webMethods EntireX

EntireX PL/I Wrapper

Version 9.6

April 2014

This document applies to webMethods EntireX Version 9.6.

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

Document ID: EXX-EEXXPLIWRAPPER-96-20140628

Table of Contents

EntireX PL/I Wrapper

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

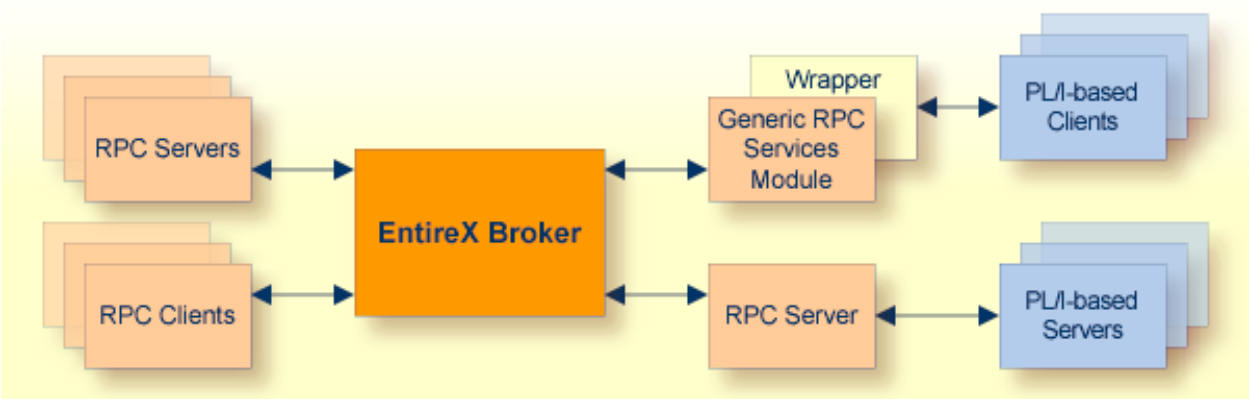
Introduction	Introduction to the PL/I Wrapper.
Using	Step-by-step guide on how to generate interactively and build (write, compile and link) clients and server applications with the PL/I Wrapper. Programming models for batch, CICS and IMS PL/I RPC applications are introduced.
Command-line Mode	Using the PL/I Wrapper in command-line mode.
IDL to PL/I Mapping	Describes the specific mapping of Software AG IDL data types, groups, arrays and structures to the PL/I programming language.
Writing Applications	A step-by-step guide how to write applications with the PL/I Wrapper and an introduction to further advanced topics.
Examples	Describes the delivered PL/I Wrapper example programs.
Reference	Provides reference material for the PL/I Wrapper.
Tracing	Tracing for the PL/I Wrapper.

1 Introduction to the PL/I Wrapper

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

Description

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



The generated functions can be compiled with the PL/I compiler of your target platform.

The PL/I Wrapper works as follows:

- PL/I code is generated from the Software AG IDL file.
- The generic RPC services module implements functionality that is not specific to a given IDL file (e.g., broker logon and logoff). The generated PL/I code makes use of this functionality.
- The Software AG IDL Compiler and an appropriate template are used for the PL/I code generation.

Generic RPC Services Module

In order to minimize the amount of code generated for a specific IDL, all service-type functionality required by the client interface object is implemented in a generic RPC services module.

The generic RPC services module 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:

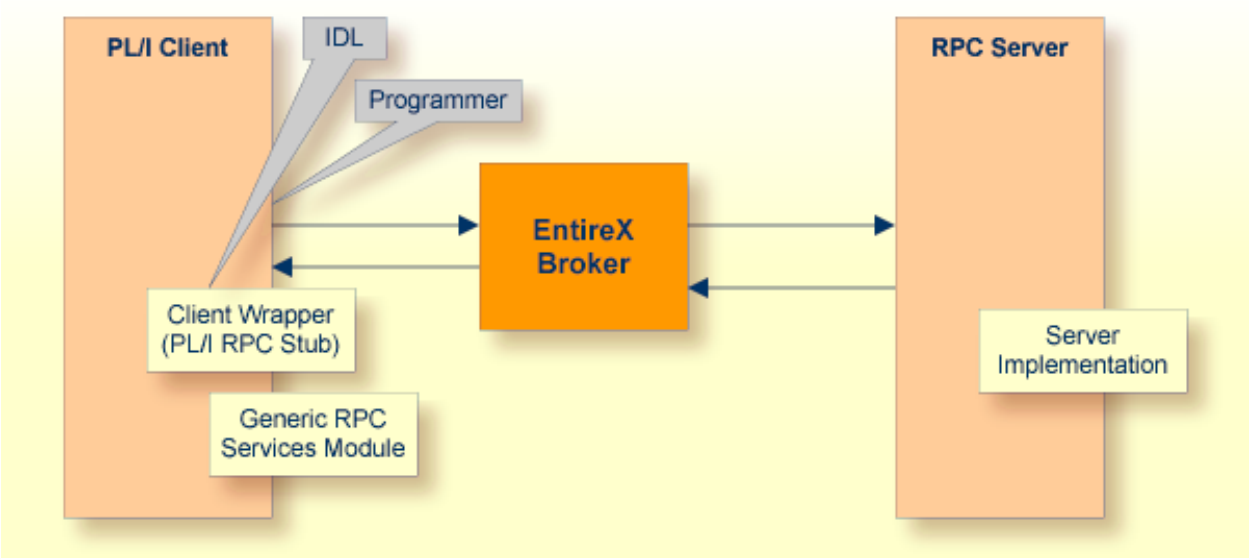
- Broker Logon and logoff

- Conversational support
- Connecting RPC clients to RPC servers via the broker
- etc.

For more information, see *Using the Generic RPC Services Module*.

PL/I Client Applications

For a given IDL file, the Software AG IDL Compiler and a PL/I code generation template for clients are used to generate client interface objects and include files. The source code generated by the PL/I Wrapper can be compiled with your target PL/I compiler. Application developers use the generated client interface object(s) and the include files to write PL/I applications that access RPC servers.

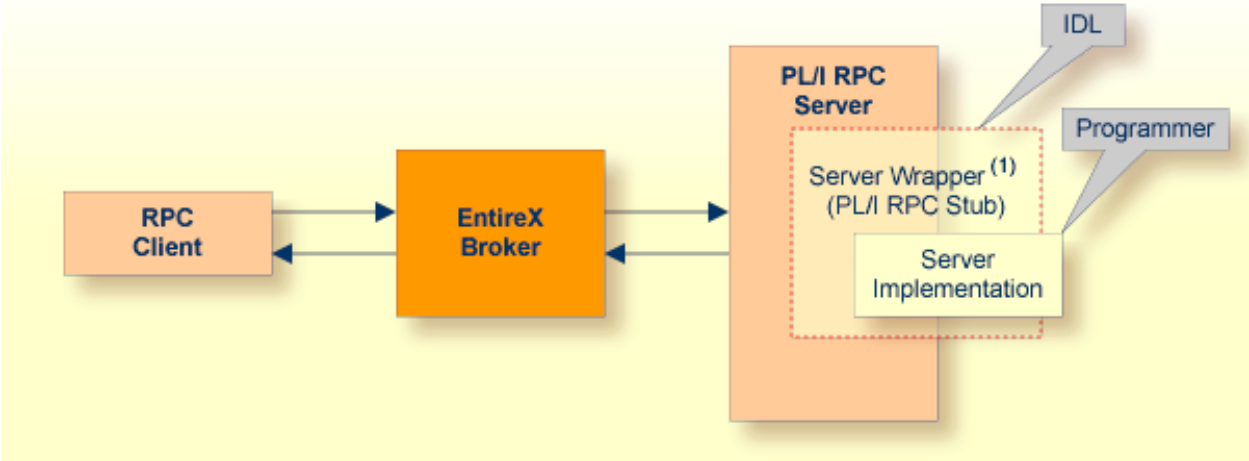


For more information, see *Using the PL/I Wrapper*.

PL/I Server Application

The Software AG IDL Compiler and a PL/I code generation template for servers are used to generate a server (skeletons) for a specific IDL. Most target environments in batch mode and CICS work without interface objects. For IMS, server interface objects are generated too and have to be used.

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 PL/I compiler. Your server program names need to match the program name as specified in the IDL file.



(1) The server wrapper is required for target platform IMS.

For more information, see *Using the PL/I Wrapper*.

2 Using the PL/I Wrapper

Using the PL/I Wrapper for the Client-side

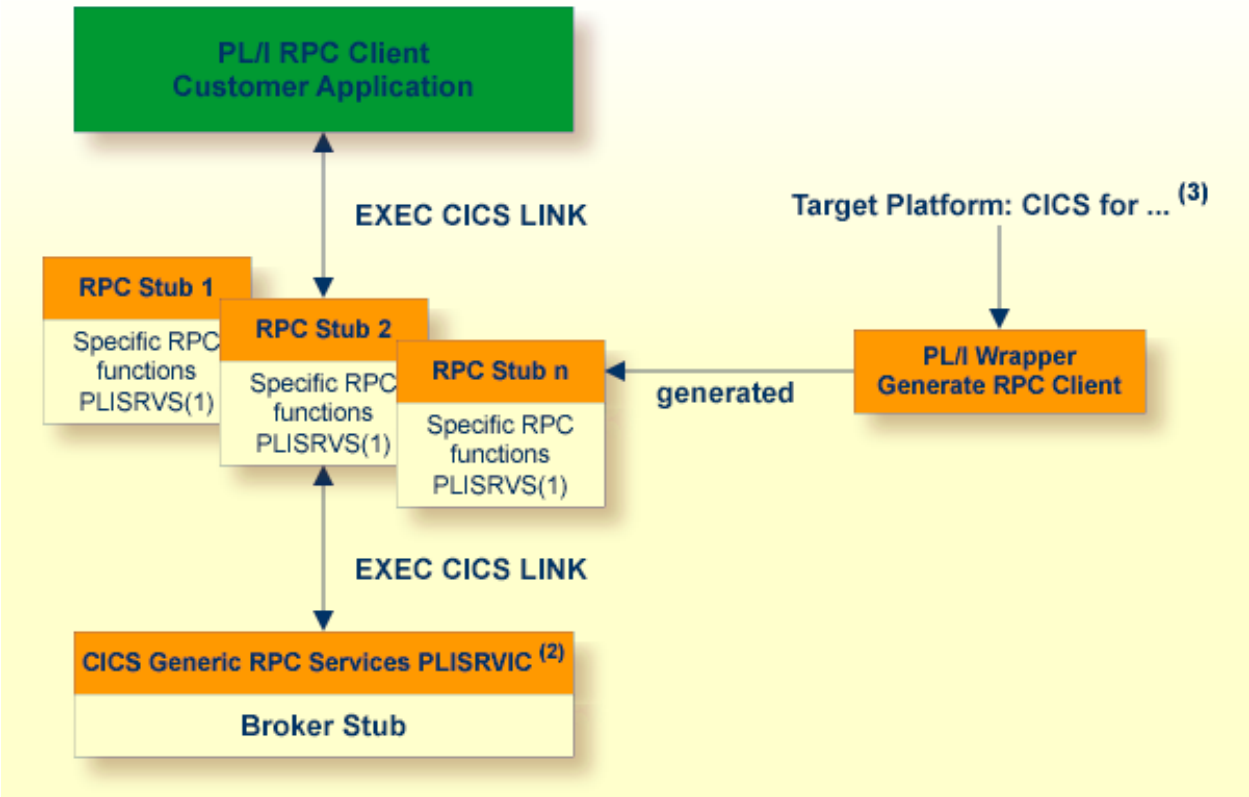
The EntireX PL/I Wrapper provides access to RPC-based components from PL/I applications. It enables you to develop both client and server applications.

This section introduces the various possibilities for RPC-based client applications written in PL/I.

- Using the PL/I Wrapper for CICS
- Using the PL/I Wrapper for CICS with Call Interfaces
- Using the PL/I Wrapper for Batch Mode
- Using the Generic RPC Services Module
- Hints for Compiling and Linking (Binding)
- PL/I Preprocessor Settings

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

Using the PL/I Wrapper for CICS



(1) The linkage name of the module can be customized, see *ERXPREFIX* under ***PL/I Preprocessor Settings***.

