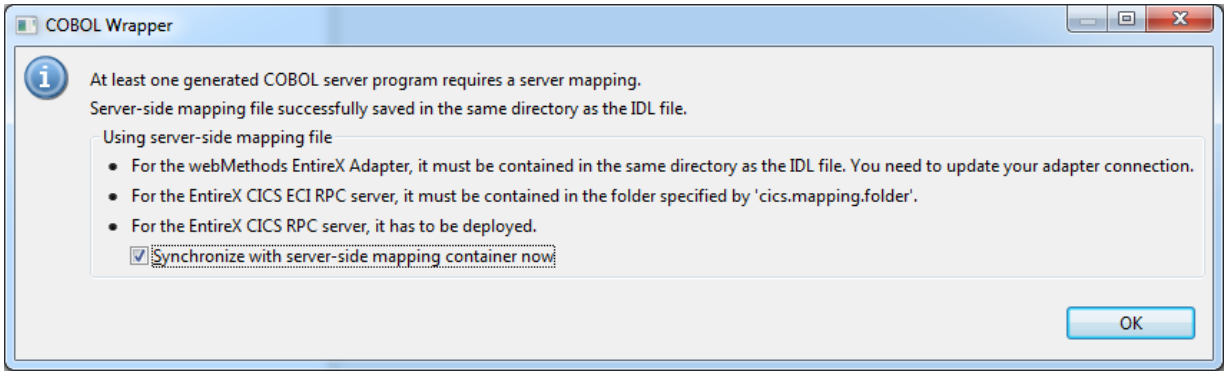
1. The generic RPC service module `COBSRVI` is only generated if the option **Generate Generic RPC Service Module COBSRVI** is set, see ***Generate Generic RPC Service Module COBSRVI***.
2. For further information on the purpose and usage of associated copybooks, see ***Using the Generated Copybooks***.
3. For further information on the purpose and usage of the RPC communication area copybook `ERXCOMM` , see ***Using the RPC Communication Area***.
4. The copybooks `COBINIT` and `COBEXIT` are only generated if **Copybook** has been selected as ***RPC Communication Area***.

Results for **RPC server**:

- If you are using client-side mapping files, 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 EntireX Workbench documentation) and re-generate the client interface objects. For connections with the webMethods EntireX Adapter you need to update your Adapter connection. See *Step 3: Select the Connection Type* in the Integration Server Wrapper documentation.
- If you are using server-side mapping files, the dialog below is displayed:



The generated server-side mapping file need to be synchronized with the server-side mapping container of the target RPC server, except for IMS Connect and CICS ECI connections with the EntireX Adapter, where they are wrapped into the Integration Server connection - the same as client-side mapping files, see *Integration Server Wrapper*.

- Check the option **Synchronize with server-side mapping container now** for the following RPC servers:
  - z/OS (CICS, Batch, IMS) | Micro Focus | BS2000/OSD | z/VSE (CICS, Batch)
- Uncheck the option **Synchronize with server-side mapping container now** for
  - EntireX Adapter and IMS Connect and CICS ECI connections
  - the following RPC servers: CICS ECI | IMS Connect
  - later synchronization of other RPC servers

- 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?* in the EntireX Workbench documentation. The server mapping file is of type client-side (extension .cvm) or server-side (.svm). See *How to Set the Type of Server Mapping Files*.

To quit the COBOL Wrapper and deploy the server-side mapping file

1. Check the option **Synchronize with server-side mapping container now** and choose **OK**. This calls the Deployment Wizard. See *Server Mapping Deployment Wizard* in the EntireX Workbench documentation.

   - If you are using the Server Mapping Deployment Wizard for first time with no predefined deployment environment preferences, continue with *Step 2a: Create a New Deployment Environment* in the EntireX Workbench documentation.

- If deployment environments are already defined, you may also continue with *Step 3: Select and Existing Deployment Environment and Deploy*.

2 Continue with ***Using the COBOL Wrapper for the Server Side***.

To quit the COBOL Wrapper without deploying the server-side mapping file

1 Clear the option **Synchronize with server-side mapping container now** and choose **OK**.

- Synchronize the server-side mapping container of the target RPC server later. See *Deploying Server-side Mapping Files to the RPC Server* in the respective sections of the documentation.
- For the webMethods EntireX Adapter and IMS Connect or CICS ECI connections, update your Adapter connection. See *Step 3: Select the Connection Type* in the Integration Server Wrapper documentation.

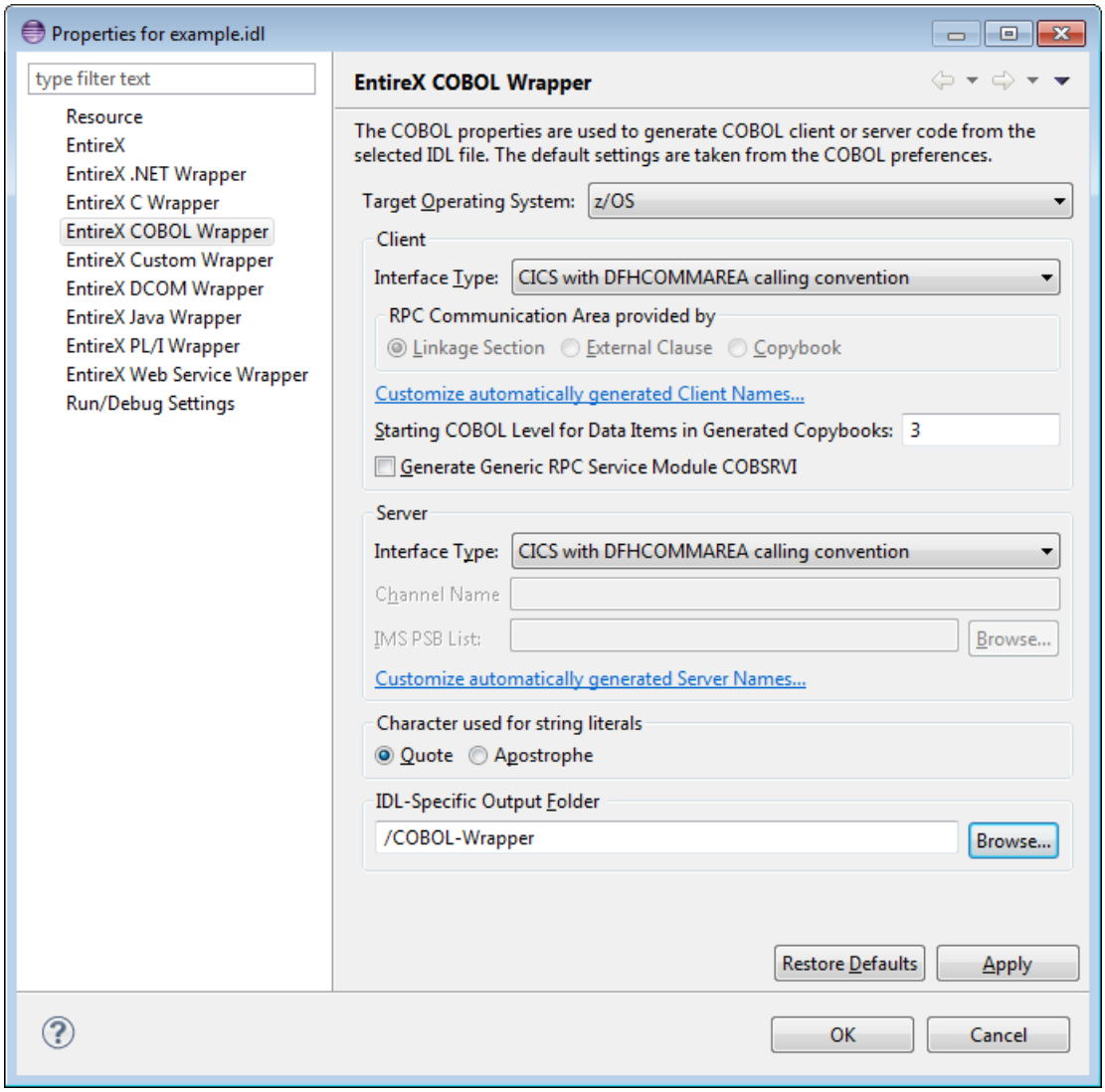
2 Continue with ***Using the COBOL Wrapper for the Server Side***.

Generation Settings - Properties

- Introduction
- Target Operating System
- Characters Used for String Literals
- IDL-specific Output Folder
- Client Interface Types
- Customize Automatically Generated Client Names
- Starting COBOL Level for Data Items in Generated Copybooks
- RPC Communication Area
- Generate Generic RPC Service Module COBSRVI
- Customize Automatically Generated Server Names
- Server Interface Types
- IMS PSB List
- Channel Name

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*

- *Generate 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/OSD operating system.
IBM i	IBM i operating system.
Windows	Microsoft Windows operating system.
UNIX	UNIX 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	Scenario	Generic RPC Services Module Usage [1]	RPC Communication Area Usage
CICS with DFHCOMMAREA calling convention	z/OS, z/VSE [2]	Use this option if you want to build a CICS RPC client application that calls the client interface object(s) with the DFHCOMMAREA interface. Follow the steps under ***Using the COBOL Wrapper for CICS with `DFHCOMMAREA` Calling Convention (z/OS and z/VSE)***.	The generic RPC service module COBSRVI for CICS with DFHCOMMAREA calling convention is installed only once within CICS as a CICS program and shared by all COBOL RPC client applications. The module has an `EXEC CICS LINK` interface to your COBOL RPC client application.	The RPC communication area is passed as a separate parameter in the DFHCOMMAREA. See ***Using the RPC Communication Area with `EXEC CICS LINK`*** and ***RPC Communication Area***.
CICS with standard linkage calling convention	z/OS, z/VSE [2]	Use this option if you want to build a CICS RPC client application that calls the client interface object(s) with a standard linkage interface. Follow the steps under ***Using the COBOL Wrapper for CICS with Call Interfaces (z/OS and z/VSE)***.	The generic RPC service module COBSRVI for CICS with standard linkage calling convention is linked to your client application or can be called dynamically. The module has a call interface to your COBOL RPC client application.	The RPC communication area is passed with one of the following options: ■ as a separate parameter in the call interface of the client interface object ■ With a COBOL `EXTERNAL` clause if a static link to the generic RPC service module is favored ■ embedded as a copybook in the client interface object and therefore invisible in the client application
Batch with standard linkage calling convention	z/OS, z/VSE [2], BS2000/OSD, IBM i [3]	Use this option if you want to build a batch RPC client application that calls the client interface object(s) with a standard linkage interface. Follow the steps under ***Using the COBOL Wrapper for Batch (z/OS, BS2000/OSD, z/VSE and IBM i)***.	The generic RPC service module COBSRVI with standard linkage calling convention is linked to your client application or can be called dynamically. The module has a call interface to your COBOL RPC client application.	

Interface Type	Target Operating System	Scenario	Generic RPC Services Module Usage [1]	RPC Communication Area Usage
IMS BMP with standard linkage calling convention	z/OS [4]	Use this option if you want to build an IMS RPC client application that calls the client interface object(s) with a standard linkage interface for IMS BMP mode. Follow the steps under *Using the COBOL Wrapper for IMS (z/OS)*.	The module has a call interface to your COBOL RPC client application.	See *Using the RPC Communication Area with a Standard Call Interface* and *RPC Communication Area*.
IMS MPP with standard linkage calling convention	z/OS [4]	Use this option if you want to build an IMS RPC client application that calls the client interface object(s) with a standard linkage interface for IMS MPP mode. Follow the steps under *Using the COBOL Wrapper for IMS (z/OS)*.	The module has a call interface to your COBOL RPC client application.	
IDMS/DC with standard linkage calling convention	z/OS [4]	Use this option if you want to build an IDMS/DC client application that calls the client interface object(s) with a standard linkage interface for IDMS/DC. Follow the steps under *Using the COBOL Wrapper for IDMS/DC with Call Interfaces (z/OS)*.	The module has a call interface to your COBOL RPC client application.	
Micro Focus with standard linkage calling convention	UNIX [4], Windows [4]	Use this option if you want to build a Micro Focus client application that calls the client interface object(s) with a standard linkage interface. Follow the steps under *Using the COBOL Wrapper for Micro Focus (UNIX and Windows)*.	The module has a call interface to your COBOL RPC client application.	

Notes:

1. The generic RPC service module depends on target environment (CICS, Batch...), interface type and operating system (z/OS, z/VSE...). Using the wrong module leads to unpredictable results. Except for the interface type "CICS with DFHCOMMAREA calling convention" we recommend

you use the generic RPC service module generated by the *EntireX Workbench* for a scenario as described under note 5.

2. For z/VSE (see *Installing EntireX under z/VSE*), the generic RPC service modules are provided with names that are different from COBSRVI. See ***Delivered Modules for z/VSE***.
3. For IBM i, the generic RPC service module is provided with a name different from COBSRVI. See ***Delivered Modules for IBM i***. Do not use this module; it is out of date. Generate the generic RPC service module as described under note 5.
4. No generic RPC service module is delivered directly with the installation for this environment and/or operating system. Generate the generic RPC service module as described under note 5.
5. Generating the generic RPC service module COBSRVI for a scenario is described under ***Generate Generic RPC Service Module COBSRVI***.

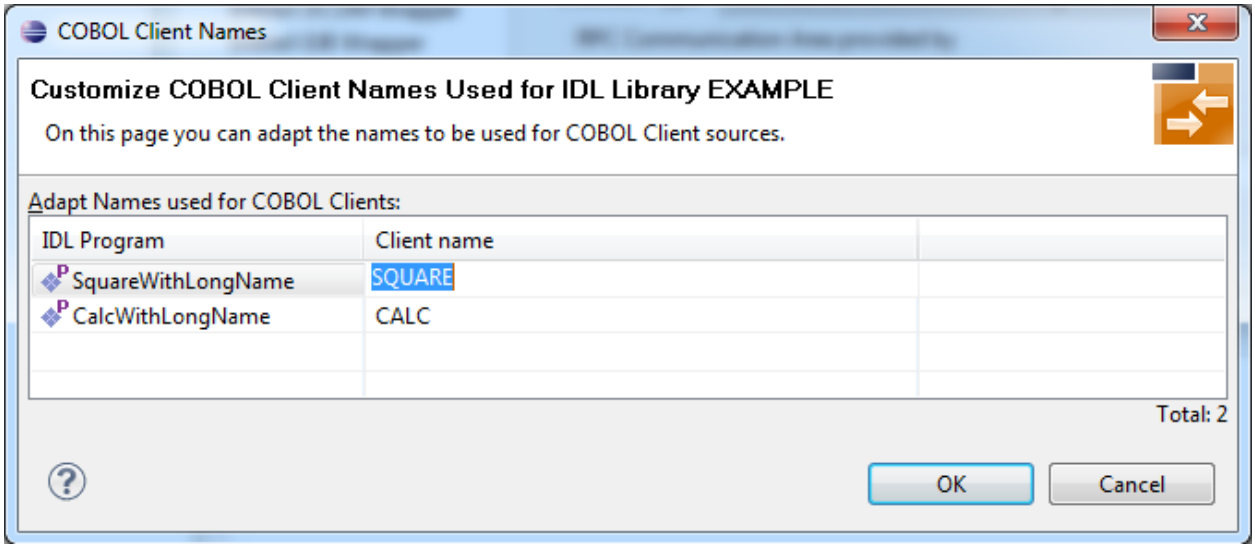
Customize Automatically Generated Client Names

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
- UNIX and Windows with Micro Focus
- BS2000/OSD

z/OS and z/VSE

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



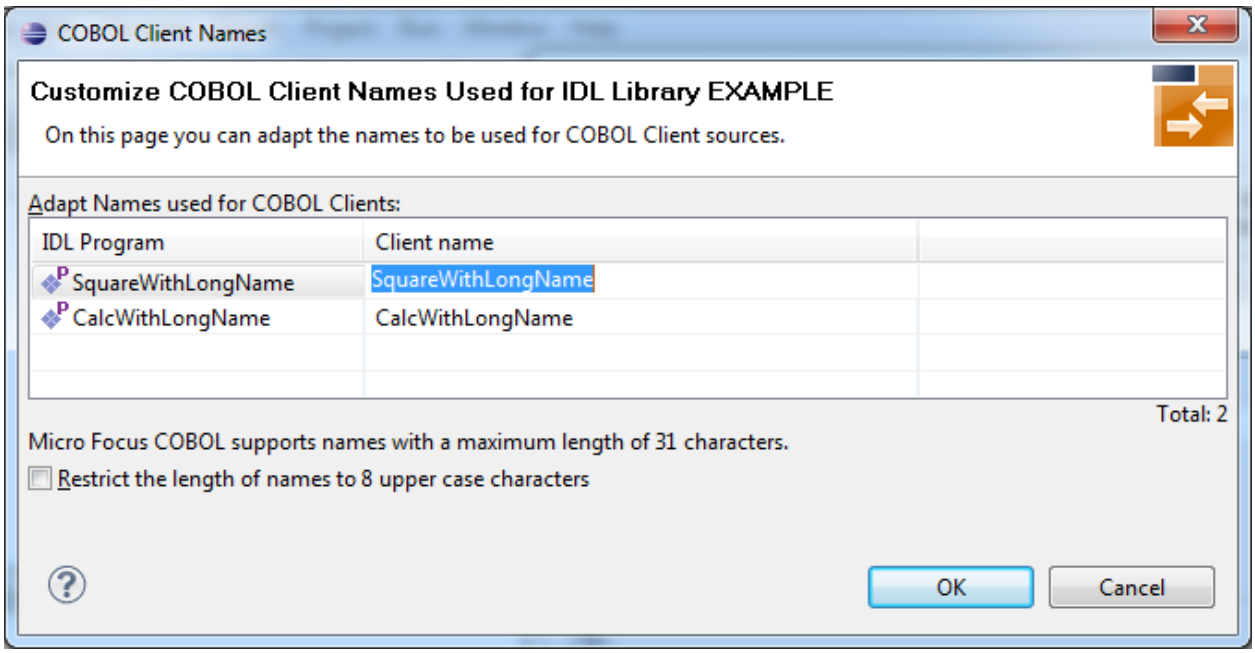
Note: If your IDL file contains more than one IDL library, the additional column **IDL Library** is displayed.

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*.

UNIX and Windows with Micro Focus

Max. 31 characters are supported as COBOL names. By default, names are generated with a maximum of 8 characters (short names).

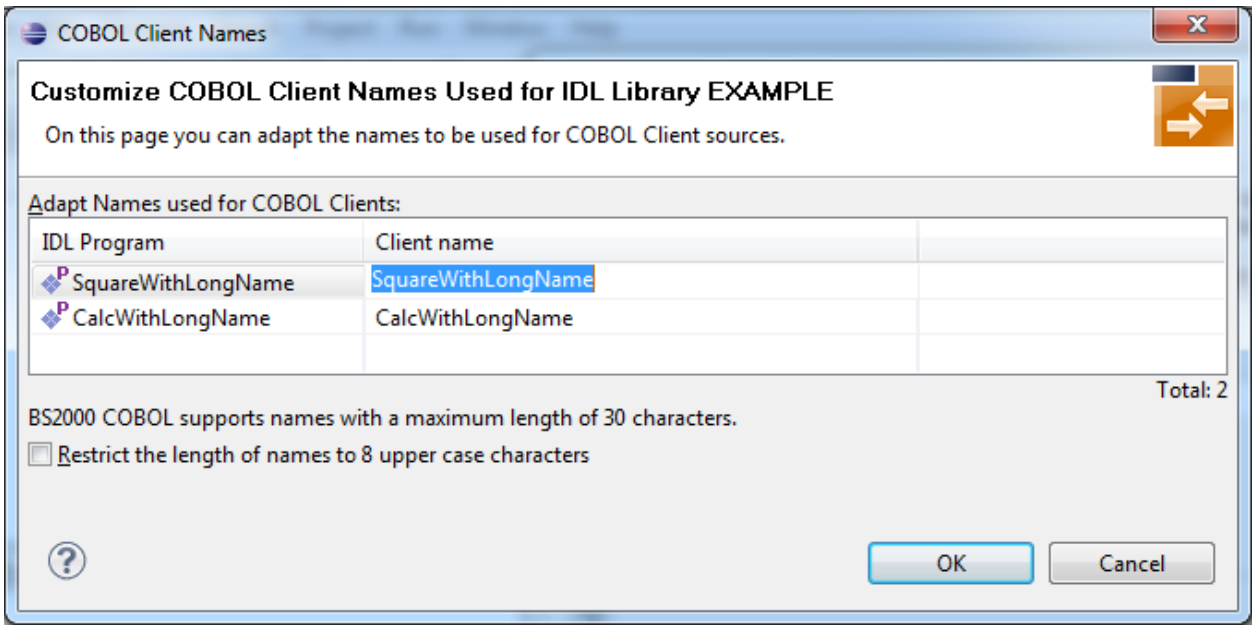


Notes:

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.

BS2000/OSD

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 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 `userID` and `password` and the server address such as *`class/servername/service`* etc.

Value	Description
`External Clause`	The RPC communication area is provided as a global area to the RPC client application and the generated client interface object(s). For more information, see option `External Clause` under ***Using the RPC Communication Area with a Standard Call Interface***. The COBOL external clause is an extension to COBOL 85 standards and might not be supported by every COBOL compiler. Check your COBOL compiler documentation.
`Linkage Section`	The RPC communication area is provided via an additional parameter between your RPC client application and the generated client interface object(s). For more information, see option `Linkage Section` under ***Using the RPC Communication Area with a Standard Call Interface*** and ***Using the RPC Communication Area with `EXEC CICS LINK`***.
`Copybook`	The RPC communication area is provided inside the generated client interface object(s). It is not visible in the RPC client application. Default values are retrieved from EntireX workbench preferences or IDL-specific properties and can be overwritten in the copybook

Value	Description
	`COBINIT` (see folder *include*). For more information, see option `Copybook` under ***Using the RPC Communication Area with a Standard Call Interface***.

Generate Generic RPC Service Module COBSRVI

The generic RPC service module COBSRVI can be optionally generated in the folder *client* in the container *folder*. It contains functions needed for RPC communication where a client interface object(s) is not needed. 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...). Using the wrong module leads to unpredictable results. Use this option to control the generation of this module. See ***Delivered Modules***.

Handling depends on the interface type:

- **CICS with DFHCOMMAREA calling convention**
  The module COBSRVI with the `EXEC CICS LINK` interface is already installed as a separate CICS program during installation (see *Installing EntireX* under z/OS | z/VSE).
  - *Clear* this option if you want to use the version already installed in CICS. This prevents the generation of the generic RPC service module. See *Delivered Modules* for **z/OS** | **z/VSE**.
  - *Check* this option if you want to replace the installed version in CICS with the version generated by the COBOL Wrapper. This makes sense if you need an update of the generic RPC service module because of a newer COBOL Wrapper version (Eclipse update without mainframe update).
- **All other calling conventions**
  - The preferred approach is to *check* this option. This will generate the generic RPC service module.
  - *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***
- ***UNIX and Windows with Micro Focus***

- *BS2000/OSD*

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. A server mapping file is an EntireX Workbench file with extension .svm or .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, z/VSE, BS2000/OSD, IBM i	Use this option if you want to build a batch RPC server application. Follow the steps under ***Using the COBOL Wrapper for Batch (z/OS, BS2000/OSD, z/VSE and IBM i)***.
IMS BMP with standard linkage calling convention	z/OS	Use this option if you want to build an IMS RPC server application for IMS BMP mode (no MPP) with standard call interfaces. If your server uses PCB pointers, see ***IMS PSB List*** below. Follow the steps under ***Using the COBOL Wrapper for IMS BMP (z/OS)***.
Micro Focus with standard linkage calling convention	UNIX, Windows	Use this option if you want to build a Micro Focus RPC server application with standard linkage interface(s). Follow the steps under ***Using the COBOL Wrapper for Micro Focus (UNIX and Windows)***.

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.