(2) The CICS name of the module can be customized, see *ERXPREFIX* under ***PL/I Preprocessor Settings***.

(3) For the target platforms, see ***Generating PL/I Source Files from Software AG IDL Files***.

In this scenario the PL/I RPC Client Customer Application, every generated client interface object and the CICS ***Using the Generic RPC Services Module*** (`PLISRVIC`) are installed each as separate individual CICS programs.

The broker stub is linked together with the CICS Generic RPC Services (`PLISRVIC`) only, thus an update of the broker stub merely requires relinking and reinstallation of the Generic RPC Service module. The client interface objects themselves are not involved.

Use the PL/I Wrapper for CICS if

- you want to have an `EXEC CICS LINK` interface to your RPC stubs,
- you wish to separate the broker stub from the client interface objects(s)

- you require a distributed program link to the client interface object(s).

To use the PL/I Wrapper for CICS

1 Generate the RPC stub for the target, e.g. "CICS for z/OS". See *Generating PL/I Source Files from Software AG IDL Files*.

2 If necessary, use FTP to transfer the client interface object to the target platform where you write your application.

3 Write your PL/I RPC client application (see *Writing Applications with the PL/I Wrapper*), taking into consideration the manner in which CICS programs are called.

4 If necessary, use FTP to transfer the client interface object and your application to the target platform where you translate and compile your application.

5 Set the preprocessor switch `ERXTARGET` in file `RPCPPS` (*PL/I Preprocessor Settings*) to "`CICS`".

6 Using the CICS translator for PL/I provided with your CICS installation and a PL/I compiler supported by PL/I Wrapper, translate and compile the following:

- the generated client interface object(s)
- the specific RPC functions module (`PLISRVS`)
- the CICS Generic RPC Service module (`PLISRVIC`)
- your PL/I RPC client customer application

Note the *Hints for Compiling and Linking (Binding)*.

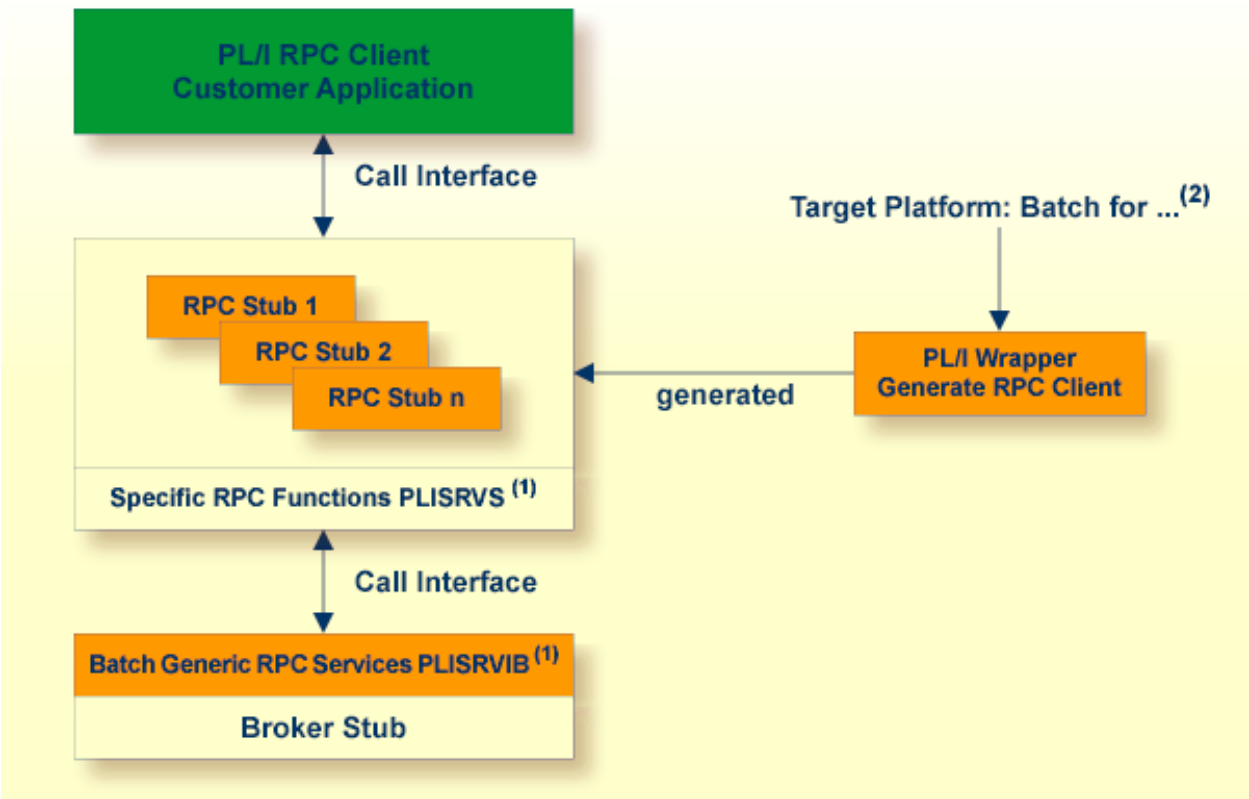
7 Link (bind) every generated client interface object together with thespecific RPC functions module (`PLISRVS`) to a CICS program, using the standard linker (binder) of the target platform.

8 Link (bind) the CICS RPC service module (`PLISRVIC`) together with the broker stub to a CICS program, using the standard linker (binder) of the target platform. Use a broker stub supported in CICS. The CICS name of the module is `PLISRVI` and it can be customized, see `ERXPREFIX` under *PL/I Preprocessor Settings*.

9 Link (bind) your PL/I RPC client customer application, using the standard linker (binder) of the target platform.

10 Install every client interface object, the CICS RPC service module and your PL/I RPC client customer application from the previous steps as separate CICS programs.

Using the PL/I Wrapper for CICS with Call Interfaces



(1) The linkage name of the module can be customized, see *ERXPREFIX* under ***PL/I Preprocessor Settings***

(2) For the target platforms, see ***Generating PL/I Source Files from Software AG IDL Files***

The PL/I Wrapper could be used with a call interface, even in CICS. This means you build an application where the PL/I RPC client customer application, every generated client interface object, the specific RPC functions module (`PLISRVS`), the Batch ***Using the Generic RPC Services Module*** (`PLISRVIB`) and the broker stub are linked together to an executable application, similar to the ***Batch*** scenario.

Using a call interface within CICS may be useful if

- the restriction of the COMMAREA length (about 31 KB) prevents you from using the ***CICS*** scenario
- you prefer a call interface instead of `EXEC CICS LINK`

To use the PL/I Wrapper with a call interface within CICS

1 Generate the client interface object for the target, e.g. "Batch for z/OS". See *Generating PL/I Source Files from Software AG IDL Files*.

2 If necessary, use FTP to transfer the client interface object to the target platform where you write your application.

3 Write your PL/I RPC client application. See *Writing Applications with the PL/I Wrapper*.

4 If necessary, use FTP transfer the client interface object and your application to the target platform where you translate and compile your application.

5 Set the preprocessor switch `ERXTARGET` in file `RPCPPS` (*PL/I Preprocessor Settings*) to "`BATCH`".

6 Using the CICS translator for PL/I provided with your CICS installation and a PL/I compiler supported by PL/I Wrapper, translate and compile the following:

- the generated client interface object(s)
- the specific RPC functions module (`PLISRVS`)
- the Batch RPC Service module (`PLISRVIB`)
- PL/I RPC client customer application

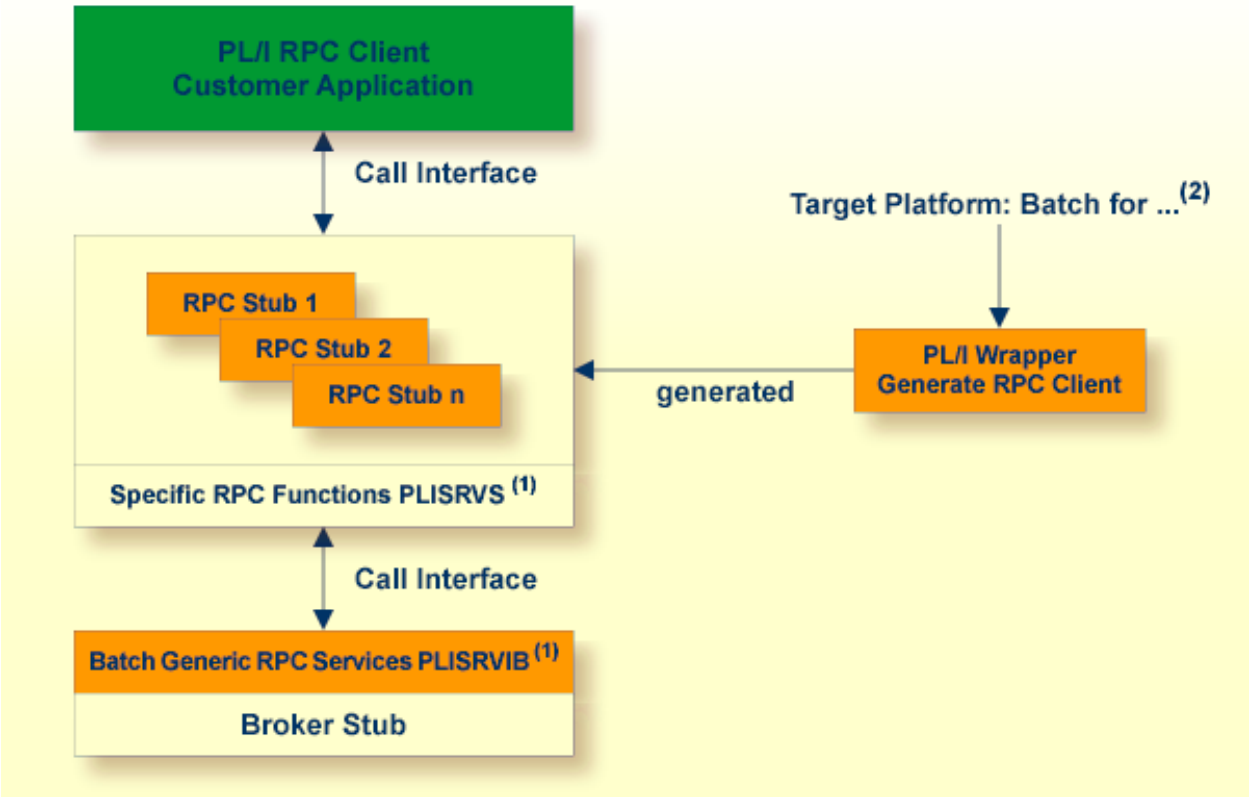
See *Hints for Compiling and Linking (Binding)*.

7 Link (bind) all translated and compiled modules together with the broker stub to a CICS program, using the standard linker (binder) of the target platform. Use a broker stub supported in CICS.

8 Install the program within CICS.

Using the PL/I Wrapper for Batch Mode

This mode applies to z/OS.



(1) The linkage name of the module can be customized, see `ERXPREFIX` under *PL/I Preprocessor Settings*.

(2) For the target platforms see *Generating PL/I Source Files from Software AG IDL Files*.

In this scenario, the PL/I RPC client customer application, every generated client interface object, the specific RPC functions module (`PLISRVS`), the Batch *Using the Generic RPC Services Module* (`PLISRVIB`) and the broker stub are linked together to an executable application.

To use the PL/I Wrapper for Batch

1 Generate the client interface objects for the target, e.g. "Batch for z/OS". See *Generating PL/I Source Files from Software AG IDL Files*.

2 If necessary, use FTP to transfer the client interface object to the target platform where you write your application.

3 Write your PL/I RPC client application. See *Writing Applications with the PL/I Wrapper*.

4 If necessary, use FTP to transfer the client interface object and your application to the target platform where you compile your application.

5 Set the preprocessor switch `ERXTARGET` in file `RPCPPS` (*PL/I Preprocessor Settings*) to "`BATCH`".

6 Compile the following, using a PL/I compiler supported by PL/I Wrapper:

- the generated client interface object(s)
- the specific RPC functions module (`PLISRVS`)
- the batch generic RPC service module (`PLISRVIB`)
- your PL/I RPC client customer application

Note the ***Hints for Compiling and Linking (Binding)***.