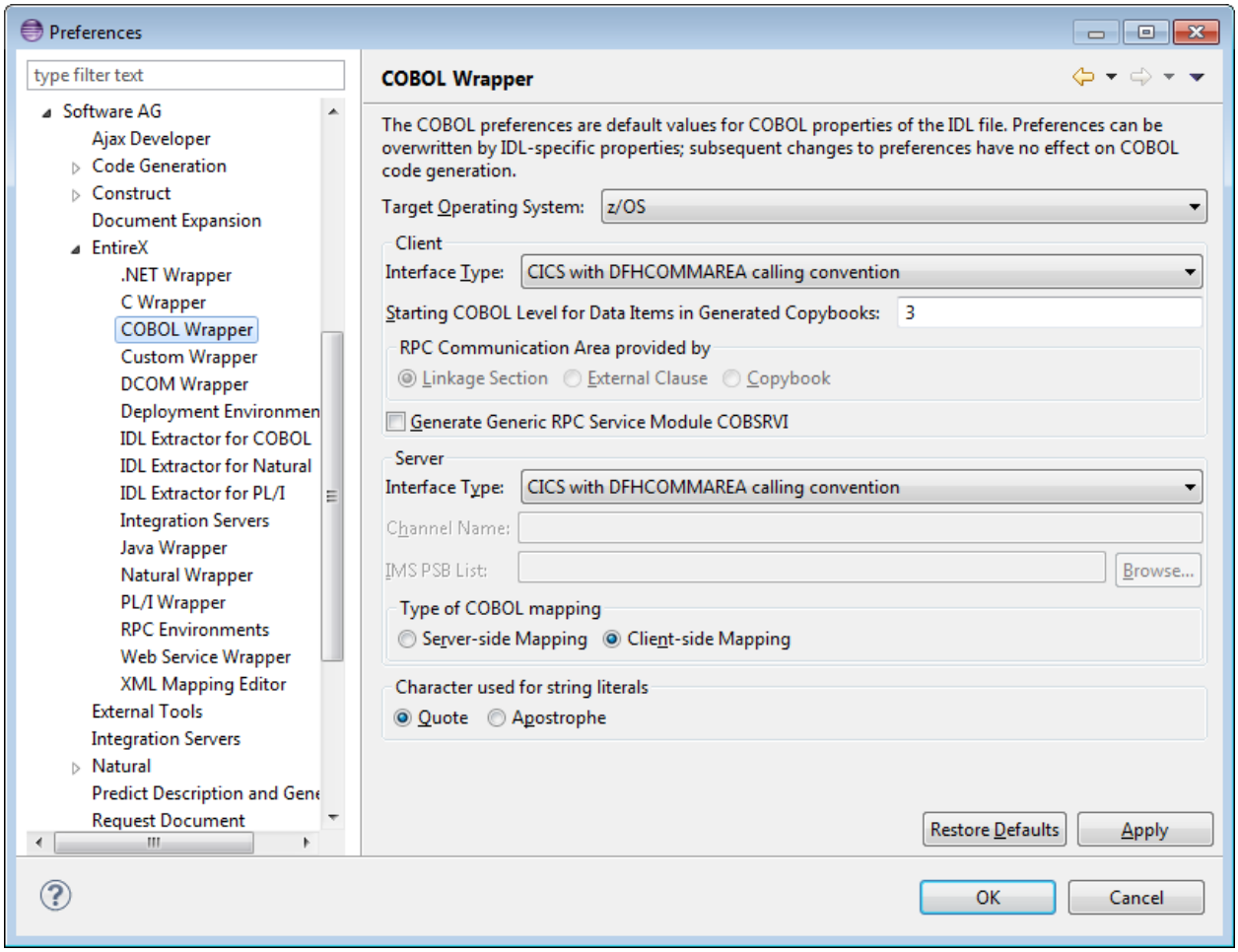
If no channel name is specified, the server is

- 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*.



- **Type of COBOL mapping**
  - Every EntireX Workbench (Eclipse) workspace is either in client-side mapping mode (generating EntireX Workbench server mapping files with extension .cvm) or server-side mapping mode (generating EntireX Workbench server mapping files with extension .svm). See *Server Mapping Files for COBOL* for an introduction. You can adjust the mode here, which will also set the mode of the IDL Extractor for COBOL to the same value. See *IDL Extractor for COBOL Preferences* in the IDL Extractor for COBOL documentation.
  - Server mapping files are generated automatically for RPC servers if required. See *When is a Server Mapping File Required? - COBOL Wrapper* in the *EntireX Workbench* documentation.
  - Server mapping files are not generated for RPC clients.

For a description of all other preferences, see ***Generation Settings - Properties***.

6 Using the COBOL Wrapper in Command-line Mode

- Command-line Options 76
- Example Generating an RPC Client 79
- Example Generating an RPC Server 80
- Further Examples 80

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 EntireX Workbench documentation.

Command-line Options

- Generate a COBOL RPC Client from IDL File
- Generate a COBOL RPC Server from IDL File

See *Using the EntireX Workbench in 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, IDMS_ZOS, MICROFOCUS_WINDOWS or MICROFOCUS_UNIX. 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	DFHCOMMAREA	LINKAGE	CICS with DFHCOMMAREA calling convention for z/OS.
		LINKAGE	LINKAGE EXTERNAL COPYBOOK	CICS with standard linkage calling convention for z/OS.
	CICS_VSE	DFHCOMMAREA	LINKAGE	CICS with DFHCOMMAREA calling convention for z/VSE.
		LINKAGE	LINKAGE EXTERNAL	CICS with standard 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/OSD.
	BATCH_I5OS	LINKAGE	LINKAGE EXTERNAL	Batch with standard linkage calling convention for IBM i.
	BATCH_ZOS	LINKAGE	LINKAGE EXTERNAL	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_ZOS with standard linkage calling convention for z/OS.
	MICROFOCUS_WINDOWS	LINKAGE	LINKAGE EXTERNAL COPYBOOK	Micro Focus with standard calling

Option	Description			
	-target	**-interface**	**-comm**	**Usage for**
				convention for Windows.
	MICROFOCUS_UNIX	LINKAGE	LINKAGE EXTERNAL COPYBOOK	Micro Focus with standard calling convention for various UNIX operating systems.
`-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 *Generate 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.

Option	Description		
	-target	**-interface**	**Usage for**
	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/OSD.
	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.
	MICROFOCUS_WINDOWS	LINKAGE	Micro Focus with standard linkage calling convention for Windows.
	MICROFOCUS_UNIX	LINKAGE	Micro Focus with standard linkage calling convention for various UNIX operating systems.
`-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 Workbench starter as described under *Using the EntireX Workbench in 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 subfolders *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 Workbench starter as described under *Using the EntireX Workbench in 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

- Mapping IDL Data Types to COBOL Data Types 84
- Mapping Library Name and Alias 88
- Mapping Program Name and Alias 89
- Mapping Parameter Names 89
- Mapping Fixed and Unbounded Arrays 90
- Mapping Groups and Periodic Groups 91
- Mapping Structures 91
- Mapping the Direction Attributes In, Out, InOut 92
- Mapping the ALIGNED Attribute 93
- Calling Servers as Procedures or Functions 93

This chapter describes the specific mapping of Software AG IDL data types, groups, arrays and structures to the COBOL programming language. Please note also the remarks and hints on the IDL data types valid for all language bindings found under *Software AG IDL File* 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 "]" surround 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*)	14
B*number*	Binary	PIC X(*number*)	12
BV	Binary variable length	not supported	
BV[*number*]	Binary variable length with maximum length	PIC X(*number*)	12, 14
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 S9(2) COMP-5	10
		PIC X	9,13
I2	Integer (medium)	PIC S9(4) COMP-5	10
		PIC S9(4) BINARY	11,13
I4	Integer (large)	PIC S9(9) COMP-5	10
		PIC S9(9) BINARY	11,13
K*number*	Kanji	PIC G(*number*/2) DISPLAY-1	5

Software AG IDL	Description	COBOL Data Type	Note
KV	Kanji variable length	not supported	
KV[*number*]	Kanji variable length with maximum length	PIC G(*number*/2 DISPLAY-1)	5, 14
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, 14

See also the hints and restrictions under *Software AG IDL File* in the IDL Editor documentation valid for all language bindings.

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* under *Software AG IDL File* in the IDL Editor documentation. 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.
   - **Micro Focus**
     The total number of digits (*number1*+*number2*) is restricted to 38 digits. The compiler option INTLEVEL"4" is generated into the client interface objects and server skeletons if more than 18 digits are defined in the IDL.

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

  Unicode is represented in UTF-16 big-endian format.
- **Micro Focus (UNIX and Windows)**
  Set the compiler directive `NSYMBOL"NATIONAL"`.

  For *clients*, Unicode can be represented in UTF-16 big-endian format (compiler directive `UNICODE(PORTABLE)`) or machine-dependent endianness UTF-16 big or little endian (compiler directive `UNICODE(NATIVE)`).

  For *servers*, Unicode can be represented in UTF-16 machine-dependent endianness (big or little endian) format only. `UNICODE(PORTABLE)` is not upported.
- **Other Operating Systems or Compilers**
  Refer to your COBOL compiler documentation.

9. COBOL for operating systems z/OS, z/VSE, BS2000/OSD 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. Supported for Micro Focus COBOL for operating systems UNIX and Windows only.
11. The value range for COBOL data type `BINARY` on z/OS, z/VSE, BS2000/OSD 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* under *Software AG IDL File* in the IDL Editor documentation. 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.

12. 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 translation or conversion will be performed.
13. Supported for operating systems z/OS, z/VSE, BS2000/OSD and IBM i only.
14. With variable length fields with maximum (AV*n*, BV*n*, KV*n* and UV*n*), mapping 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 string 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 `SPACE`s 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* under *Software AG IDL File* in the IDL Editor documentation.

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 an EntireX Workbench file with extension .svm or .cvm. See *Server Mapping Files for COBOL*. See also *Locating and Calling the Target Server* in the platform-specific 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 an EntireX Workbench file with extension .svm or .cvm. See *Server Mapping Files for COBOL*. 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* in the platform-specific 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 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`.
- For clients on all operating systems, and for servers on the operating systems z/OS, BS2000/OSD, z/VSE, UNIX and Windows for Micro Focus COBOL, IDL unbounded arrays with a 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*. 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.
- For servers on the operating system IBM i, IDL unbounded arrays with a maximum are mapped to fixed COBOL tables. On the reply, the number of occurrences is determined by `NULL` value contents. Occurrences with null values are not sent back to the calling RPC client.
- Unbounded arrays without a maximum are *not* supported.

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 clients on all operating systems and for servers on the operating systems z/OS, BS2000/OSD, z/VSE, UNIX and Windows for Micro Focus COBOL, IDL with unbounded groups with a maximum:
  - the same applies as for unbounded arrays, see ***Mapping Fixed and Unbounded Arrays***
  - if unbounded groups are nested, and depending on your target COBOL compiler,
    - they may not be supported (e.g. BS2000/OSD).
    - there is a restriction on the number of indices. Most COBOL compiler support 7 indices as a maximum.

      The EntireX Workbench 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.
- For server on the operating system IBM i, Software AG IDL unbounded groups with a maximum are mapped to fixed COBOL tables. On the reply the number of occurrences is determined by NULL value contents. Occurrences with null values are not sent back to the calling RPC client.
- Unbounded groups without a maximum are not supported.

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

If you are using a server mapping file, the RPC server considers the direction attribute found in the server mapping file. A server mapping file is an EntireX Workbench file with extension .svm or .cvm. See *Server Mapping Files for COBOL*.

If your RPC server is generated with a previous version of EntireX without a server mapping file, the RPC server considers the direction attribute sent from any RPC client, for example Java, DCOM, C, COBOL, .NET, XML and PL/I.

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*.

8 Writing Standard Call Interface Clients

- Step 1: Declare and Initialize the RPC Communication Area 96
- Step 2: Declare the Data Structures for RPC Stubs 96
- Step 3: Required Settings in the RPC Communication Area 97
- Step 4: Optional Settings in the RPC Communication Area 97
- Step 5: Issue the RPC Request 98
- Step 6: Examine the Error Code 98

This chapter describes in six steps how to write your first COBOL RPC client program.

The following steps describe how to write a COBOL client program for the client scenarios with a standard call interface: ***CICS*** | ***Batch*** | ***IMS*** | ***Micro Focus***. We recommend reading them first before writing your first RPC client program and following them if appropriate.

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 (see ***Using the RPC Communication Area***) is your interface (API) to the ***Delivered Modules***. It is important that the RPC communication area is initialized correctly with the data format defined in the copybook `ERXCOMM`, otherwise results will be unpredictable. Do not move `SPACES` to `ERX-COMMUNICATION-AREA`! The easiest approach is to use a `COBOL INITIALIZE` statement. Declare and initialize the communication area in your RPC client program as follows:

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

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

The example given here uses option `External Clause` to access the RPC communication area. See ***Using the RPC Communication Area with a Standard Call Interface***. For further options to access the RPC communication area, see ***RPC Communication Area***.

Step 2: Declare the Data Structures for RPC Stubs

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.
 MOVE "PASSWORD"       to COMM-PASSWORD.
```

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.
```

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

Step 5: Issue the RPC Request

Issue the RPC request with a standard COBOL program call:

```
CALL "CALC" USING  OPERATOR OPERAND1 OPERAND2 RESULT.
```

Step 6: Examine the Error Code

When the RPC reply is received, check that the call was successful:

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

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*.

9 Using the RPC Communication Area

- Purpose of the RPC Communication Area 100
- Using the RPC Communication Area with a Standard Call Interface 101
- Using the RPC Communication Area with EXEC CICS LINK 103

The RPC communication area is not relevant for servers.

Purpose of the RPC Communication Area

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

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

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

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

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

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

Using the RPC Communication Area with a Standard Call Interface

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

- Option External Clause
- Option Linkage Section
- Option Copybook

Option External Clause

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

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

The RPC communication area is allocated (declared) in the COBOL client application. The client interface objects are statically linked (it is not possible to call them dynamically) to the COBOL client application. It is important that the RPC communication area is initialized correctly with the data format defined in the copybook `ERXCOMM`, otherwise results will be unpredictable. Do not move `SPACES` to `ERX-COMMUNICATION-AREA`! The easiest approach is to use a `COBOL INITIALIZE` statement.

Examples

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

Option Linkage Section

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

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

The RPC communication area is allocated (declared) in the COBOL client application in the working storage section. The client interface objects can be statically linked or called dynamically. For IBM compilers, refer to documentation on the DYNAM compiler option; for other compilers, to your compiler documentation. It is important that the RPC communication area is initialized correctly with the data format defined in the copybook `ERXCOMM`, otherwise results will be unpre-

dictable. Do not move `SPACES` to `ERX-COMMUNICATION-AREA`! The easiest approach is to use a `COBOL INITIALIZE` statement. See example below.

Example

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

Step 1 has no `EXTERNAL` attribute.

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

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

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

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

Option Copybook

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

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

The RPC communication area is not visible in the client application - it is local to the client interface objects. The client interface objects can be statically linked or called dynamically. For IBM compilers,

refer to documentation on the DYNAM compiler option and for other compilers to your compiler documentation.

Example

With option `copybook`, two steps are different from the example in ***Writing Standard Call Interface Clients*** (which uses option `External Clause`):

- ***Step 1: Declare and Initialize the RPC Communication Area***

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

- ***Step 6: Examine the Error Code***

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

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

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

Using the RPC Communication Area with EXEC CICS LINK

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

The RPC communication area is allocated (declared) in the COBOL client application and passed in the DFHCOMMAREA to the client interface objects. It is important that the RPC communication area is initialized correctly with the data format defined in the copybook `ERXCOMM`, otherwise results will be unpredictable. Do not move `SPACES` to `ERX-COMMUNICATION-AREA`! The easiest approach is to use a `COBOL INITIALIZE` statement. See examples below:

Example 1

Two steps are different from the example in *Writing a COBOL RPC Client Application*. See ***Writing Standard Call Interface Clients***. Assume `CALC` is your generated *IDL interface copybook*:

Copybook CALC

```
  3 OPERATOR                    PIC X.
  3 OPERAND1                    PIC S9(8) COMP.
  3 OPERAND2                    PIC S9(8) COMP.
  3 RESULT                      PIC S9(8) COMP.
```

Step 1 contains the *IDL interface copybook* as well as the RPC communication area within one area:

RPC Client Program

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

Step 5 uses *EXEC CICS LINK interface*:

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

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

Example 2

Assume your IDL File contains an unbounded array that is mapped to `OCCURS DEPENDING ON` (see *Mapping Fixed and Unbounded Arrays*). Your generated *IDL interface copybook* looks similar to:

Copybook MYIDL

```
    3 IDL-FIELD1                  PIC X(8).
    3 IDL-FIELD2                  PIC X(32).
    3 . . .
    3 ODO-OBJECT                  PIC 9(8) BINARY.
    3 ODO-SUBJECT OCCURS 1 TO 24 DEPENDING ON ODO-OBJECT.
      4 ODO-FIELD1                PIC X(5).
      4 ODO-FIELD1                PIC X(1).
      4 . . .
```

RPC Client Program

In your RPC client program, embed the *IDL interface copybook* `MYIDL` as shown below. Use a subprogram for proper initialization of the RPC communication area. It is important to set the ODO object to the required value for upper-bound before you call the initialization subprogram:

```
  01 IDL-AREA.
    COPY MYIDL.
    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.

* 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 RPC-COMMUNICATION-AREA.
      MOVE '2000'       TO COMM-VERSION.
      MOVE '<brokerid>' TO COMM-ETB-BROKER-ID.
      MOVE 'RPC'        TO COMM-ETB-SERVER-CLASS.
      MOVE '<server>'   TO COMM-ETB-SERVER-NAME.
      MOVE 'CALLNAT'    TO COMM-ETB-SERVICE-NAME.
      MOVE '<userid>'   TO COMM-USERID.
```

```
     MOVE . . .
     EXIT PROGRAM.
END PROGRAM INIT-RPC.
```

10 Using the Generated Copybooks

- IDL Interface Copybooks 108
- COBINIT Copybook 109
- COBEXIT Copybook 109

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. We recommend you generate the IDL interface copybooks with a starting level greater than one. See ***Starting COBOL Level for Data Items in Generated Copybooks***. This 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. to the IDL:

```
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 writing a standard call interface client according to scenario ***CICS*** | ***Batch*** | ***IMS*** | ***Micro Focus***, see ***Step 2: Declare the Data Structures for RPC Stubs***.
- For writing a client according to scenario ***Using the COBOL Wrapper for CICS with `DFHCOMMAREA` Calling Convention (z/OS and z/VSE)***, see ***Using the RPC Communication Area with `EXEC CICS LINK`***.

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. See ***The RPC Communication Area (Reference)***. If the counterpart of your RPC client application is a Natural RPC server running with Natural Security, or an RPC server running with impersonation (see *Impersonation* in the respective RPC Server documentation), the security token can be generated. See ***Using the COBOL Wrapper with Natural Security and Impersonation***.

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

This chapter explains how clients built with the COBOL Wrapper use explicit broker logon and logoff functions.

It is assumed that you are familiar with the concepts of explicit and implicit broker logon. 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 ***The RPC Communication Area (Reference)***

To log on to the Broker

1 Log on to the broker with the function Logon `LO` provided by the generic RPC services module.

In the scenarios ***Micro Focus***, ***Batch***, ***CICS*** and ***IMS*** with the *Call Interface*:

```
...
* Broker Logon
 MOVE "2000" TO COMM-VERSION.
 MOVE "LO"   TO COMM-FUNCTION.
* Set broker user ID in RPC Communication Area
 MOVE "COB-USER"  TO COMM-ETB-USER-ID.
*  Call the broker
 CALL "COBSRVI" USING ERX-COMMUNICATION-AREA
 ON EXCEPTION
 ...
 NOT ON EXCEPTION
 ...
 END-CALL.
* begin of application logic
 ...
```

Or:

In the scenario ***Using the COBOL Wrapper for CICS with*** `DFHCOMMAREA` ***Calling Convention (z/OS and z/VSE)*** with the *EXEC CICS LINK Interface*:

```
...
* Broker Logon
 MOVE "2000" TO COMM-VERSION.
 MOVE "LO"   TO COMM-FUNCTION.
* Set broker user ID in RPC Communication Area
 MOVE "COB-USER"  TO COMM-ETB-USER-ID.
* Call the 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
    END-IF
 ELSE
*    Perform error-handling
 END-IF.
* begin of application logic
...
```

2 Issue your RPC requests as without using explicit logon and logoff.

Notes:

1. The logon call is the first call to the broker, before any RPC call.
2. 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.
3. If EntireX Security is to be used, see ***Using the COBOL Wrapper with EntireX Security***.

To log off from the Broker

- Log off from the broker with the function Logoff `LF` provided by the generic RPC services module with the *Call Interface*

```
...
* end of application logic including calls to generated interface objects
* Broker Logoff
 MOVE "2000" TO COMM-VERSION.
 MOVE "LF"   TO COMM-FUNCTION.
* Call the broker
 CALL "COBSRVI" USING ERX-COMMUNICATION-AREA
 ON EXCEPTION
 . . .
 NOT ON EXCEPTION
 . . .
 END-CALL.
 ...
```

Or:

with the `EXEC CICS LINK` interface (see *Logon* above).

The logoff call should be issued as soon as RPC communication is no longer needed.

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 ***The RPC Communication Area (Reference)***

To use conversational RPC

1 Open a conversation with the function Open Conversation `OC` provided by the generic RPC services module.

In the scenarios ***Micro Focus***, ***Batch CICS*** and ***IMS*** with the *Call Interface*:

```
MOVE "2000" TO COMM-VERSION.
MOVE "OC"   TO COMM-FUNCTION.
CALL "COBSRVI" USING ERX-COMMUNICATION-AREA
ON EXCEPTION
. . .
NOT ON EXCEPTION
. . .
END-CALL.
```

Or:

In the scenario ***Using the COBOL Wrapper for CICS with `DFHCOMMAREA` Calling Convention (z/OS and z/VSE)*** with the *EXEC CICS LINK Interface*:

```
 MOVE "2000" TO COMM-VERSION.
 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
    END-IF
 ELSE
*    Perform error-handling
 END-IF.
```

2 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

  In the scenarios ***Micro Focus***, ***Batch***, ***CICS*** and ***IMS*** with the *Call Interface*:

```
 MOVE "2000" TO COMM-VERSION.
 MOVE "CB"   TO COMM-FUNCTION.
 CALL "COBSRVI" USING ERX-COMMUNICATION-AREA
 ON EXCEPTION
 . . .
 NOT ON EXCEPTION
 . . .
 END-CALL.
```

  Or:

  In the scenario ***Using the COBOL Wrapper for CICS with `DFHCOMMAREA` Calling Convention (z/OS and z/VSE)*** with the *EXEC CICS LINK Interface*:

```
 MOVE "2000" TO COMM-VERSION.
 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
    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

  In the scenarios ***Micro Focus***, ***Batch***, ***CICS*** and ***IMS*** with the *Call Interface*:

```
 MOVE "2000" TO COMM-VERSION.
 MOVE "CE"   TO COMM-FUNCTION.
 CALL "COBSRVI" USING ERX-COMMUNICATION-AREA
 ON EXCEPTION
 . . .
 NOT ON EXCEPTION
 . . .
 END-CALL.
```

  Or:

  In the scenario ***Using the COBOL Wrapper for CICS with `DFHCOMMAREA` Calling Convention (z/OS and z/VSE)*** with *the EXEC CICS LINK Interface*:

```
 MOVE "2000" TO COMM-VERSION.
 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
    END-IF
 ELSE
*    Perform error-handling
 END-IF.
```

13 Using the COBOL Wrapper with Natural Security and Impersonation

This chapter explains how clients built with the COBOL Wrapper can communicate with Natural RPC Servers running under Natural Security and RPC servers running with impersonation. See *Impersonation* in the respective RPC Server documentation.

This chapter assumes that you are familiar with the concepts of Natural Security and impersonation. To communicate with such a server you will need the following components:

- the ***Delivered Modules***, which are provided to create and get a security token,
- the ***RPC Communication Area***

To authenticate against Natural Security or impersonated RPC server

1 Specify a user ID, password and optional Natural library in the RPC communication area:

```
*  Client information :                    bytes 101-300
    10 COMM-USERID.
              15 COMM-USERID1      PIC X(8).
              15 COMM-USERID2      PIC X(8).
    10 COMM-PASSWORD               PIC X(8).
    10 COMM-LIBRARY                PIC X(8).
    10 COMM-SECURITY-TOKEN-LENGTH  PIC 9(4) BINARY.
    10 COMM-SECURITY-TOKEN         PIC X(100).
    10 FILLER                      PIC X(66).
```

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

In the scenarios ***Micro Focus***, ***Batch***, ***CICS*** and ***IMS*** with the *Call Interface*:

- For ***RPC Communication Area*** setting `Linkage` and `External`:

```
  MOVE "2000" TO COMM-VERSION.
  MOVE "CT"   TO COMM-FUNCTION.
* Set user ID and password in RPC Communication Area
  MOVE "NAT-USER"  TO COMM-USERID.
  MOVE "NAT-PWD"   TO COMM-PASSWORD.
* Additional for Natural Security set library in RPC Communication Area
  MOVE "NAT-LIB"   TO COMM-LIBRARY.
  CALL "COBSRVI" USING ERX-COMMUNICATION-AREA
  ON EXCEPTION
  . . .
  NOT ON EXCEPTION
  . . .
  END-CALL.
```

- For ***RPC Communication Area*** setting `Copybook`. Add the following COBOL Statements to the `COBINIT` copybook:

```
  MOVE "CT"   TO COMM-FUNCTION.
* Set user ID and password in RPC Communication Area
  MOVE "NAT-USER"  TO COMM-USERID.
  MOVE "NAT-PWD"   TO COMM-PASSWORD.
* Additional for Natural Security set library in RPC Communication Area
  MOVE "NAT-LIB"   TO COMM-LIBRARY.
  CALL "COBSRVI" USING ERX-COMMUNICATION-AREA
```

  See also ***Using the Generated Copybooks***.