7 Link (bind) all compiled modules together with the broker stub to an executable program, using the standard linker (binder) of the target platform. Use a broker stub supported for batch processing in your environment.

Using the Generic RPC Services Module

The generic RPC services module contains the call to the broker stub, as well as other functions needed for RPC communication where a client interface object is not needed, such as

- Logon to broker LO and Logoff from broker LF. See ***Using Broker Logon and Logoff***.
- Open Conversation OC, Close Conversation CB and Close Conversation with Commit CE. See ***Conversational RPC***.
- Create a Natural Security Token. See ***Using Natural Security***.

Depending on your target environment the generic RPC services module is delivered in various sources:

- For ***CICS***, use the source `PLISRVIC`. This module is shared by all PL/I RPC client applications because it is installed only once within CICS.
- For ***Batch*** and ***CICS with Call Interfaces*** use the source `PLISRVIB`.

The program and linkage name of the generic RPC service module

- is `PLISRVI` by default
- does not depend on the source module used (`PLISRVIB` or `PLISRVIC`)
- can be customized, see ***PL/I Preprocessor Settings***

The delivery of the generic RPC service module and other required modules depends on the platform you are using.

- For platform z/OS, see ***PL/I Wrapper Modules Delivered as Sources on z/OS***.

Hints for Compiling and Linking (Binding)

- The delivered generic RPC include data set EXP960.INCL (this data set may be delivered as a patch with a different name EXP960.IN*nn*, where *nn* is the patch level number, make sure you install the highest patch level available) is required to SYSLIB input for the PL/I compiler.
- For a non-tracing version, the Trace Functions module delivered in the source `PLISRVT` in the generic RPC source data set EXP960.SRCE must *not* be compiled and linked to your application.

PL/I Preprocessor Settings

The PL/I Wrapper requires the PL/I preprocessor for all scenarios, that is, the templates generate source code including preprocessor statements. See your PL/I compiler documentation on how to switch on the PL/I preprocessor.

The preprocessor settings are customized in the file `RPCPPS`; see ***PL/I Wrapper Modules Delivered as Sources on z/OS***. The following switches are available:

Preprocessor Switch	Values	Explanation
`ERXTARGET`	`BATCH | CICS`	This is the most important switch. The setting must always be the same as the target option during generation, i.e. when using the PL/I Wrapper ■ for Batch, it *must* be set to "`BATCH`" ■ for CICS, it *must* be set to "`CICS`" ■ for CICS with call interfaces, it *must* be set to "`BATCH`" If the setting is not the same, it will *not* be possible to compile the PL/I Wrapper successfully.
`ERXPREFIX`	A prefix composed of any three characters valid for PL/I names	With this switch you can customize a prefix of the program and linkage name of the ■ Specific RPC Functions module (`xxxSRVS`) ■ Generic RPC Services module (`xxxSRVI`) ■ Trace Functions module (`xxxSRVT`) The default of the prefix is PLI, the resulting linkage names therefore are `PLISRVS`, `PLISRVI` and `PLISRVT`. In the scenario of CICS the prefix also customizes the CICS program name (`xxxSRVI`) for the Generic RPC Services module.
`ERXTRACE`	YES NO	With this switch you can build a trace version, See ***Using Trace***. For a non-tracing version, if set to “NO”, the Trace Functions module delivered in source `RPCSRVT` *must not* be compiled and linked to your application.

Using the PL/I Wrapper for the Server Side

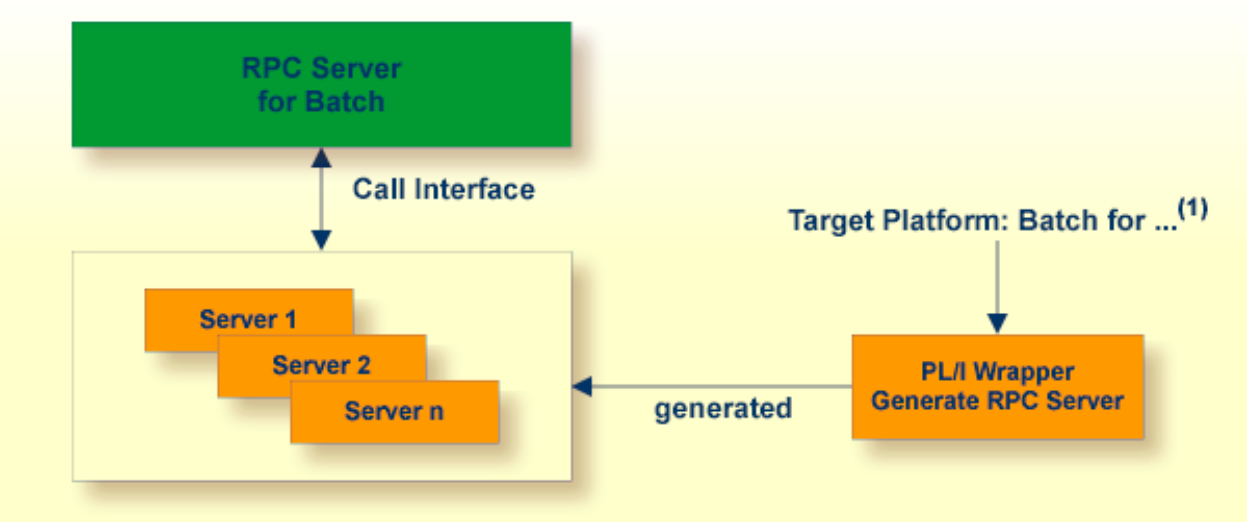
The EntireX PL/I Wrapper provides access to RPC-based components from PL/I applications. It enables you to develop both client and server applications.

This section introduces the various possibilities for RPC-based server applications written in PL/I.

- Using the PL/I Wrapper for Batch Mode
- Using the PL/I Wrapper for CICS
- Using the PL/I Wrapper for IMS BMP
- Hints for Compiling and Linking (Binding)

Using the PL/I Wrapper for Batch Mode

This scenario applies to z/OS. See also *PL/I Scenarios* under *Scenarios and Programmer Information*.



(1) For the target platforms, see *Generating PL/I Source Files from Software AG IDL Files*

In batch mode the RPC server requires no server interface objects. All parameters of your server are set up dynamically in the format required. Your server is called dynamically using standard call interfaces.

To use the PL/I Wrapper for batch

1 Generate the server (skeleton) for the target, for example "Batch for z/OS". See *Generating PL/I Source Files from Software AG IDL Files*.

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

3 Use the generated server (skeleton) and complete it by applying your application logic. Note the information given in *Software AG IDL to PL/I Mapping*.

4 If necessary, use FTP to transfer the server interface object and your server to the target platform where you compile your server.

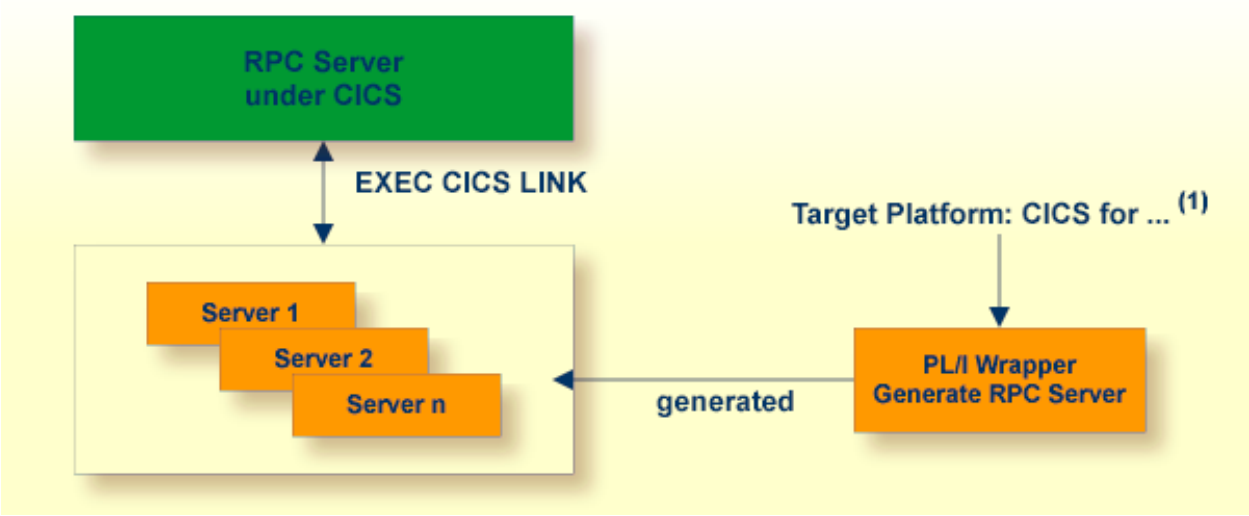
5 Compile it using a PL/I compiler supported by PL/I Wrapper on your server.

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

7 Provide the server and the server interface object to the RPC Server for Batch.

Using the PL/I Wrapper for CICS

This mode applies to z/OS. See also *PL/I Scenarios* under *Scenarios and Programmer Information* in the CICS RPC Server documentation.



(1) For the target platforms, see *Generating PL/I Source Files from Software AG IDL Files*

In CICS the RPC Server requires no server interface objects. All parameters of your server are set up dynamically in the format required. Your server is called using `EXEC CICS LINK`.

To use the PL/I Wrapper for CICS

1 Generate the server (skeleton) for the target, e.g. "CICS for z/OS". See *Generating PL/I Source Files from Software AG IDL Files*.

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

3 Use the generated server (skeleton) and complete it by applying your application logic. Note the information given under *Software AG IDL to PL/I Mapping* and *Aborting RPC Server*

Customer Code and Returning Error to RPC Client under *Scenarios and Programmer Information* in the CICS RPC Server documentation.

4 If necessary, use FTP to transfer the server interface object and your server to the target platform where you translate and compile your server.

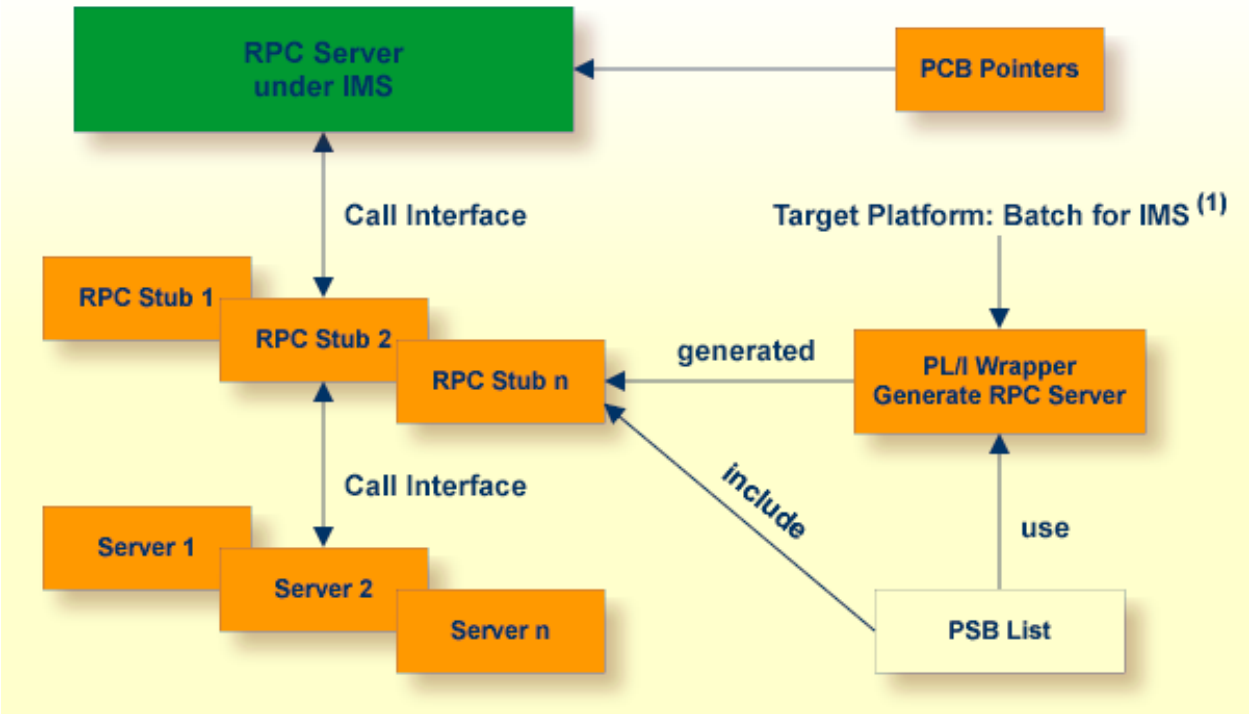
5 Translate and compile your server (including your application logic) using the CICS translator for PL/I provided with your CICS installation and a PL/I compiler supported by PL/I Wrapper.