Or:

In the scenario ***Using the COBOL Wrapper for CICS with `DFHCOMMAREA` Calling Convention (z/OS and z/VSE)*** with *the EXEC CICS LINK Interface*:

```
  MOVE "2000" TO COMM-VERSION.
  MOVE "CT"   TO COMM-FUNCTION.
* Set user ID and password in RPC Communication Area
  MOVE "NAT-USER"  TO COMM-USERID.
  MOVE "NAT-PWD"   TO COMM-PASSWORD.
* Additional for Natural Security 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
```

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

After successful return from the generic RPC services module, the security fields in the RPC communication area are properly set, so they can be used in subsequent RPC requests to a secure RPC server, such as:

- Natural RPC server running with Natural Security
- RPC server running with impersonation. See *Impersonation* in the respective RPC Server documentation.

We strongly recommend using *Using SSL/TLS* if you send a security token with the COBOL Wrapper to the secure RPC server. See also *SSL/TLS Parameters for EntireX Clients and Servers* under *SSL/TLS and Certificates with EntireX* in the EntireX Security documentation.

14 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 the COBOL Wrapper with Natural Security and Impersonation***.

You will need the following components:

- the ***Delivered Modules***, which are provided to create and get a security token,
- the ***RPC Communication Area***

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

1 Specify the Natural library in the RPC communication area:

```
*  Client information :                    bytes 101-300
    10 COMM-USERID.
              15 COMM-USERID1      PIC X(8).
              15 COMM-USERID2      PIC X(8).
    10 COMM-PASSWORD               PIC X(8).
    10 COMM-LIBRARY                PIC X(8).
    10 COMM-SECURITY-TOKEN-LENGTH  PIC 9(4) BINARY.
    10 COMM-SECURITY-TOKEN         PIC X(100).
    10 FILLER                      PIC X(66).
```

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

In the scenarios with the *Call Interface* for ***Micro Focus***, ***Batch***, ***CICS*** and ***IMS***:

- For ***RPC Communication Area*** setting `Linkage` and `External`:

```
  MOVE "2000" TO COMM-VERSION.
  MOVE "CT"   TO COMM-FUNCTION.
* Set library in RPC Communication Area
  MOVE "NAT-LIB"   TO COMM-LIBRARY.
  CALL "COBSRVI" USING ERX-COMMUNICATION-AREA
  ON EXCEPTION
  . . .
  NOT ON EXCEPTION
  . . .
  END-CALL.
```

- For ***RPC Communication Area*** setting `Copybook`. Add the following COBOL Statements to the `COBINIT` copybook:

```
   MOVE "CT"   TO COMM-FUNCTION.
 * Set library in RPC Communication Area
   MOVE "NAT-LIB"   TO COMM-LIBRARY.
   CALL "COBSRVI" USING ERX-COMMUNICATION-AREA
```

  See also ***Using the Generated Copybooks***.

Or:

In the scenario ***Using the COBOL Wrapper for CICS with `DFHCOMMAREA` Calling Convention (z/OS and z/VSE)*** with the `EXEC CICS LINK` interface:

```
  MOVE "2000" TO COMM-VERSION.
  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
     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.

15 Reliable RPC for COBOL Wrapper

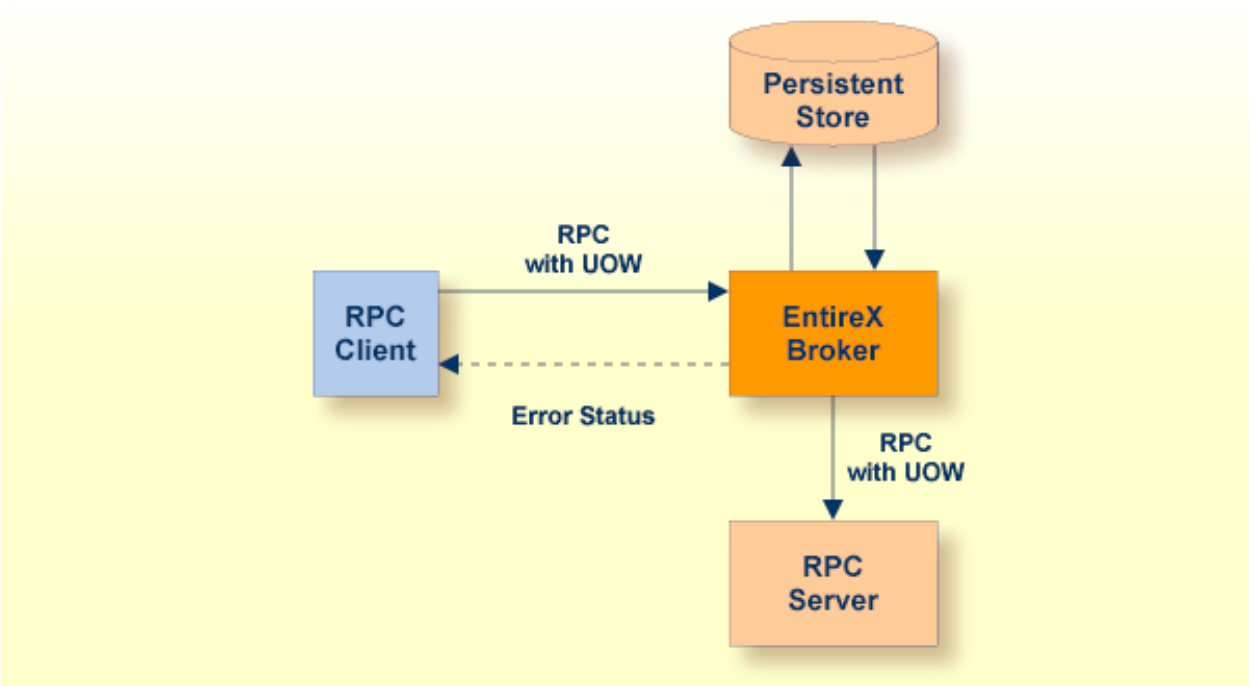
- Introduction to Reliable RPC 126
- Writing a Client 127
- Writing a Server 132
- Broker Configuration 132

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)*** and Linkage access to RPC communication.

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

Step 1: Declare the Data Structures for RPC Client Interface Objects

For every program definition in the Software AG IDL file, the templates will generate a copybook file that describes the customer data of the interface as a COBOL structure. For ease of use, the copybook can be embedded into the RPC client program.

However, if more appropriate, customer data structures can be used. In this case the COBOL data types and structures must match the interfaces of the generated client interface objects, otherwise unpredictable results will occur.

```
* Declare the customer data of the generated RPC interface
 01 SENDMAIL.
    02 SM-COMA.
       03 SM-TOADDRESS          PIC  X(60).
       03 SM-SUBJECT            PIC  X(20).
       03 SM-TEXT               PIC X(100).
```

Step 2: Declare and Initialize the RPC Communication Area

The RPC communication area must be declared and initialized in your RPC client program as follows:

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

* 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.
 MOVE "PASSWORD"       to COMM-PASSWORD.
```

Step 4a: 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.
```

Step 4b: Examine the Error Code

Check whether the logon call was successful or not.

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 6a: 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.
```

Step 6b: Examine the Error Code

When the RPC message is returned, it needs to be checked whether it was successful or not:

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

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*.

Note: After successful call (Step 6a) the `UOWID` is available in the RPC communication area field `COMM-ETB-UOW-ID`. See ***The RPC Communication Area (Reference)***.

Step 7a: 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.
```

Note: After successful call the UOW status is available in the RPC communication area field `COMM-RELIABLE-STATUS`. See *The RPC Communication Area (Reference)*.

Step 7b: Examine the Error Code

Check whether the check status call was successful or not.

Step 8: Send a Second RPC Message

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

Step 9: Check the Reliable RPC Message Status

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

Step 10a: 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.
```

Step 10b: Examine the Error Code

Check whether the commit call was successful or not.

Step 11: Send a Third RPC Message

Send a third reliable RPC message. See **Step 5a** and **Step 5b**.

Step 12: Check the Reliable RPC Message Status

Check the reliable RPC message before the rollback call. See **Step 6**.

Step 13a: 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.
```

Step 13b: Examine the Error Code

When the rollback call is returned, check whether it was successful or not. If the rollback call failed, an explicit EOC needs to be sent:

```
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.
```

Step 14a: 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.
```

Step 14b: Examine the Error Code

Check whether the logoff call was successful or not.

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` in the relevant sections of the 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 `UWTIME` and `UWSTAT-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`).

16 Using the COBOL Wrapper with EntireX Security

This chapter explains how clients built with the COBOL Wrapper use EntireX Security.

To use EntireX Security you need the following components:

- *Delivered Modules*
- *The RPC Communication Area (Reference)*

To use EntireX Security

1. Set the `COMM-ETB-PASSWORD` and set `COMM-KERNEL-SECURITY` to "Y". See *The RPC Communication Area (Reference)*.
2. Log on to the broker with the function Logon `LO` provided by the generic RPC services module as described under *Using Broker Logon and Logoff*.

   - In the scenarios with the *Call Interface* for *CICS*, *Batch*, *IMS* and *Micro Focus*:

```
 ...
* Broker Logon
 MOVE "2000" TO COMM-VERSION.
 MOVE "LO"   TO COMM-FUNCTION.

* Set Broker userid in RPC Communication Area
 MOVE "COB-USER"  TO COMM-ETB-USERID.

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

* Call the broker
 CALL "COBSRVI" USING ERX-COMMUNICATION-AREA
 ON EXCEPTION
 ...
```

```
 NOT ON EXCEPTION
 ...
 END-CALL.
* begin of application logic
 ...
```

- In the scenario ***Using the COBOL Wrapper for CICS with*** `DFHCOMMAREA` ***Calling Convention (z/OS and z/VSE)*** with the `EXEC CICS LINK` interface:

```
 ...
* Broker Logon
 MOVE "2000" TO COMM-VERSION.
 MOVE "LO"   TO COMM-FUNCTION.

* Set Broker userid in RPC Communication Area
 MOVE "COB-USER"  TO COMM-ETB-USERID.

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

* Call the 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
    END-IF
 ELSE
*    Perform error-handling
 END-IF.

* begin of application logic
 ...
```

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

4 For logoff, see ***Using Broker Logon and Logoff***.

17 Using SSL/TLS

- z/OS 136
- z/VSE 138
- UNIX, Windows, BS2000/OSD 140

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 and Certificates with EntireX* in the EntireX Security documentation. This chapter describes using SSL with the COBOL Wrapper on z/OS and z/VSE.

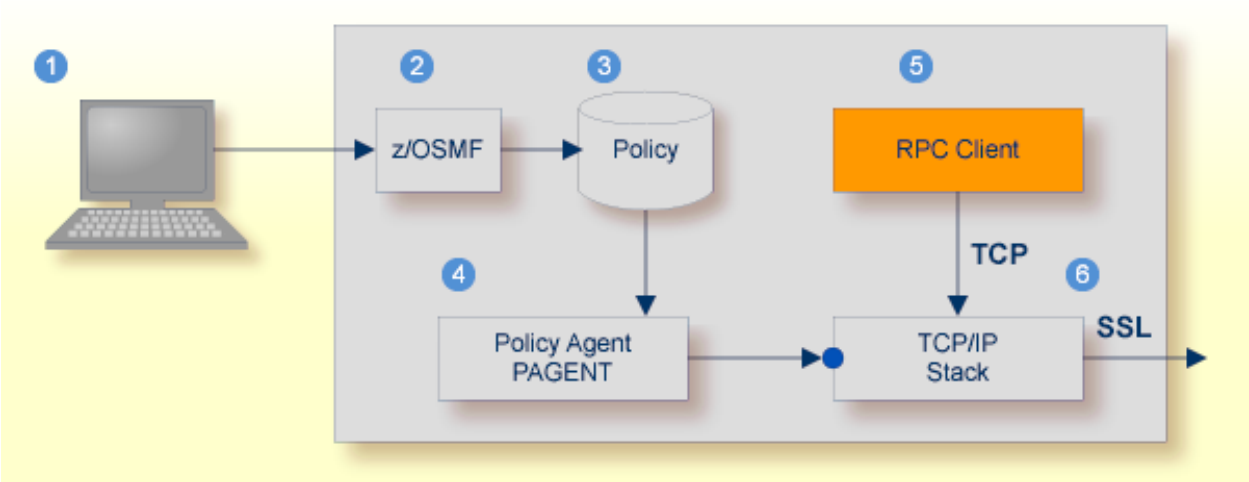
z/OS

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).

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 *Default Certificates Delivered with EntireX* under *SSL/TLS and Certificates 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, 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:

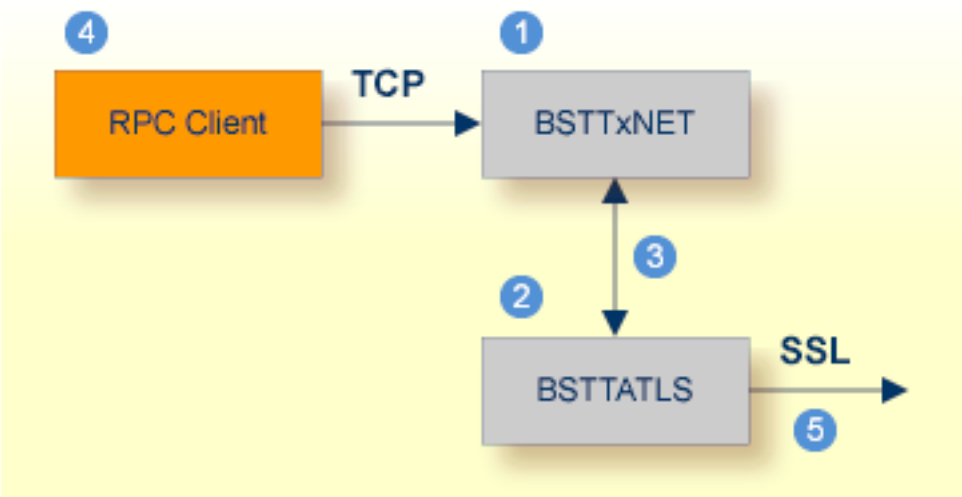
- *Running Broker with SSL/TLS Transport* in the platform-specific Administration documentation
- *Settting up and Administering the EntireX Broker SSL Agent* in the UNIX and Windows Administration documentation
- *Support for SSL/TLS* in the EntireX Adapter documentation (for Direct RPC)

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 *Default Certificates Delivered with EntireX* under *SSL/TLS and Certificates 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
   - *Settting up and Administering the EntireX Broker SSL Agent* in the UNIX and Windows 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/OSD

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/OSD.

18 Client and Server Examples for Micro Focus (UNIX and Windows)

- Basic RPC Client Examples - CALC, SQUARE 142
- Basic RPC Server Examples - CALC, SQUARE 142

This chapter describes the examples provided for the COBOL Wrapper for Micro Focus. All examples here can be found in the EntireX directory *examples/RPC* under UNIX and Windows.

Basic RPC Client Examples - CALC, SQUARE

For Micro Focus environments, the CALC and SQUARE clients are built with COBOL Wrapper "Micro Focus with standard linkage calling convention" interface type. See ***Client Interface Types*** for more information.

Name	Type	Description	Notes
CALCCLT.cbl	COBOL source code	A client application calling the remote procedure (RPC service) CALC, with associated *example.idl*.	1
SQRECLT.cbl	COBOL source code	A client application calling the remote procedure (RPC service) SQUARE, with associated *example.idl*.	1

Notes:

1. Application built according to the client-side build instructions under ***Using the COBOL Wrapper for Micro Focus (UNIX and Windows)***.

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

Basic RPC Server Examples - CALC, SQUARE

For Micro Focus environments, the CALC and SQUARE servers are built with COBOL Wrapper "Micro Focus with standard linkage calling convention" interface type. See ***Server Interface Types*** for more information.

Name	Type	Description	Notes
CALC.cbl	COBOL source code	A server application providing the remote procedure CALC (RPC service), with associated *example.idl*.	1
SQUARE.cbl	COBOL source code	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 under ***Using the COBOL Wrapper for Micro Focus (UNIX and Windows)***.

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

19 Client and Server Examples for z/OS Batch

- Basic RPC Client Examples - CALC, SQUARE 146
- Basic RPC Server Examples - CALC, SQUARE 148

This chapter describes the examples provided for the COBOL Wrapper for z/OS Batch. All examples here can be found in the EntireX directory *examples/RPC* under UNIX and Windows. They are also available for z/OS, if this is installed. See *Extracting the EntireX RPC Examples from their Container Data Set*.

Basic RPC Client Examples - CALC, SQUARE

- CALC Client
- SQUARE Client

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	EXP910.CCCO	Client interface object for IDL program CALC.	1
CALCCLT	COBOL source code	EXP910.CCCO	A client application calling the remote procedure (RPC service) CALC, with associated *example.idl*.	2
CALCIGY	JCL	EXP910.CCCO	Job (JCL) to build the RPC client CALCCLT.	3
CALCRUN	JCL	EXP910.CCCO	Job (JCL) to execute the RPC client CALCCLT.	3
CALC	COBOL copybook	EXP910.CICO	Client interface object copybook for IDL program CALC.	1
COBSRVI	COBOL source code	EXP910.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 EntireX Workbench.
2. Application built according to the client-side build instructions, see ***Using the COBOL Wrapper for Batch (z/OS, BS2000/OSD, z/VSE and IBM i)***.
3. Adapt the JCL to your needs.
4. See ***Generate 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	EXP910.CCCO	Generic RPC service module for Batch.	4
SQRECLT	COBOL source code	EXP910.CCCO	A client application calling the remote procedure (RPC service) SQUARE, with associated example.idl.	1
SQREIGY	JCL	EXP910.CCCO	Job (JCL) to build the RPC client SQRECLT.	2
SQRERUN	JCL	EXP910.CCCO	Job (JCL) to execute the RPC client SQRECLT.	2
SQUARE	COBOL source code	EXP910.CCCO	Client interface object for IDL program SQUARE.	3
SQUARE	COBOL copybook	EXP910.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/OSD, 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 EntireX Workbench.
4. See *Generate 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
- SQUARE Server

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	EXP910.CVCO	A server application providing the remote procedure CALC (RPC service), with associated *example.idl*.	1
CALCIGY	JCL	EXP910.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/OSD, 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.

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	EXP910.CVCO	Job (JCL) to build the remote procedure SQUARE (RPC service)	2
SQUARE	COBOL source code	EXP910.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/OSD, 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

- Basic RPC Client Examples - CALC, SQUARE 152
- Basic RPC Server Examples - CALC, SQUARE 156
- Advanced CICS Channel Container RPC Server Example 157

This chapter describes the examples provided for the COBOL Wrapper for z/OS CICS. All examples here can be found in the EntireX directory *examples/RPC* under UNIX and Windows. They are also available for z/OS, if this is installed. See *Extracting the EntireX RPC Examples from their Container Data Set*.

Basic RPC Client Examples - CALC, SQUARE

- CALC Client using DFHCOMMAREA
- CALC Client using Call Interface
- SQUARE Client using DFHCOMMAREA
- SQUARE Client using Call Interface

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	EXP910.DCCO	CSD Definition for RPC client CALC1CLT.	
CALC1IGY	JCL	EXP910.DCCO	Job (JCL) to build the RPC client CALC1CLT.	2
CALC1MAP	CICS Map	EXP910.DCCO	CICS Map definition for RPC client and CALC1CLT.	
CALC1	COBOL source code	EXP910.DCCO	Client interface object for IDL program CALC1, alias of CALC.	1
CALC1CLT	COBOL source code	EXP910.DCCO	An RPC client application calling the remote procedure (RPC service) CALC.	3
CALC1MAP	COBOL copybook	EXP910.DICO	Description of input and output fields of map CALC1MAP.	
CALC1	COBOL copybook	EXP910.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 EntireX Workbench.
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.

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	EXP910.DCCO	Client interface object for IDL program CALC.	1
CALCCLT	COBOL source code	EXP910.DCCO	An RPC client application calling the remote procedure (RPC service) CALC.	2
CALCDFH	CICS CSD	EXP910.DCCO	CSD Definition for RPC client CALCCLT.	
CALCIGY	JCL	EXP910.DCCO	Job (JCL) to build the RPC client CALCCLT.	3
CALCMAP	CICS Map	EXP910.DCCO	CICS Map definition for RPC client CALCCLT.	
CALC	COBOL copybook	EXP910.DICO	Client interface object copybook for IDL program CALC.	1
CALCMAP	COBOL copybook	EXP910.DICO	Description of input and output fields of map CALCMAP.	
COBSRVI	COBOL source code	EXP910.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 EntireX Workbench.
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 ***Generate 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	EXP910.DCCO	CSD Definition for RPC client SQRE1CLT.	
SQREI1GY	JCL	EXP910.DCCO	Job (JCL) to build the RPC client SQRE1CLT.	2
SQRE1MAP	CICS Map	EXP910.DCCO	CICS Map definition for RPC clients SQRE1CLT.	
SQRE1	COBOL source code	EXP910.DCCO	Client interface object for IDL program SQRE1, alias of SQUARE.	1
SQRE1CLT	COBOL source code	EXP910.DCCO	An RPC client application calling the remote procedure (RPC service) SQUARE.	3
SQRE1MAP	COBOL copybook	EXP910.DICO	Description of input and output fields of map SQRE1MAP.	
SQRE1	COBOL copybook	EXP910.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 EntireX Workbench.
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.

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	EXP910.DCCO	Generate RPC service module for CICS with call interface.	4
SQRECLT	COBOL source code	EXP910.DCCO	An RPC client application calling the remote procedure (RPC service) SQUARE.	2
SQREDFH	CICS CSD	EXP910.DCCO	CSD Definition for RPC client SQRECLT.	
SQREIGY	JCL	EXP910.DCCO	Job (JCL) to build the RPC client SQRECLT.	3
SQREMAP	CICS Map	EXP910.DCCO	CICS Map definition for RPC client SQRECLT.	
SQUARE	COBOL source code	EXP910.DCCO	Client interface object for IDL program SQUARE.	1
SQREMAP	COBOL copybook	EXP910.DICO	Description of input and output fields of map SQREMAP.	
SQUARE	COBOL copybook	EXP910.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 EntireX Workbench.
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 *Generate 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.

Basic RPC Server Examples - CALC, SQUARE

- CALC Server
- SQUARE Server

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	EXP910.DVCO	A server application providing the remote procedure CALC (RPC service), with associated *example.idl*.	1
CALCDFH	CICS CSD	EXP910.DVCO	CSD Definition for remote procedure CALC (RPC service).	
CALCIGY	JCL	EXP910.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	EXP910.DVCO	CSD Definition for remote procedure SQUARE (RPC service).	
SQREIGY	JCL	EXP910.DVCO	Job (JCL) to build the remote procedure SQUARE (RPC service).	2
SQUARE	COBOL source code	EXP910.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	EXP910.DVCO	A server application providing the remote procedure TWOC (RPC service), with associated *CICSChannelContainer.idl*.	1
TWOCDFH	CICS CSD	EXP910.DVCO	CSD Definition for remote procedure TWOC (RPC service).	
TWOCIGY	JCL	EXP910.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

- CALC Server 162
- SQUARE Server 162

This chapter describes examples provided for COBOL on operating system z/OS with the TP system IMS for an MP region. All examples here can be found in the EntireX directory *examples/RPC* under UNIX and Windows. They are also available for z/OS if installed. See *Extracting the EntireX RPC Examples from their Container Data Set*.

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 *IMS Connect RPC Server* or the *EntireX Adapter*.

Name	Type	Data Set	Description	Notes
CALC	COBOL source code	EXP910.MVCO	A server application providing the remote procedure CALC (RPC service) with associated *example.idl*.	
CALCIGY	JCL	EXP910.MVCO	Job (JCL) to build the remote procedure CALC (RPC service).	1
CALCSTG	IMS definition	EXP910.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 *IMS Connect RPC Server* or the *EntireX Adapter*.