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

7 Install your server as a CICS program to provide it to the CICS RPC Server.

Using the PL/I Wrapper for IMS BMP

This mode applies to z/OS IMS mode BMP. See also *PL/I Scenarios* under *Scenarios and Programmer Information* in the IMS RPC Server documentation



(1) For the target platforms, see *Generating PL/I Source Files from Software AG IDL Files*

In IMS, the RPC Server works with server interface objects. The interface objects provide the IMS-specific PCB pointers to your server. Your server is called dynamically using standard call interfaces. See *IMS-specific PCB Pointers* in the IMS RPC Server documentation.

PSB List

All IMS PCB pointers are listed in a so-called PSB list, an include file containing the PCB pointers as PL/I declarations.

Example

Assume a program uses two PSB definitions:

- IOPCB for the terminal interactions to check whether all outstanding messages are displayed or not
- DBPCB for all database interactions to read, add, delete or update segments in a given database.

Technically the PLI / DLI interface module PLITDLI is called with the selected function code and PCB parameters. On return the necessary status information can be checked to control the further processing within the application program.

This status information is provided as parameter on return of the PLITDLI call. For this purpose, the PCB areas have to be defined with a DECLARE statement and as parameters.

```
   DCL 01 IOPCB                          ,
         02 LTERM-NAME         CHAR (08) ,
         02 FILLER_IO1         CHAR (02) ,
         02 TPSTATUS           CHAR (02) ,
         02 FILLER_IO2         CHAR (20);
   DCL 01 DBPCB                          ,
         02 DBNAME             CHAR (08) ,
         02 SEG_LEVEL_NO       CHAR (02) ,
         02 DBSTATUS           CHAR (02) ,
         02 FILLER             CHAR (20);

DBREAD: PROC(IOPCB, DBPCB) OPTIONS(...)
```

Now IMS is able to pass ADDRESSES for the IOPCB and DBPCB on entry of the application program.

The PSB list would be:

```
10  IOPCB POINTER,
10  DBPCB POINTER;
```

In the application program itself the IMS parameter can only be identified by its name, thus it is necessary to provide a list of PCBs as an include file.

This include file is provided by the customer.

- The PSB list must be named PSBLIST (otherwise generation will not be possible) and it must not be empty, it must contain at least 1 PCB pointer.

- It is used by the *IDL Extractor for PL/I* to find the IMS-specific PCB pointers. In the IDL file, parameters originating from PCB pointers are marked with the attribute `IMS`.
- It is completed in the generated server interface object with the following level-1 parameter name, to address the IMS-specific PCB pointers correctly:

```
/* IMS parameter */
DCL 1 IMS_PARAMETER,
%INCLUDE PSBLIST;
```

- Together with the level-1 parameter name the data declarations in the PSB list include file must form valid PL/I data declarations - otherwise compilation is not possible.

To use the PL/I Wrapper for IMS

1. Generate the server (skeleton) and server interface objects for the target "IMS for z/OS". See ***Generating PL/I Source Files from Software AG IDL Files***.
2. If necessary, use FTP to transfer the server interface object and the server (skeleton) to the target platform where you write your server.
3. Use the generated server (skeleton) and complete it by applying your application logic. You can use the IMS specific PCB pointers in your server as usual. Note the information under ***Software AG IDL to PL/I Mapping*** and *Aborting RPC Server Customer Code and Returning Error to RPC Client* under *Scenarios and Programmer Information* in the IMS RPC Server documentation.
4. If necessary, use FTP to transfer the server interface object and your server to the target platform where you compile your server.
5. Using a PL/I compiler supported by PL/I Wrapper, compile:
   - the generated server interface object(s) providing the ***PSB List*** (see above) as an include file
   - your server (including your application logic).

   See ***Hints for Compiling and Linking (Binding)***.
6. Link (bind) the server to an executable program, using the standard linker (binder) of the target platform.
7. Provide the server interface object(s) in the server interface object library with the parameter `stublib` to the IMS RPC Server. See *Configuring the RPC Server* under *Administering the EntireX RPC Server under z/OS IMS*.
8. Provide the server to the IMS RPC Server.

Hints for Compiling and Linking (Binding)

For **IMS**

- The delivered generic RPC include data set EXP960.INCL (this data set may be delivered as a patch with a different name EXP960.IN*nn*, where *nn* is the patch level number; make sure you install the highest patch level available) is required to SYSLIB input for the PL/I compiler.

For **all other platforms**

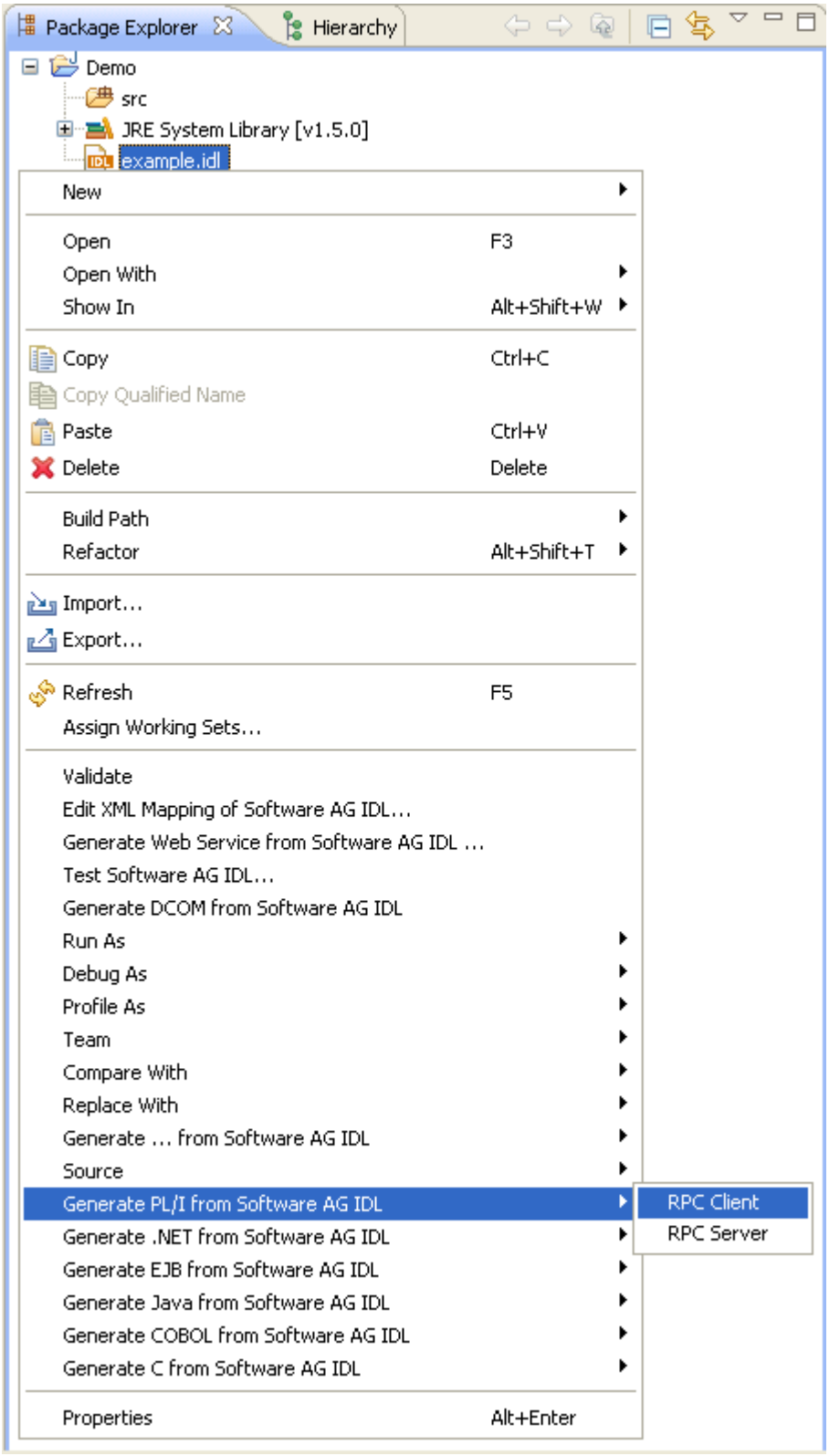
- No special considerations apply.

Generating PL/I Source Files from Software AG IDL Files

This section describes how to generate PL/I source files from Software AG IDL files.

Selecting an IDL File and Generating an RPC Client or RPC Server

From the context menu, choose **Generate PL/I from Software AG IDL > RPC Client** and **... > RPC Server** to generate the PL/I source files.



For the **RPC client**,

- this creates the folders *client* and *include* in the *Container* folder, defined in the properties. These contain the client interface objects and the RPC client declarations.
- In command-line mode, use the command `-"pli:client"`. See ***Using the PL/I Wrapper in Command-line Mode***

For the **RPC server**,

- this creates the folder *server* in the *Container* folder, defined in the properties. It contains the RPC server implementation skeletons.
- Additionally, server interface objects may be generated depending on the platform, e.g. “IMS for z/OS” and features chosen in the folder *serverstub* in the *Container* folder, defined in the properties. It contains the server interface objects.
- In command-line mode, use command `-pli:server`. See ***Using the PL/I Wrapper in Command-line Mode***.

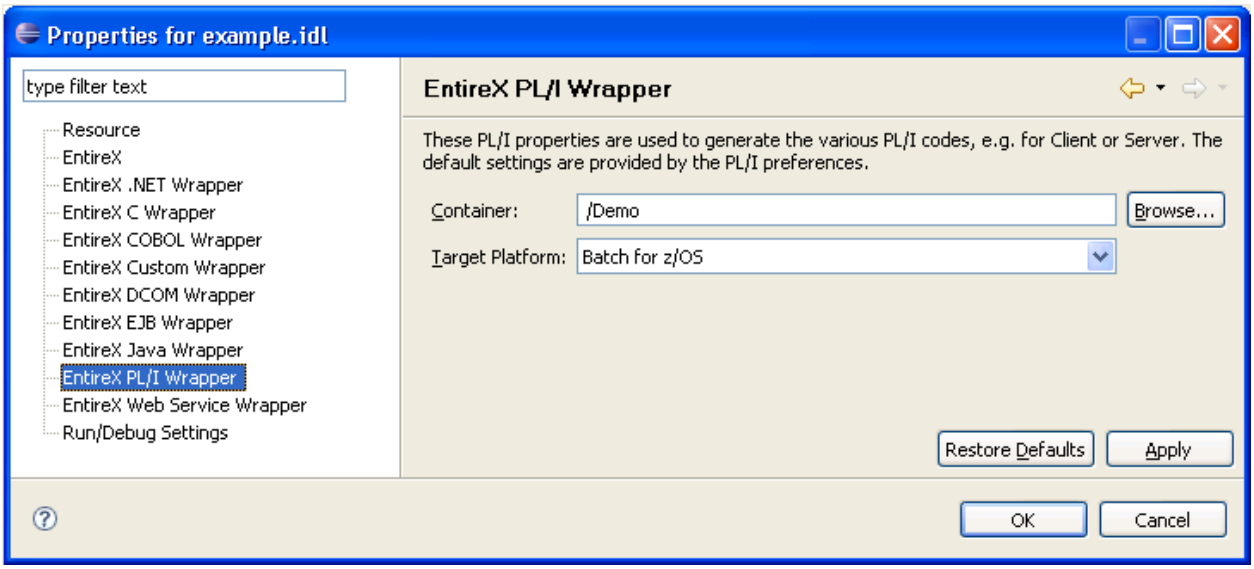
Caution: Take care not to overwrite an existing server implementation with a server skeleton.We recommend you move your server implementation to a different folder.

For both **RPC client** and **RPC server**

- If you generate using the GUI and generated files exist from a previous generation, you are prompted to overwrite them.
- If you generate using command-line mode, existing files are always overwritten.

Settings

Use the properties of the IDL file to set the container folder where the source file subfolders will be stored. The target platform setting is responsible for the file extension and the content of the generated files.

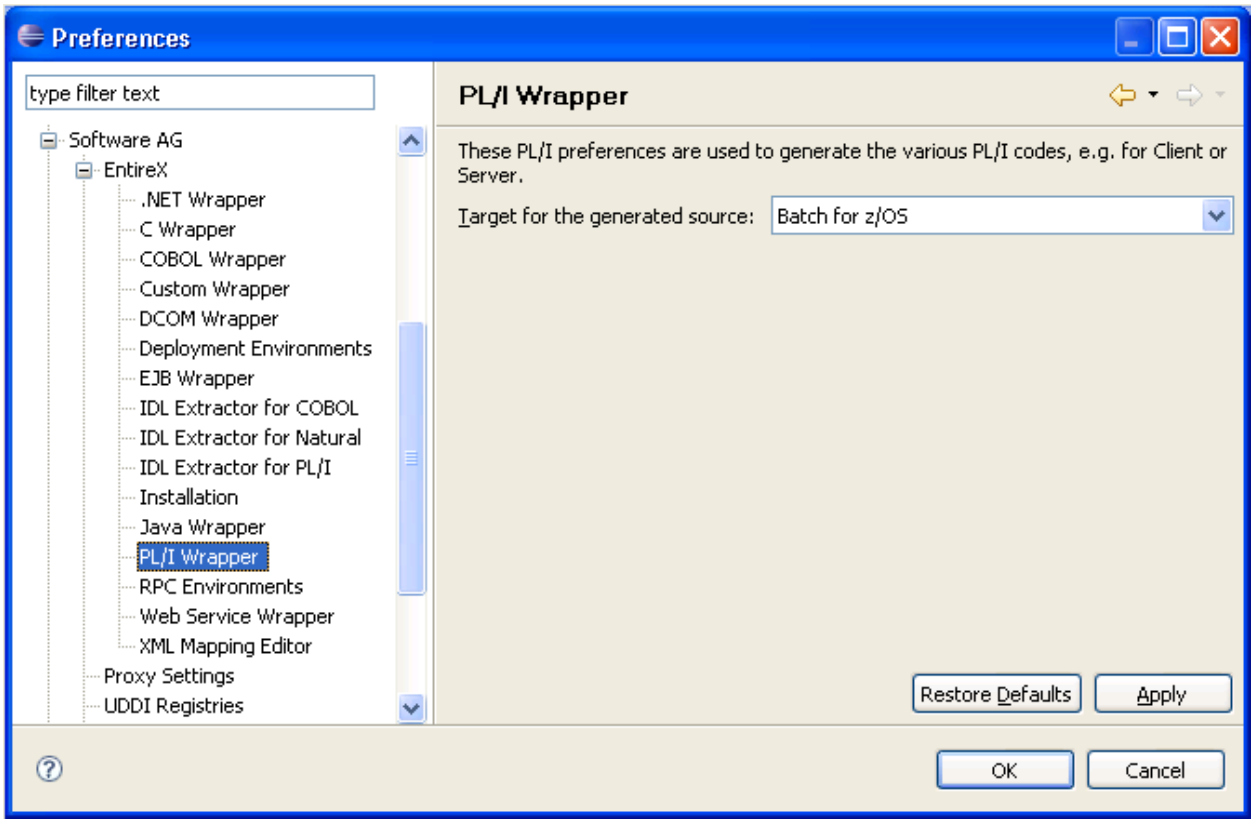


Target Platform

Select Batch, TP Monitor and Operating System for which PL/I code is to be generated.

Target	Description
Batch for z/OS	Batch-dependent PL/I code will be generated. Interface objects and servers are called using standard call interface. Use this option if you want to build an RPC application as described under ■ *Using the PL/I Wrapper for Batch Mode* (***client*** or ***server***). ■ *Using the PL/I Wrapper for CICS with Call Interfaces* (***client***).
CICS for z/OS	CICS-dependent PL/I code will be generated. The interface is mapped to `DFHCOMMAREA`. Interface objects and servers are called using `EXEC CICS LINK`. Use this option if you want to build an RPC application as described under ■ *Using the PL/I Wrapper for CICS* (***client*** or ***server***).
IMS for z/OS	IMS-dependent PL/I code will be generated. Interface objects and servers are called using standard call interface, considering also IMS-specific PCB pointers (IDL parameters marked with the attribute `IMS`) for servers. See `attribute-list` under *Software AG IDL Grammar* in the *IDL Editor* documentation. Use this option if you want to build an RPC application for IMS as described under ■ *Using the PL/I Wrapper for Batch Mode* (***client***) ■ *Using the PL/I Wrapper for IMS BMP* (***server***)

The workspace default for the target platform is set in the preferences page of the PL/I Wrapper.



In command-line mode, use `-target` to set this option.

3 Using the PL/I Wrapper in Command-line Mode

Command-line Options

See *Using the EntireX Workbench in Command-line Mode* for the general command-line syntax. The table below shows the command-line options for the PL/I Wrapper.

Task	Command	Option	Description
Generate a PL/I client from the specified IDL file.	`-pli:client`	`-folder`	Folder where the PL/I files will be stored.
		`-help`	Display this usage message.
		`-target`	Target platform, one of `BATCH_ZOS`, `CICS_ZOS`, `IMS_ZOS`. See ***Target Platform*** for more information.
Generate a PL/I server from the specified IDL file.	`-pli:server`	`-folder`	Folder where the PL/I files will be stored.
		`-help`	Display this usage message.
		`-target`	Target platform; for more information, see above.

Example Generating an RPC Client

```
<workbench> -pli: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 Software AG 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 PL/I source files (client interface objects and the RPC 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> -pli:server /Demo/example.idl -target IMS_ZOS
```

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

The generated PL/I source files (server interface objects and the server (skeletons))

- will be stored in parallel to the Software AG IDL file, in the generated subfolders *server* and *server_stubs*, e.g. *Demo/server* and *Demo/server_stubs*.
- 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 move your server implementation to a different folder.

Further Examples

Windows

Example 1

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

Uses the IDL file *C:\Temp\example.idl* and generates the PL/I 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 PL/I client wrapper with C:/Temp/example.idl and target CICS_ZOS.
Processing IDL file C:/Temp/example.idl
Store PL/I Source (1/2): C:\Temp\src/include/CALC
Store PL/I Source (2/2): C:\Temp\src/client/CALC
Exit value: 0
```

Example 2

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

Generates PL/I source files for all IDL files in *C:\Temp*.

Example 3

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

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

Example 4

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

or

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

Both show a short help for the PL/I client wrapper.

Linux

Example 1

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

Generates PL/I client files 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> -pli:client -help
```

or

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

Both show a short help for the PL/I client wrapper.

4 Software AG IDL to PL/I Mapping

This chapter describes the specific mapping of Software AG IDL data types, groups, arrays and structures to the PL/I programming language. See also remarks and hints on the Software AG IDL data types valid for all language bindings under *Software AG IDL File* in the IDL Editor documentation.

Mapping IDL Data Types to PL/I Data Types

The following metasymbols and informal terms are used for the IDL in the table below.

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

Software AG IDL	Description	PL/I Data Type	Notes	Client Support	Server Support
A*number*	Alphanumeric	CHAR(*number*)	1	x	x
		CHAR(*)	1, 2		
AV	Alphanumeric variable length	not supported			
AV[*number*]	Alphanumeric variable length with maximum length	CHAR(*number*) VAR	1, 16	x	x
		CHAR(*) VAR	1,2		
B*number*	Binary	BIT(*number* * 8)	3, 15	x	x
		BIT(*)	3, 2		
BV	Binary variable length	not supported			
BV[*number*]	Binary variable length with maximum length	BIT(*number**8) VAR	3, 15		x
D	Date	CHAR(8)DATE('YYYYMMDD')	4	x	x
F4	Floating point (small)	FLOAT DEC(6)	5, 10	x	x
		FLOAT BIN(21)	2, 5, 10		
F8	Floating point (large)	FLOAT DEC(16)	5, 10	x	x
		FLOAT BIN(53)	2, 5, 10		
I1	Integer (small)	BIN FIXED(7)	12	x	x
I2	Integer (medium)	BIN FIXED(15)		x	x
I4	Integer (large)	BIN FIXED(31)		x	x
K*number*	Kanji	GRAPHIC (*number*/2)	6	x	x
		GRAPHIC (*)	2, 6		

Software AG IDL	Description	PL/I Data Type	Notes	Client Support	Server Support
KV	Kanji variable length	not supported			
KV[*number*]	Kanji variable length with maximum length	GRAPHIC (*number*/2) VAR	6, 16	x	x
		GRAPHIC (*) VAR	2, 6		
L	Logical	BIT(1)	7, 14	x	x
N*number1*[.*number2*]	Unpacked decimal	PIC 'S(*number1*)9[V(*number2*)9]'	13	x	x
NU*number1*[.*number2*]	Unpacked decimal unsigned	PIC '(*number1*)9[V(*number2*)9]'	13	x	x
P*number1*[.*number2*]	Packed decimal	DEC FIXED (*number1*+*number2*,*number2*)	8	x	x
PU*number1*[.*number2*]	Packed decimal unsigned	DEC FIXED (*number1*+*number2*,*number2*)	8, 9	x	x
T	Time	CHAR(17)	11	x	x
U*number*	Unicode	WIDECHAR(*number*)		x	x
UV	Unicode variable length	not supported			
UV*number*	Unicode variable length with maximum length	WIDECHAR(*number*) VAR		x	x

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

Notes:

1. The maximum length is restricted by the PL/I programming language, usually 32767 characters, i.e. 32767 bytes in the IDL. A warning message is produced during the generation process if this limit is exceeded.
2. This form is an alternative PL/I mapping for the server side. It is not directly generated by the PL/I Wrapper. The RPC Server source file can be manually modified.
3. The maximum length is restricted by the PL/I programming language, usually 32767 bits, i.e. 4095 bytes in the IDL (the data type length in PL/I is a multiple of 8 of the length given in the IDL). A warning message is produced during the generation process if this limit is exceeded.
4. The format for date is compatible with the date part of the built-in PL/I DATETIME function.
5. When floating-point data types are used, rounding errors can occur, so that the values of senders and receivers might differ slightly.
6. For IDL data type K and KV, graphic support - also known as DBCS support - *must be* switched on in your PL/I compiler; see your compiler documentation on how to switch on graphic support. The maximum length for graphic is restricted by the PL/I programming language, usually 16383

graphics, i.e. 32766 bytes in the IDL (the graphic data type length in PL/I is half the length given in the IDL, therefore the length given in the IDL *must be* even). A warning message is produced during the generation process if this limit is exceeded.