Name	Type	Data Set	Description	Notes
SQUARE	COBOL source code	EXP910.MVCO	A server application providing the remote procedure SQUARE (RPC service), with associated *example.idl*.	
SQREIGY	JCL	EXP910.MVCO	Job (JCL) to build the remote procedure SQUARE (RPC service).	1

Name	Type	Data Set	Description	Notes
SQRESTG	IMS definition	EXP910.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/OSD

- Overview of Client and Server Examples for BS2000/OSD 166
- Creating the Sample COBOL Client Programs 169
- Creating the Sample COBOL Server Programs 170
- Running the Sample COBOL Client Programs 170

This chapter describes the examples provided for the COBOL Wrapper for BS2000/OSD.

Overview of Client and Server Examples for BS2000/OSD

The following examples are delivered for BS2000/OSD:

- CALC Example
- SQUARE Example
- SENDMAIL Reliable RPC Example
- Notes

All examples here can be found in the EntireX directory *examples/RPC* under UNIX and Windows. If EntireX is installed under BS2000/OSD, the examples are also available on this platform.

CALC Example

Client

Element	Type	LMS Library	Comment	Notes
CREATE-CALC-CLIENT	J	EXP811.COBC	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	EXP811.COBC	S-procedure to bind the CALC COBOL sample client application.	
RUN-COBOL-COMPILER	J	EXP811.COBC	S-procedure to run the COBOL2000 / COBOL85 compiler.	2
RUN-CALC-CLIENT	J	EXP811.COBC	S-procedure to run the CALC COBOL sample client application.	
CALCCLT.COB	S	EXP811.COBC	Main program source of the CALC COBOL example.	1
CALC.COB	S	EXP811.COBC	COBOL RPC client interface object.	1
CALC	S	EXP811.COBC	COBOL RPC interface copybook.	1
COBSRVI.COB	S	EXP811.COBC	Generic RPC service.	1
ERXCOMM	S	EXP811.COBC	Layout of the RPC communication area. See ***The RPC Communication Area (Reference)***.	1
CLIENT-ADAPARM	S	EXP811.COBC	Adabas ADALNK `IDTNAME` parameter required when using the NET transport method. It is shared by all clients.	
CLIENT-INPARM-CALC	S	EXP811.COBC	CALC client input parameters.	

Server

Element	Type	LMS Library	Comment	Notes
CREATE-CALC-SERVER	J	EXP811.COBS	S-procedure to generate the CALC COBOL example server. It makes use of `RUN-COBOL-COMPILER`.	2
RUN-COBOL-COMPILER	J	EXP811.COBS	S-procedure to run the COBOL2000 / COBOL85 compiler.	2
CALC.COB	S	EXP811.COBS	Server program source of CALC COBOL example.	1

SQUARE Example

Client

Element	Type	LMS Library	Comment	Notes
CREATE-SQUARE-CLIENT	J	EXP811.COBC	S-procedure to generate the `SQUARE` COBOL sample client application. It uses `RUN-COBOL-COMPILER` and `BIND-SQUARE-CLIENT`.	2
BIND-SQUARE-CLIENT	J	EXP811.COBC	S-procedure to bind the `SQUARE` COBOL sample client application.	
RUN-COBOL-COMPILER	J	EXP811.COBC	S-procedure to run the COBOL2000 / COBOL85 compiler.	2
RUN-SQUARE-CLIENT	J	EXP811.COBC	S-procedure to run the `SQUARE` COBOL sample client application.	
SQRECLT.COB	S	EXP811.COBC	Main program source of `SQUARE` COBOL example.	1
SQUARE.COB	S	EXP811.COBC	COBOL RPC client interface object.	1
SQUARE	S	EXP811.COBC	COBOL RPC interface copybook.	1
COBSRVI.COB	S	EXP811.COBC	Generic RPC service.	1
ERXCOMM	S	EXP811.COBC	Layout of the RPC communication area. See *The RPC Communication Area (Reference)*.	1
CLIENT-ADAPARM	S	EXP811.COBC	Adabas ADALNK `IDTNAME` parameter required when using the NET transport method. It is shared by all clients	
CLIENT-INPARM-SQUARE	S	EXP811.COBC	`SQUARE` client input parameters.	

Server

Element	Type	LMS Library	Comment	Notes
CREATE-SQUARE-SERVER	J	EXP811.COBS	S-procedure to generate the `SQUARE` COBOL sample server. It uses `RUN-COBOL-COMPILER`.	2
RUN-COBOL-COMPILER	J	EXP811.COBS	S-procedure to run the COBOL2000 / COBOL85 compiler.	2
SQUARE.COB	S	EXP811.COBS	Server program source of the `SQUARE` COBOL example.	1

SENDMAIL Reliable RPC Example

Client

Element	Type	LMS Library	Comment	Notes
CREATE-MAIL-CLIENT	J	EXP811.COBC	S-procedure to generate the `SENDMAIL` reliable RPC COBOL sample client application. It uses `RUN-COBOL-COMPILER` and `BIND-MAIL-CLIENT`.	2
BIND-MAIL-CLIENT	J	EXP811.COBC	S-procedure to bind the `SENDMAIL` reliable RPC COBOL sample client application.	
RUN-COBOL-COMPILER	J	EXP811.COBC	S-procedure to run the COBOL2000 / COBOL85 compiler.	2
RUN-MAIL-CLIENT	J	EXP811.COBC	S-procedure to run the `SENDMAIL` reliable RPC COBOL sample client application.	
MAILCLT.COB	S	EXP811.COBC	Main program source of the `SENDMAIL` reliable RPC COBOL example.	1
SENDMAIL.COB	S	EXP811.COBC	COBOL RPC client interface object.	1
SENDMAIL	S	EXP811.COBC	COBOL RPC interface copybook.	1
COBSRVI.COB	S	EXP811.COBC	Generic RPC service.	1
ERXCOMM	S	EXP811.COBC	Layout of the RPC communication area. See *The RPC Communication Area (Reference)*.	1
CLIENT-ADAPARM	S	EXP811.COBC	Adabas ADALNK `IDTNAME` parameter required when using the NET transport method. It is shared by all clients.	
CLIENT-INPARM-MAIL	S	EXP811.COBC	SENDMAIL reliable RPC client input parameters.	

Server

Element	Type	LMS Library	Comment	Notes
CREATE-MAIL-SERVER	J	EXP811.COBS	S-procedure to generate the `SENDMAIL` reliable RPC COBOL sample server. It makes use of `RUN-COBOL-COMPILER`.	2
RUN-COBOL-COMPILER	J	EXP811.COBS	S-procedure to run the COBOL2000 / COBOL85 compiler.	2
SENDMAIL.COB	S	EXP811.COBS	Server program source of the `SENDMAIL` reliable RPC COBOL example.	1

Notes

1. When compiling the COBOL client and server sample source programs, the compiler may issue warnings depending on the compiler used. These warnings can be ignored.
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.

Creating the Sample COBOL Client Programs

To create the `CALC`, `SQUARE` and `SENDMAIL` clients, parametrize S-procedures `CREATE-CALC-CLIENT`, `CREATE-SQUARE-CLIENT` and `CREATE-MAIL-CLIENT` in `EXP811.COBC` and choose the compiler installed on your system.

For more details, see also see the procedure headers in the delivered job control.

Enter the following commands:

Procedure Parameter	Description	Default
EXP-COB-CLT	COBOL client examples library	EXP811.COBC
COMPILER	The COBOL compiler to be used: COBOL2000 or COBOL85	COBOL2000

For more details, see also see the procedure headers in the delivered job control.

Enter the following commands:

```
/CALL-PROCEDURE *LIB(LIB=EXP811.COBC,ELE=CREATE-CALC-CLIENT)
/CALL-PROCEDURE *LIB(LIB=EXP811.COBC,ELE=CREATE-SQUARE-CLIENT)
/CALL-PROCEDURE *LIB(LIB=EXP811.COBC,ELE=CREATE-MAIL-CLIENT)
```

These procedures call the COBOL compiler and binder to generate corresponding L-elements stored in the EXP-COB-CLT library (the default is EXP811.COBC).

Creating the Sample COBOL Server Programs

To create the `CALC`, `SQUARE` and `SENDMAIL` server programs, parametrize S-procedures `CREATE-CALC-SERVER`, `CREATE-SQUARE-SERVER` and `CREATE-MAIL-SERVER` in `EXP811.COBS` and choose the compiler installed on your system.

Procedure Parameter	Description	Default
EXP-SRV-LIB	COBOL server examples library	EXP811.COBS
COMPILER	The COBOL compiler to be used: COBOL2000 or COBOL85	COBOL2000

For more details, see also see the procedure headers in the delivered job control.

Enter the following commands:

```
/CALL-PROCEDURE *LIB(LIB=EXP811.COBS,ELE=CREATE-CALC-SERVER)
/CALL-PROCEDURE *LIB(LIB=EXP811.COBS,ELE=CREATE-SQUARE-SERVER)
/CALL-PROCEDURE *LIB(LIB=EXP811.COBS,ELE=CREATE-MAIL-SERVER)
```

These procedures call the COBOL Compiler to generate three corresponding object modules stored as R-elements in EXP-SRV-LIB (the default is EXP811.COBS).

There is no need to link the object modules with the BS2000/OSD Common Runtime Environment (CRTE) library. The CRTE is loaded once dynamically in the corresponding worker task of the RPC server where the server program is executed.

Running the Sample COBOL Client Programs

Running the `CALC` client is described below. Running the `SQUARE` and the `SENDMAIL` clients is similar.

To run the `CALC` client

1 Adapt S-element CLIENT-INPARM-CALC in EXP811.COBC.

```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*             Example CALC Client Input Parameter                 *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
BROKERID <ipaddr>:<port>:TCP                                       *
* BROKERID ETB<nnnnn>::NET                                          *
* USERID   <userid>                                                 *
* PASSWORD <password>                                               *
CLASS    RPC                                                       *
SERVER   SRV1                                                      *
SERVICE  CALLNAT                                                   *
LOGON                                                              *
CALC     + 00012345 00067890                                       *
CALC     - 00067890 00012345                                       *
CALC     * 00001234 00005678                                       *
CALC     / 00005678 00001234                                       *
CALC     % 00005678 00001234                                       *
LOGOFF                                                             *
END
```

Set up `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**
  ETB<*nnnnn*>::NET

  where *nnnnn* is the ID under which EntireX Broker is connected to the Adabas ID table and
  NET is the protocol name.

2 Adapt S-element CLIENT-ADAPARM.

If "NET" is chosen as transport method, specify the name of the ID table to which the broker is connected:

```
ADALNK IDTNAME=ADAxxxxx
```

where *xxxxx* is any uppercase value.

This parameter is shared between all sample clients.

3 Make sure the RPC server runs as COBOL RPC server (refer to the RPC-CONFIG S-element in library EXP811.JOBS) and library EXP811.COBS is included as `PROGRAM-LIB` in the start up procedure `START-RPC-SERVER`.

4 Enter the following command to run the `CALC` COBOL example client:

```
/CALL-PROCEDURE *LIB(LIB=EXP811.COBC,ELE=RUN-CALC-CLIENT)


CALCCLT : START
OPEN  IN: -------- : <00>
        : BROKERID : ETB001
        : CLASS    : RPC
        : SERVER   : SRV1
        : SERVICE  : CALLNAT
CALCCLT : BROKER LOGON.
CALC called successfully: 000012345 + 000067890 = 000080235
CALC called successfully: 000067890 - 000012345 = 000055545
CALC called successfully: 000001234 * 000005678 = 007006652
CALC called successfully: 000005678 / 000001234 = 000000004
CALC called successfully: 000005678 % 000001234 = 000000742
CALCCLT : BROKER LOGOFF.
CLOSE IN: -------- : <00>
CALCCLT : LEAVE
```

24 Client and Server Examples for IBM i

- Overview of Client and Server Examples for IBM i 174
- Installing and Running the Client Examples for IBM i 175
- Installing and Running the Server Examples for IBM i 175

This chapter describes the examples provided for the COBOL Wrapper for z/OS Batch.

Overview of Client and Server Examples for IBM i

The following examples are delivered for IBM i in the library EXAMPLE of the Developer's Kit for IBM i.