7. The logical values are defined as '1'b for true and '0'b for false. We recommend using the values `ERX_TRUE` and `ERX_FALSE` defined in the include file `RPCAPI`. See ***PL/I Wrapper Modules Delivered as Sources on z/OS***.
8. The maximum number `(number1+number2)` of digits is restricted by your PL/I environment, usually 15 or 31 depending on the PL/I compiler in use. See your compiler documentation. The number of digits after the decimal point (number2) is restricted by the IDL. The value range of -127 to 128 of PL/I for number2 is not supported. See *IDL Data Types* under *Software AG IDL File* in the IDL Editor documentation.
9. Negative values cannot be sent by clients and will be rejected on a send.
10. For servers, the “typical” platform-dependent representation of `FLOAT` will be used: On S/390 systems this is "Hexadecimal floating-point".
11. The value for time has the form YYYYMMDDHHIISST00 compatible with the PL/I `DATETIME` function. The fractional part of a second (hundredths and thousandths of a second) is restricted by the IDL. See *IDL Data Types* under *Software AG IDL File* in the IDL Editor documentation. On receive, hundredths and thousandths are set to zero, whereby on a send they are cut off.
12. For the stubless RPC Server the compiler must use a 1-byte field for the target PL/I data type. It is not supported by the PL/I for MVS & VM V1R1.1 compiler. For PL/I Wrapper clients there are no restrictions; any compiler can be used.
13. The maximum number of digits `(number1+number2)` is restricted by your PL/I environment, usually 15 or 31 depending on the PL/I compiler in use. See your compiler documentation.
14. For the server side the data type L is aligned, even if no `ALIGNED` attribute is given in the IDL. The data type L used by servers without alignment is not supported.
15. For the server side the data type B is aligned, even if no `ALIGNED` attribute is given in the IDL. The data type B used by servers without alignment is not supported.
16. For the client side the IDL datatype may produce a warning message during compilation, which can be ignored. Linkage and execution are possible. For the PL/I for MVS & VM V1R1.1 Compiler, the warning message is IEL0872 'ADDR' BUILTIN FUNCTION RETURNS A POINTER TO THE TWO-BYTE LENGTH FIELD PRECEDING THE VARYING STRING VALUE.

Mapping Library Name and Alias

The library name is sent from a client to the server. Special characters are not replaced. The library alias is not sent to the server. In the RPC server the library name sent may be used to locate the target server. See *Locating and Calling the Target Server* in the platform-specific administration or RPC server documentation.

Client Side

For a batch interface (`TARGET=BATCH_xxx`)

- the library alias as given in the library-definition of the IDL File is used to compose the name of an include file with `DECLARE ENTRY` statements to describe the PL/I generated interfaces. The generated name is `<library-alias>`. If no library alias is given, the library name as given in the library definition of the IDL file is used instead. See `library-definition` under *Software AG IDL Grammar* in the *IDL Editor* documentation.

Server Side

The library name sent along with the client RPC request is not used in PL/I servers. See *Locating and Calling the Target Server* in the platform-specific administration or RPC server documentation.

Mapping Program Name and Alias

The program name is sent from a client to the server. Special characters are not replaced. The program alias is not sent to the server.

In the RPC server the program name sent is used to locate the target server. See *Locating and Calling the Target Server* in the platform-specific administration or RPC server documentation.

Client Side

The program alias names as given in the `program-definition` under *Software AG IDL Grammar* in the *IDL Editor* documentation of the IDL file are

- mapped to procedure or function names within the generated PL/I sources.
- used to compose the file names of the generated output files. Therefore they must be names that are supported by the underlying file system.

When building procedure, function and source file names, the special characters '&', '+', '-', '.' and '/' are replaced by the character underscore '_' valid for PL/I names. Other special characters used

in the program alias name are not changed and may lead to compilation errors when compiling the generated sources.

For a batch interface (`TARGET=BATCH_xxx`)

- An include file and a source file are generated for every `program-definition` under *Software AG IDL Grammar* in the *IDL Editor* documentation given in the IDL. The generated names for both are <program-alias>. When the program alias is not given in the program-definition the program name is used instead.

For a CICS interface (`TARGET=CICS_xxx`)

- A source file is generated for every `program-definition` under *Software AG IDL Grammar* in the *IDL Editor* documentation given in the IDL. The generated name is <program-alias>. When the program alias is not given in the program-definition the program name is used instead.

Example:

- A program name of `#HU$GO` in the IDL results in `#HU$GO_` as the procedure name for the PL/I programming language and source file names `#HUG$GO` and `#HUG$GO`.

Server Side

There is no program name mapping on the server side. The RPC Server for PL/I calls the server using the program name sent along with the client RPC request. See *Locating and Calling the Target Server* in the platform-specific administration or RPC server documentation.

Mapping Parameter Names

Client and Server Side

When building parameter names the special characters '&', '+', '-', '.' and '/' allowed within names of parameters of the IDL, are mapped to the character underline '_' valid for PL/I names. The characters '#', '$' and '@' are allowed in PL/I and are not changed. Trailing and leading special characters are not removed.

Example

`HU&GO` results in `HU_GO` as a valid PL/I parameter name.
`&HUGO-` results in `_HUGO_` as a valid PL/I parameter name.

Mapping Fixed and Unbounded Arrays

Depending whether you are using the client or the server side, there are several possibilities regarding arrays. See also the *array-definition* under *Software AG IDL Grammar* in the IDL Editor documentation for the syntax on how to describe fixed arrays and unbounded arrays within the IDL file, and refer to `fixed-bound-array-index`.

Client Side

- Fixed arrays within the IDL file are mapped to fixed PL/I arrays. The dimension and upper bounds are kept.

  Example: The IDL definition "`MYARRAY (A5/1:10)`" is mapped to the PL/I array "`MYARRAY (10) CHAR(5)`".

- Unbounded arrays with and without `maximum-upper-bound` are *not* supported by PL/I client-side wrapping.

  Example: The IDL definition "`MYARRAY (A10/1:V10)`" as well as "`MYARRAY (A10/1:V)`" is *not* supported.

Server Side

- Fixed arrays within the IDL file are mapped to fixed PL/I arrays. The dimension and upper bounds are kept.

  Example: The IDL definition "`MYARRAY (A5/1:10)`" is mapped to the PL/I array "`MYARRAY (10) CHAR(5)`".

  Note: An RPC server with a PL/I array defined with asterisks is supported on the server side and is an alternative mapping to an array with fixed upper bounds of the same dimension.

  Example: The IDL definition "`MYARRAY (A5/10,10,10)`" can either be mapped to the PL/I array "`MYARRY (10,10,10) CHAR(5)`" or the PL/I array "`MYARRAY (*,*,*) CHAR(10)`". The RPC server source can be manually modified.

- Unbounded arrays with or without `maximum-upper-bound` are mapped to PL/I arrays defined with asterisks. The dimension is kept.

  Example: The IDL definitions "`MYARRAY (A5/V10)`" and "`MYARRAY (A5/1:V)`" are mapped to the PL/I array "`MYARRAY (*) CHAR(5)`".

Mapping Groups and Periodic Groups

Client and Server Side

Groups within the IDL file are mapped to the PL/I structures or PL/I arrays of 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.

Mapping Structures

Client and Server Side

Structures within the IDL file are dissolved at the location where they are used. They are mapped to PL/I structures using level numbers like groups. See the `structure-parameter-definition` (IDL) 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 `IN OUT` 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 the `direction-attribute`.

Client Side

The direction specification is reflected in the generated client interface objects as follows:

- Direction attributes do not change the PL/I call interface.
- 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.

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.

Server Side

The RPC Server for PL/I considers the direction attribute send from any RPC client Java, DCOM, C, COBOL, NET , XML and PL/I. Parameters with the `IN` attribute are *not* sent back to the RPC client.

Mapping the ALIGNED Attribute

See the `attribute-list` under *Software AG IDL Grammar* in the *IDL Editor* documentation for the syntax of attributes in the IDL file and refer to `aligned-attribute`.

Client Side

A PL/I client can send the parameters with the `ALIGNED` attribute to an RPC server. The RPC server decides (depending on programming language and environment) whether the parameter is aligned or not. The `ALIGNED` attribute is not considered in the generated PL/I client interface itself.

Server Side

The RPC Server for PL/I server considers `ALIGNED` parameters as needed, when the `ALIGNED` attribute is sent by an RPC client Java, DCOM, C, COBOL, .NET, PL/I XML and PL/I.

Calling Servers as Procedures or Functions

The IDL syntax allows definitions of procedures only. It does not have the concept of a function. A function is a procedure which, in addition to the parameters, returns a value. Procedures and functions are transparent between clients and server, i.e. a client using a function can call a server implemented as a procedure and vice versa.

Client Side

It is possible to call the remote procedure as a function and not as a procedure, if you prefer it, if it suits your interface and if the client interface objects are generated with a Batch interface (`TARGET=BATCH_xxx`). An `EXEC CICS LINK` interface (`TARGET=CICS_xxx`) *cannot* be invoked as a function. However, you can call a batch interface within CICS.

Example. The function `float sin(float x)` will be called as a function and not as a procedure, when defined in the IDL file as follows:

```
Library ... is
  Program 'sin' is
    Define Data Parameter
      1 x (F4) In
      1 Function_Result (F4) Out
    End-Define
```

It can be invoked as:

```
y = sin(x);
```

The client template generates a PL/I function instead of a PL/I procedure if the following is true:

- A batch interface is generated, i.e. the value for the template option `TARGET` is set to "`BATCH_xxx`"
- In the interface description (IDL file) the last
  - parameter's name is `function_result`. The name `function_result` is not case-sensitive.
  - parameter's direction is Out. See `attribute-list` under *Software AG IDL Grammar* in the *IDL Editor* documentation.
  - parameter is a scalar variable, i.e. not an array, group or structure.

Server Side

The RPC Server for PL/I is able to call any PL/I procedure and any PL/I function. For a PL/I function, the returned parameter is always the last parameter.

5 Writing an RPC Client Application with the PL/I Wrapper

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

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

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

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

Step 1a: Embed PL/I Wrapper Preprocessor Definitions

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

```
%include  RPCPPD;
```

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

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

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

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

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

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

Step 1d: Declare and Initialize the RPC Communication Area

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

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

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

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

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

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

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

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

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

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

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

Step 4: Required Settings in the RPC Communication Area

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

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

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

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

Step 5: Optional Settings in the RPC Communication Area

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

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

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

Step 6: Issue the RPC Request

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

Using the Call Interface

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

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

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

Using the EXEC CICS LINK Interface

This interface is used in the scenario *CICS*.

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

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

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

Step 7: Examine the Error Code

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

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

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

END;
```

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

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

6 Using Broker Logon and Logoff

Broker logon and logoff functions are provided through Generic RPC Services module. See *Using the Generic RPC Services Module*.

Log on to the Broker

With the Call Interface

```
...
ERXCOM.COM_FUNCTION = 'LO'; /* Broker Logon */
ERXCOM.COM_CLIENT_USERID   = 'PLI-USER';
ERXCOM.COM_CLIENT_PASSWORD = 'PLI-PASS';
call xxxSRVI(ERXCOM); /* see (1) below */
IF SUBSTR(ERXCOM.COM_ERROR,8) ^= ERX_S_SUCCESS then
DO;
/* error handling */
/* ... */
END;
/* begin of application logic including calls to interface objects */
...
```

With the EXEC CICS LINK Interface

```
...
ERXCOM.COM_FUNCTION = 'LO'; /* Broker Logon */
ERXCOM.COM_CLIENT_USERID   = 'PLI-USER';
ERXCOM.COM_CLIENT_PASSWORD = 'PLI-PASS';
CICS_LEN = STORAGE(ERXCOM);
CICS_RESP1 = DFHRESP(NORMAL);
CICS_RESP2 = DFHRESP(NORMAL);
/* called CICS program name depends on PP switch ERXFCTPRE */
EXEC CICS LINK PROGRAM  ('xxxSRVI') /* see (1) below */
               RESP     (CICS_RESP1)
               RESP2    (CICS_RESP2)
               COMMAREA (ERXCOM)
               LENGTH   (CICS_LEN);
IF SUBSTR(ERXCOM.COM_ERROR,8) ^= ERX_S_SUCCESS then
DO;
/* error handling */
/* ... */
END;
/* begin of application logic including calls to interface objects */
...
```

(1) The prefix of the program name (`xxxSRVI`) can be customized, see *PL/I Preprocessor Settings*. The default is `PLISRVI`.

Log off from the Broker

With the Call Interface

```
...
/* end of application logic including calls to interface objects */
ERXCOM.COM_FUNCTION  = 'LF'; /* Broker Logoff */
call xxxSRVI(ERXCOM); /* see (1) below */
IF SUBSTR(ERXCOM.COM_ERROR,8) ^= ERX_S_SUCCESS then
DO;
/* error handling */
/* ... */
END;
...
```

(1) The prefix of the program name (`xxxSRVI`) can be customized, see *PL/I Preprocessor Settings*. The default is `PLISRVI`.

With the EXEC CICS LINK Interface

See *Log on to the Broker* above.

Additional Hints

- The `COM_CLIENT_USERID` field (and the `COM_CLIENT_TOKEN` field, when provided) must not change from logon, during call of interface objects, until final logoff.
- If explicit logon is used, as demonstrated here, the `COM_CLIENT_PASSWORD` field may only be provided for the broker logon function call.
- The logon call is the first call to the broker, before any application logic including interface object calls. The logoff call should be issued as soon as RPC communication is no longer needed.
- It is also possible to work with implicit logon, see *Writing Applications with the PL/I Wrapper*.
- Whenever possible we recommend using explicit logon as demonstrated here.

7 Using the RPC Communication Area

This chapter explains how clients use the RPC communication area. The RPC communication area defines a context for RPC clients

The purpose of the RPC communication area includes the following:

- to assign the COM_BROKER_ID and server name, see COM_SERVER_CLASS, COM_SERVER_NAME and COM_SERVER_SERVICE
- to assign the COM_CLIENT_USERID and COM_CLIENT_TOKEN
- for use with ***Conversational RPC*** to hold, for example, the conversation ID, see COM_SERVER_CONVID
- for use with EntireX Security to hold the COM_CLIENT_PASSWORD, COM_CLIENT_SECTOKEN and others
- to keep the results of the last RPC request, for example the error code

The layout of the ***RPC Communication Area*** is described in the reference section.

The PL/I Wrapper allows the RPC Communication Area to be provided as an additional parameter for the generated RPC stubs.

8 Conversational RPC

RPC conversations are supported when communicating with an RPC server.

It is assumed that you are familiar with the concepts of conversational RPC and non-conversational RPC. Open and closing conversations are provided through the *Generic RPC Services Module*.

Using Conversational RPC

To use conversational RPC

1 Open a conversation with the function Open Conversation `OC` (see `COM_FUNCTION` under *RPC Communication Area*) from Generic RPC Services module:

With the Call Interface:

```
...
ERXCOM.COM_FUNCTION = 'OC'; /* Open Conversation */
ERXCOM. COM_SERVER_LIBRARY = 'MYLIB';
call xxxSRVI(ERXCOM); /* see (1) below */
IF SUBSTR(ERXCOM.COM_ERROR,8) ^= ERX_S_SUCCESS then
DO;
/* error handling */
 /* ... */
END;
/* begin of application logic including calls to interface objects */
...
```

With the `EXEC CICS LINK` Interface:

```
ERXCOM.COM_FUNCTION = 'OC'; /* Open Conversation */
ERXCOM. COM_SERVER_LIBRARY = 'MYLIB';
CICS_LEN = STORAGE(ERXCOM);
CICS_RESP1 = DFHRESP(NORMAL);
CICS_RESP2 = DFHRESP(NORMAL);
/* called CICS program name depends on PP switch ERXFCTPRE */
EXEC CICS LINK PROGRAM  ('xxxSRVI') /* see (1) below */
               RESP     (CICS_RESP1)
               RESP2    (CICS_RESP2)
               COMMAREA (ERXCOM)
               LENGTH   (CICS_LEN);
IF SUBSTR(ERXCOM.COM_ERROR,8) ^= ERX_S_SUCCESS then
DO;
/* error handling */
 /* ... */
END;
/* begin of application logic including calls to interface objects */
...
```

(1) The prefix of the program name (xxxSRVI) can be customized, see ***PL/I Preprocessor Settings***. The default is PLISRVI.

- The Open Conversation requires a library to be set in the RPC communication area field COM_SERVER_LIBRARY. See ***The RPC Communication Area (Reference)***.
- After a successful Open Conversation, the broker's conversation ID is stored within the RPC communication area field COM_SERVER_CONVID. See ***The RPC Communication Area (Reference)***. The conversation ID
  - is used during calls to interface objects and also needed for closing the conversation.
  - is cleared if the end of conversation is forced by the broker or the RPC server. This happens if an error with message class 0003 occurs. See *Message Class 0003 - EntireX ACI - Conversation Ended* under *Error Messages and Codes*.
  - is not cleared and remains for any other error returned to be able to continue the conversation.

2 Issue your RPC requests as is done within non-conversational mode, using the generated interface objects.

- Different interface objects can participate in the same RPC conversation.
- RPC conversations and simple non-conversational RPC requests can *not* be handled in parallel using the same RPC communication area without saving and restoring some fields.
- If you need to handle RPC conversations in parallel, or simple non-conversational RPC requests within an ongoing RPC conversation, use multiple RPC communication areas or save and restore the following fields:
  - COM_BROKER_ID (if another broker)
  - COM_SERVER_CLASS (if another class)
  - COM_SERVER_NAME (if another name)
  - COM_SERVER (if another service)
  - COM_SERVER_LIBRARY (if another library)
  - COM_SERVER_CONVID
  - and possibly others, for example user ID, token and password if needed

Terminating a Conversational RPC Communication

Terminate an RPC conversation unsuccessfully with the function Close Conversation `CB` (see `COM_FUNCTION` under ***RPC Communication Area***) from Generic RPC Services module:

With the Call Interface:

```
...
ERXCOM.COM_FUNCTION = 'CB'; /* Close Conversation */
call xxxSRVI(ERXCOM); /* see (1) below */
IF SUBSTR(ERXCOM.COM_ERROR,8) ^= ERX_S_SUCCESS then
DO;
/* error handling */
 /* ... */
END;
/* begin of application logic including calls to interface objects */
...
```

(1) The prefix of the program name (`xxxSRVI`) can be customized, see ***PL/I Preprocessor Settings***. The default is `PLISRVI`.

With the `EXEC CICS LINK` Interface:

See ***Using Conversational RPC*** above.

Closing and Committing a Conversational RPC Communication

Close the RPC conversation successfully with the function Close Conversation and Commit `CE` (see `COM_FUNCTION` under ***RPC Communication Area***) from Generic RPC Services module:

With the Call Interface:

```
...
ERXCOM.COM_FUNCTION = 'CE'; /* Close Conversation and Commit */
call xxxSRVI(ERXCOM); /* see (1) below */
IF SUBSTR(ERXCOM.COM_ERROR,8) ^= ERX_S_SUCCESS then
DO;
/* error handling */
 /* ... */
END;
/* begin of application logic including calls to interface objects */
...
```

(1) The prefix of the program name (`xxxSRVI`) can be customized, see ***PL/I Preprocessor Settings***. The default is `PLISRVI`.

With the `EXEC CICS LINK` Interface:

See *Open Conversation* above.

9 Using Natural Security

Natural Security is only relevant when communicating with Natural RPC Servers.

To communicate with a Natural RPC Server running under Natural Security

1 Set the flag `COM_CLIENT_NATSECURITY` to "`ERX_TRUE`". If set to "`ERX_FALSE`" (or other values), communication with a Natural RPC Server that is secured with Natural Security is *not* possible.

2 The flag must be set prior to issuing any interface object calls. It is not needed for broker communication (see ***Using Broker Logon and Logoff***), but it is also harmful if set.

3 The Natural Security user ID is inherited from the broker's user ID field `COM_CLIENT_USERID` of the RPC Communication Area if no user ID is provided in the field `COM_CLIENT_RPCUSERID`. The Natural Security user ID can always be provided (overwritten) in the field `COM_CLIENT_RPCUSERID` if different from the broker's user ID.

4 The same mechanism (inheritance and override) as described above for the user ID is available for the `COM_CLIENT_PASSWORD` and `COM_CLIENT_RPCPASSWORD` of the RPC communication area.

10 Using Trace

This chapter describes use of the trace function in batch, CICS with call interfaces, IMS and in CICS for the PL/I Wrapper.

Using Trace in Batch, CICS with Call Interfaces, and IMS

To build a trace version for the scenarios Batch and CICS with Call Interfaces

1 Set the preprocessor switch ERXTRACE in file RPCPPS (*PL/I Preprocessor Settings*) to "YES" before you compile the generated interface objects and provided sources (see corresponding step in scenarios *Batch* and *CICS with Call Interfaces*).

2 Compile and link the Trace Functions module (PLISRVT) to your application.

Using Trace in CICS

For the scenario *Using the PL/I Wrapper for CICS* you can trace every interface object and the Generic RPC Services module individually. Interface objects with trace and without trace can co-exist.

To trace generated interface objects

1 Set the preprocessor switch ERXTRACE in file RPCPPS (*PL/I Preprocessor Settings*) to "YES" before you translate and compile the following: (see corresponding step in scenario *CICS*)

- the interface object you want to trace
- the Specific RPC Functions module (PLISRVS) you link into the interface object you want to trace

2 Compile the Trace Functions module (PLISRVT).

3 Link the Trace Functions module (PLISRVT) to the interface object you want to trace.

Compile and link interface objects you do not want to trace with ERXTRACE set to "NO".

To trace the *Using the Generic RPC Services Module*

1 Set the preprocessor switch ERXTRACE in file RPCPPS (*PL/I Preprocessor Settings*) to "YES" before you translate and compile (see corresponding step in scenario *CICS*) the CICS Generic RPC Services module (PLISRVIC) and the Trace Functions module (PLISRVT).

2 Link the Trace Functions module (PLISRVT) to the CICS Generic RPC Services module (PLISRVIC).

11 Using Internationalization with the PL/I Wrapper

It is assumed that you have read the document *Internationalization with EntireX* and are familiar with the various internationalization approaches described there.

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

The PL/I Wrapper programmer is responsible for providing a suitable codepage. If a codepage is provided it must also be a codepage supported by the broker, depending on the internationalization approach, and it must follow the rules described under *Locale String Mapping* in the internationalization documentation.

The codepage is assigned to the RPC communication area in the field `COM_CLIENT_CODEPAGE` as described in ***Step 5: Optional Settings in the RPC Communication Area***.