Module	Source file	Windows File Name	Description	Notes
CALCMENU	QCBLLESRC	*- not delivered here -*	COBOL client display file (source)	1
CALCMAIN	QCBLLESRC	*- not delivered here -*	COBOL client dialog program (source)	1
CCALC	QCBLLESRC	*- not delivered here -*	client interface object (generated)	1
RPCSRVI	QCBLLESRC	*- not delivered here -*	generic RPC service module	1
CALC	QCBLLESRC	*- not delivered here -*	RPC server calc (source)	2

Module

The name of the delivered module.

Source file

The name of the source file where the modules are delivered.

Windows File Name

IBM i examples are not delivered in the Windows installation

Description

The purpose of the module

Notes:

1. The client application is built by the source members: CALCMENU, CALCMAIN, CCALC and RPCSRVI. You can find the associated IDL file *example.idl* in the Windows installation.
2. The server application.

Installing and Running the Client Examples for IBM i

To run the client examples for IBM i

1. The EntireX product library EXX must be in your library list. It contains the Broker ACI service program EXA.
2. Confirm that the broker and the RPC server are active.
3. Start the client application CALCCLIENT that you built, see *Using the COBOL Wrapper for Batch (z/OS, BS2000/OSD, z/VSE and IBM i)*.
4. A menu similar to the following will be displayed:

```
Calculator Menu
-----------------------------------------------------

Operation:    +         (type + - * / to calculate or
                         type .       to terminate)
Operand 1:    _____

Operand 2:    _____

Result:  ___________


Broker-ID: localhost:1971    Server: SRV1
```

Specify the ID of the remote Broker and the name of the server that provides the CALC program. Specify the numbers you want to compute and press ENTER. If the Broker connection fails, you will get an appropriate error message.

Installing and Running the Server Examples for IBM i

To install and run the server examples for IBM i

1. For IBM i, the delivered program CALC in QCBLLESRC source file must be provided to the RPC server under IBM i.
2. Confirm that the broker is active.
3. Start the RPC server under IBM i.

25 Client and Server Examples for z/VSE Batch

- Basic RPC Client Examples - CALC, SQUARE 178
- Basic RPC Server Examples - CALC, SQUARE 180

This chapter describes the examples provided for the COBOL Wrapper for z/VSE Batch. All examples here can be found in the EntireX directory *examples/RPC* under UNIX and Windows.

Basic RPC Client Examples - CALC, SQUARE

- CALC Client
- SQUARE Client

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 EntireX Workbench.
2. Application built according to the client-side build instructions, see ***Using the COBOL Wrapper for Batch (z/OS, BS2000/OSD, 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 ***Generate 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/OSD, 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 EntireX Workbench.
4. See *Generate 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
- SQUARE Server

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/OSD, 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.

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/OSD, 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.

26 Client and Server Examples for z/VSE CICS

- Basic RPC CALC Example 184
- Basic RPC SQUARE Example 186

This chapter describes the examples provided for the COBOL Wrapper for z/VSE CICS. All examples here can be found in the EntireX directory *examples/RPC* under UNIX and Windows.

Basic RPC CALC Example

- CALC Client using Call Interface (CALCCLT)
- CALC Client using DFHACOMMAREA (CALC1CLT)
- CALC Server (CALC)

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 EntireX Workbench.
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 *Generate 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 EntireX Workbench.
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 *Generate 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 EntireX Workbench.
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 ***Generate 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 EntireX Workbench.
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 ***Generate 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.

27 COBOL Wrapper Reference

- The RPC Communication Area (Reference) 192
- Delivered Modules 196

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 table below for a complete listing.

Notes:

1. See below the table for an explanation of column headings.
2. The RPC communication area is provided with the generated copybook `ERXCOMM` in the folder *include* for RPC client generation. See ***Generating COBOL Source Files from Software AG IDL Files***.
3. See section ***Using the RPC Communication Area*** for the usage of the RPC communication area.

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.	R	I	-
`COMM-FUNCTION`	`LO` - log on to the Broker	O	I	1
	`LF` - log off from the Broker			1
	`OC` - open conversation			3
	`CE` - close conversation with commit			3
	`CB` - close conversation with backout			3
	`CT` - create Natural Security token			4
	`RC` - do reliable RPC commit			6
	`RR` - do reliable RPC rollback			6
	`RS` - get reliable status			6
	`EC` - end of conversation			6
`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`	Message text provided by COBOL Wrapper (short versions).	-	O	-
`ERXCOMM-AREA1`	Label.	-	-	-

RPC Communication Area Field	Explanation	Req/ Opt/ Auto	In/ Out	Notes
COMM-USERID	Label.	-	-	-
COMM-USERID1	User ID (8 characters) created into security token.	O	I	4
COMM-USERID2	User ID extension.	O	I	-
COMM-PASSWORD	Password (8 characters) created into security token.	O	I	4
COMM-LIBRARY	Library name (8 characters) created into security token.	O	I	4,5
COMM-SECURITY-TOKEN-LENGTH	Length of Natural Security token.	-	O	4
COMM-SECURITY-TOKEN	Natural Security token.	-	O	-
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	3,7
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	6,7
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	6
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.		O	6
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	1,2
COMM-ETB-PASSWORD	Corresponds to Broker ACI field PASSWORD.	O	I	1,2
COMM-ETB-TOKEN	Corresponds to Broker ACI field TOKEN.	O	I/O	-
COMM-ETB-SECURITY-TOKEN	Internal field. Corresponds to Broker ACI field SECURITY-TOKEN.	A	I/O	7
COMM-ETB-CONV-ID	Internal field. Corresponds to Broker ACI field CONV-ID.	A	I/O	3,7
COMM-ETB-WAIT	Corresponds to Broker ACI field WAIT. Default: 60 seconds.	O	I	-

RPC Communication Area Field	Explanation	Req/ Opt/ Auto	In/ Out	Notes
COMM-ETB-APIVERS	Corresponds to Broker ACI field API-VERSION. Default=4.	O	I	-
COMM-ETB-UOW-ID	Corresponds to Broker ACI field UOWID.	O	I/O	6
COMM-ETB-STORE	Corresponds to Broker ACI field STORE.	O	I/O	6
COMM-ETB-PROGRAM-OFFSET	Fields are used internally for accounting purposes. See *Accounting in EntireX Broker* in the platform-specific Administration documentation.	A	I/O	7
COMM-ETB-LIBRARY-OFFSET		A	I/O	7
APPMON-SUPPORT	Fields are used internally to support *Application Monitoring*	A	I/O	7
APPMON-VERIFY		A	I/O	7
APPMON-TIMEVALUE		A	I/O	7
APPMON-TRANSPORT-BUFFER		A	I/O	7
APPMON-LEN-TRANSPORT-BUFFER		A	I/O	7
APPMON-RECEIVE-BUFFER		A	I/O	7
APPMON-LEN-RECEIVE-BUFFER		A	I/O	7
APPMON-LEN-DATA		A	I/O	7
APPMON-RETURN-CODE		A	I/O	7

RPC Communication Area field

Name of the filed in the RPC communication area.

Explanation

Explanation of the purpose of the field.

Req/Opt/Auto

Indicates for input fields whether they have to be given by the RPC application (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 application) or an output field (returned to your RPC application).

Notes:

1. See ***Using Broker Logon and Logoff***.
2. Optional if broker does not require security; required if broker is secured.
3. For RPC conversations. See ***Using Conversational RPC***.
4. Security credentials are relevant if communicating with a Natural RPC server running with Natural Security, or an RPC server running with impersonation; see *Impersonation* in the respect-

ive RPC Server documentation. See also ***Using the COBOL Wrapper with Natural Security and Impersonation***.

5. 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***.
6. See ***Reliable RPC for COBOL Wrapper***.
7. Field is managed internally by the ***Delivered Modules*** themselves. For these to work properly you need to initialize the RPC Communication Area before using it in your RPC client application. See ***Step 1: Declare and Initialize the RPC Communication Area***. Do not change this field in your RPC client application.

Delivered Modules

This section covers the following topics:

- Delivered Modules for z/OS
- Delivered Modules for z/VSE
- Delivered Modules for BS2000/OSD
- Delivered Modules for IBM i
- Adapting the Used Broker Stub

Delivered Modules for z/OS

Module	Data Set	Description	Notes
`COBSRVI`	EXP910.SRCE	CICS generic RPC services with `EXEC CICS LINK` interface.	2
`ERXCOMM`	EXP910.INCL	RPC communication area.	1
`ERXRCSRV`	EXP910.SRCE	C main module for application errors.	3
`ERXRCSRV`	EXP910.LD00	Ready-to-use `ERXRCSRV` module for application errors.	3
`EXPCSRVI`	EXP910.JOBS	JCL to compile the CICS generic RPC service module `COBSRVI` with `EXEC CICS LINK` interface.	2

Module
Name of the delivered module.

EXP910.INCL
Generic RPC include data set. The generic RPC include data set may be delivered as a patch with a different name EXP910.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.

EXP910.SRCE
Generic RPC source data set. The generic RPC source data set may be delivered as a patch with a different name EXP910.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. See ***The RPC Communication Area (Reference)***.
2. See ***Generate Generic RPC Service Module COBSRVI***.
3. See *Aborting RPC Server Customer Code and Returning Error to RPC Client* in the CICS RPC Server documentation.

Delivered Modules for z/VSE

File	Sublibrary	Description	Notes
ERXCOMM	EXP910	RPC Communication area.	1
COBSRVIB.C	EXP910	Batch generic RPC services with call interface (source).	2, 3
COBSRVIB.OBJ	EXP910	Batch generic RPC services with call interface (object).	2, 3
COBSRVIC.C	EXP910	CICS generic RPC services with EXEC CICS LINK interface (source).	2, 3
COBSRVIC.OBJ	EXP910	CICS generic RPC services with EXEC CICS LINK interface (object).	2, 3
COBSRVID.C	EXP910	CICS generic RPC services with call interface (source).	2, 3
COBSRVID.OBJ	EXP910	CICS generic RPC services with call interface (object).	2, 3

File
Name of the delivered file.

Sublibrary
Name of the delivered sublibrary.

Description
Purpose of the file.

Notes:

1. See *The RPC Communication Area (Reference)*.
2. See *Generate Generic RPC Service Module COBSRVI*.
3. We recommend you use module COBSRVI generated by the EntireX Workbench instead of the modules COBSRVIB, COBSRVIC and COBSRVID delivered with your z/VSE installation. The reason for this is that the EntireX Workbench is updated much more frequently. Section *Generate Generic RPC Service Module COBSRVI* under *Generating COBOL Source Files from Software AG IDL Files* explains how to generate the RPC service module.

Delivered Modules for BS2000/OSD

Module	Data Set	Description	Notes
ERXCOMM	EXP811.COBC	RPC communication area.	1
COBSRVI.COB	EXP811.COBC	Batch generic RPC services with call interface.	2, 3

Notes:

1. See *The RPC Communication Area (Reference)*.
2. See *Generate Generic RPC Service Module COBSRVI*
3. We recommend you use module COBSRVI generated by the EntireX Workbench instead of the delivered module. The reason for this is that the EntireX Workbench is updated much more

frequently. Section *Generate Generic RPC Service Module COBSRVI* under *Generating COBOL Source Files from Software AG IDL Files* explains how to generate the RPC service module.

Delivered Modules for IBM i

Module	Source file	Description	Notes
ERXCOMM	QCBLLESRC	RPC communication area.	1
RPCSRVI	QCBLLESRC	Batch generic RPC services with call interface.	2, 3

Notes:

1. See *The RPC Communication Area (Reference)*.
2. See *Generate Generic RPC Service Module COBSRVI*.
3. Do not use module RPCSRVI delivered with your IBM i installation. Use module COBSRVI generated by the EntireX Workbench instead. Section *Generate Generic RPC Service Module COBSRVI* under *Generating COBOL Source Files from Software AG IDL Files* explains how to generate the RPC service module.

Adapting the Used Broker Stub

Because multiple broker stubs may be offered per operating system and environments, it may be necessary to adapt the COBSRVI module to the correct broker stub that supports the required transport (TCP, NET). To do this, modify the COBOL subprogram `DOBROKER` inside the `COBSRVI` source file with a broker stub that meets your requirements.

For availability and information on broker stubs, see *Administering Broker Stubs* under z/OS | UNIX | Windows | BS2000/OSD | IBM i.

Caution: Do not make any modifications other than changing the broker stub name, and do not modify the COBOL subprogram `COBSRVI` inside the same COBSRVI program source. Unexpected behavior will occur.