```
ERXCOM.COM_CLIENT_CODEPAGE = 'ECS937';
```

12 Client and Server Examples for z/OS CICS

This chapter describes the examples provided for the PL/I Wrapper for CICS.

All examples here can be found in the EntireX *examples/RPC* directory under UNIX and Windows.

Basic RPC Client Example - CALC

For CICS on operating system z/OS, the CALCCLT client is built with a PL/I Wrapper interface type "CICS with call interfaces". For this purpose, the PL/I Wrapper target platform is set to "Batch for z/OS" ("Batch" because of the call interface).

Please note there is an additional interface type for CICS, "CICS for z/OS", see ***Target Platform*** under ***Generating PL/I Source Files from Software AG IDL Files***. An example of this is not yet available.

Name	Type	Data Set	Description	Notes
CALC	PL/I source code	EXP960.QCPL	Client interface object for IDL program CALC.	1
CALCCLT	PL/I source code	EXP960.QCPL	A client application calling the remote procedure (RPC service) CALC, with associated example.idl.	2
CALCDFH	CICS CSD	EXP960.QCPL	CSD Definition for RPC client CALCCLT.	
CALCIBM	JCL	EXP960.QCPL	Job (JCL) to build the RPC client CALCCLT.	3
CALCMAP		EXP960.QCPL	CICS Map definition for RPC clients CALCCLT.	
CALC	PL/I copybook	EXP960.QIPL	Client interface object copybook for IDL program CALC.	1
CALCMAP	PL/I copybook	EXP960.QIPL	Description of input and output fields of map CALCMAP.	
EXAMPLE	PL/I copybook	EXP960.QIPL	Client interface object entry points.	1
RPCPPS	PL/I copybook	EXP960.QIPL	PL/I Wrapper Preprocessor Switches and Settings.	

Notes:

1. Client interface objects are delivered with the z/OS installation, but not delivered under UNIX or Windows. Use the EntireX Workbench to generate the client interface objects under UNIX or Windows.
2. Application built according to the client-side build instructions under ***Using the PL/I Wrapper for CICS with Call Interfaces***.
3. The JCL must be adapted according to your needs.

For more information see the readme file in EntireX directory *examples/RPC/basic/example/PLIClient/zosCICS* under UNIX or Windows.

Basic RPC Server Example - CALC

For CICS on operating system z/OS, the CALC server is built with PL/I Wrapper target platform setting "CICS for z/OS". For more information on target platforms, see *Target Platform* under *Generating PL/I Source Files from Software AG IDL Files*.

Name	Type	Data Set	Description	Notes
`CALC`	PL/I source code	EXP960.QVPL	A server application providing the remote procedure `CALC` (RPC service), with associated `example.idl`.	1
`CALCDFH`	CICS CSD	EXP960.QVPL	CSD Definition for remote procedure `CALC` (RPC service).	
`CALCIBM`	JCL	EXP960.QVPL	Job (JCL) to build the remote procedure `CALC` (RPC service).	2

Notes:

1. Application built according to the server-side build instructions under *Using the PL/I Wrapper for CICS*.
2. The JCL must be adapted according to your needs.

For more information see the readme file in EntireX directory *examples/RPC/basic/example/PLIServer/zosCICS* under UNIX or Windows.

13 Client and Server Examples for z/OS Batch

This chapter describes the examples provided for the PL/I Wrapper.

All examples here can be found in the EntireX *examples/RPC* directory under UNIX and Windows.

Basic RPC Client Example - CALC

For batch on operating system z/OS the CALC client is built with PL/I Wrapper target platform setting "Batch for z/OS". See ***Target Platform*** under ***Generating PL/I Source Files from Software AG IDL Files***.

Name	Type	Data Set	Description	Notes
CALC	PL/I source code	EXP960.PCPL	Client interface object for IDL program CALC	1
CALCCLT	PL/I source code	EXP960.PCPL	A client application calling the remote procedure (RPC service) CALC, with associated example.idl.	2
CALCIBM	JCL	EXP960.PCPL	Job (JCL) to build the RPC client CALCCLT.	3
CALCRUN	JCL	EXP960.PCPL	Job (JCL) to execute the RPC client CALCCLT.	3
CALC	PL/I copybook	EXP960.PIPL	Client interface object copybook for IDL program CALC.	1
EXAMPLE	PL/I copybook	EXP960.PIPL	client interface object entry points	1
RPCPPS	PL/I copybook	EXP960.PIPL	PL/I Wrapper Preprocessor Switches and Settings.	

Notes:

1. Client interface objects are delivered with the z/OS installation, but not delivered on UNIX or Windows. Use the EntireX Workbench to generate the client interface objects on UNIX or Windows
2. Application built according to the client-side build instructions under ***Using the PL/I Wrapper for Batch Mode***.
3. The JCL must be adapted according to your needs.

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

Basic RPC Server Example - CALC

For batch on operating system z/OS, the CALC server is built with PL/I Wrapper target platform setting "Batch for z/OS". For more information on target platforms, see ***Target Platform*** under ***Generating PL/I Source Files from Software AG IDL Files***.

Name	Type	Data Set	Description	Notes
`CALC`	PL/I source code	EXP960.PVPL	A server application providing the remote procedure `CALC` (RPC service), with associated example.idl.	1
`CALCIBM`	JCL	EXP960.PVPL	Job (JCL) to build the remote procedure `CALC` (RPC service).	2

Notes:

1. Application built according to the server-side build instructions under ***Using the PL/I Wrapper for Batch Mode***.
2. The JCL must be adapted according to your needs.

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

14 Client and Server Examples for z/OS IMS BMP

There are no special IMS BMP examples delivered.

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 under ***Using the PL/I Wrapper for IMS BMP***.

15 PL/I Wrapper Reference

The RPC Communication Area (Reference)

This section provides the programmer with reference material on the RPC Communication Area. The RPC communication area is used to specify parameters which are needed to communicate with the broker and are not specific to 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.

The RPC communication area is provided in include file `RPCCOM`.

RPC Communication Area Field	Explanation	Req. Opt. Auto	In Out	Notes
`COM_EYECATCHER`	Internal use only - not for customer use.	-	-	1
`COM_VERSION`	Version of RPC Communication Area.	Req.	I	2
`COM_SIZE`	Size of RPC Communication Area.	Req.	I	2
`COM_FUNCTION`	`LO` - Logon to broker.	Opt	I	3
	`LF` - Logoff from broker.	Opt	I	3
	`OC` - Open Conversation.	Opt	I	4
	`CB` - Close Conversation.	Opt	I	4
	`CE` - Close Conversation and Commit.	Opt	I	4
`COM_ERROR`	Error code and error text returned by PL/I Wrapper.	-	O	5
`COM_BROKER_ID`	Broker ID used. Corresponds to the `BROKER-ID` field of the control block.	Req.	I	6
`COM_SERVER_CLASS`	Class Name of the RPC server. Use "`RPC`" for Natural RPC Server. Corresponds to the `SERVER-CLASS` field of the ACI control block.	Req.	I	6
`COM_SERVER_NAME`	Server Name of the RPC server. Corresponds to the `SERVER-NAME` field of the ACI control block.	Req.	I	6
`COM_SERVER_SERVICE`	Service Name of the RPC server. Use "`CALLNAT`" for Natural RPC Server. Corresponds to the `SERVICE` field of the ACI control block.	Req.	I	6
`COM_SERVER_LIBRARY`	Library sent to the RPC server. The library specified here overrides any library information specified in the IDL file, see library-definition.	Opt	I	7, 4
`COM_SERVER_CONVID`	Conversation ID if in an RPC Conversation. Corresponds to the `CONV-ID` field of the ACI control block.	Auto	-	4
`COM_SERVER_WAIT`	Gives the time-out value for the transport system in seconds. Corresponds to the `WAIT` field of the ACI control block.	Opt	I	7

RPC Communication Area Field	Explanation	Req. Opt. Auto	In Out	Notes
COM_CLIENT_USERID	Broker user identification. Corresponds to the USER-ID field of the ACI control block.	Req.	I	6, 3
COM_CLIENT_TOKEN	Token used by the broker to identify the caller. Corresponds to the TOKEN field of the ACI control block.	Opt	I	7, 3
COM_CLIENT_PASSWORD	Password to be transmitted to the broker to check authentication. Corresponds to the PASSWORD field of the ACI control block.	Opt	I	7, 3
COM_CLIENT_SECTOKEN	Broker security token. Received also from broker and assigned to this field for further use. Corresponds to the SECURITY-TOKEN field of the control block.	Auto	I	8
COM_CLIENT_RPCUSERID	RPC user ID, provided to Natural Security.	Opt	I	7, 9
COM_CLIENT_RPCPASSWORD	RPC password provided to Natural Security.	Opt	I	7, 9
COM_CLIENT_CODEPAGE	Corresponds to the LOCALE-STRING field of the ACI control block.	Opt	I	10
COM_CLIENT_BROKERLOGON	Internal use only - *not* for customer use.	-	-	1
COM_CLIENT_NATSECURITY	Flag signaling a Natural Security ticket has to be provided with interface object calls.	Opt	I	7, 9
COM_DATA_FILLED	Internal use only - *not* for customer use.	-	-	1
COM_DATA_MAXLEN	Internal use only - *not* for customer use.	-	-	1
COM_DATA_NCHUNK	Number of chunks allocated as a minimum, used by memory allocation.	Opt	I	7, 10
COM_DATA_SCHUNK	Size of a chunk, used by memory allocation.	Opt	I	7, 10
COM_TRACE_LEVEL	Internal use only - *not* for customer use.	-	-	11
COM_TRACE_FCTLVL	Internal use only - *not* for customer use.	-	-	1
COM_TRACE_INDENT	Internal use only - *not* for customer use.	-	-	1
COM_DATA	Internal use only - *not* for customer use.	-	-	1

RPC Communication Area field

Name of the field 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 (optional). Fields marked with Auto are managed internally by the interface objects and the ***Using the Generic RPC Services Module*** 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. Used internally by PL/I Wrapper. The field *must not* be modified by your application program - otherwise unexpected behavior may occur.
2. For more information, see ***Step 1d: Declare and Initialize the RPC Communication Area***.
3. For more information, see ***Using Broker Logon and Logoff***.
4. RPC conversations are supported if communicating with an RPC server. They are not supported if communicating with XI Adapters. For more information, see ***Conversational RPC***.
5. For more information, see ***Step 7: Examine the Error Code*** under ***Writing an RPC Client Application with the PL/I Wrapper***.
6. For more information, see ***Step 4: Required Settings in the RPC Communication Area*** under ***Writing an RPC Client Application with the PL/I Wrapper***.
7. For more information, ***Step 6: Issue the RPC Request*** under ***Writing an RPC Client Application with the PL/I Wrapper***.
8. If EntireX Security is used, the field *must not* be modified by your application program - otherwise unexpected behavior may occur.
9. Natural Security is only relevant when communication with Natural RPC Server. For more information, see ***Using Natural Security***.
10. For more information, see ***Using Internationalization with the PL/I Wrapper***.
11. Send and Receive buffers for the broker are allocated in blocks, whereby the required number of blocks is determined by the interface object automatically. The default size of a block (4096 byte) can be altered with the field `COM_DATA_SCHUNK` whereby the needed number is adjusted automatically by the interface object then. With the field `COM_DATA_NCHUNK` a minimum number of blocks allocated is defined, which is used if the calculation by the interface object gives a lower number than the minimum. The default minimum number (4 blocks) can be altered with the field `COM_DATA_NCHUNK`. Normally it is not required to alter `COM_DATA_SCHUNK` and `COM_DATA_NCHUNK` fields.
12. For future use.

PL/I Wrapper Modules Delivered as Sources on z/OS

Among the generated PL/I sources and include files, the following PL/I modules are delivered . Some of them are PL/I sources, some of them are PL/I include files.

Module	Data Set	Description	Notes
PLIDEF	EXP960.INCL	Broker ACI control block for PL/I, referenced by the Generic RPC Services Modules	1,5
RPCAPI	EXP960.INCL	Generic RPC services interfaces	4
RPCCOM	EXP960.INCL	RPC communication area	3
RPCDEF	EXP960.INCL	Used internally by stubs and other parts	1
RPCPPD	EXP960.INCL	Preprocessor definitions	1
RPCSRVI	EXP960.INCL	Generic RPC services used internally	1,5
RPCSRVS	EXP960.INCL	Specific RPC functions used internally	1,6
RPCSRVSB	EXP960.INCL	Specific RPC functions used internally	1
PLISRVIB	EXP960.SRCE	Batch generic RPC services	1,2,5
PLISRVIC	EXP960.SRCE	CICS generic RPC services	1,2,5
PLISRVS	EXP960.SRCE	Specific RPC functions	1,2,6
PLISRVT	EXP960.SRCE	Trace functions module	1,2,7

Module

Name of the delivered module.

Data Set

In the table *vrs* represents the version, release and service pack. You will also find the module on DVD in the folder PL/I.

EXP960.QIPL - CICS RPC example include data set for PL/I.

The CICS RPC example include data set for PL/I may be delivered as a patch with a different name, EXP960.QI*nn*, where *nn* is the patch level number. Make sure you install the highest patch level available.

EXP960.PIPL - Batch RPC example include data set for PL/I.

The Batch RPC example include data set for PL/I may be delivered as a patch with a different name, EXP960.PI*nn*, where *nn* is the patch level number. Make sure you install the highest patch level available.

EXP960.INCL - Generic RPC include data set.

The Generic RPC include data set may be delivered as a patch with a different name, EXP960.IN*nn*, where *nn* is the patch level number. Make sure you install the highest patch level available.

EXP960.SRCE - Generic RPC source data set.

The Generic RPC source data set may be delivered as a patch with a different name, EXP960.S0*nn*, where *nn* is the patch level number. Make sure you install the highest patch level available.

Description

Purpose of the module.

Notes:

1. This file is not for direct customer usage. Do *not* modify it.
2. The prefix of the linkage name can be customized, see ***PL/I Preprocessor Settings***.
3. For more information, see ***The RPC Communication Area (Reference)***.
4. For more information, see ***Step 1c: Declare API Constants to PL/I Wrapper***.
5. For a short description, see ***Using the Generic RPC Services Module***.
6. The specific RPC functions module contains the logic to build the RPC request stream and interpret the reply from the RPC server. It does *not* contain the call to the broker stub.
7. The trace module contains functions and procedures to build a trace version of the PL/I Wrapper, see ***Using Trace***.