Using the C Wrapper

- Using the C Wrapper for the Client Side
- Using the C Wrapper for the Server Side (z/OS, UNIX, Windows, BS2000/OSD, IBM i)
- Generate C Source Files from Software AG IDL Files

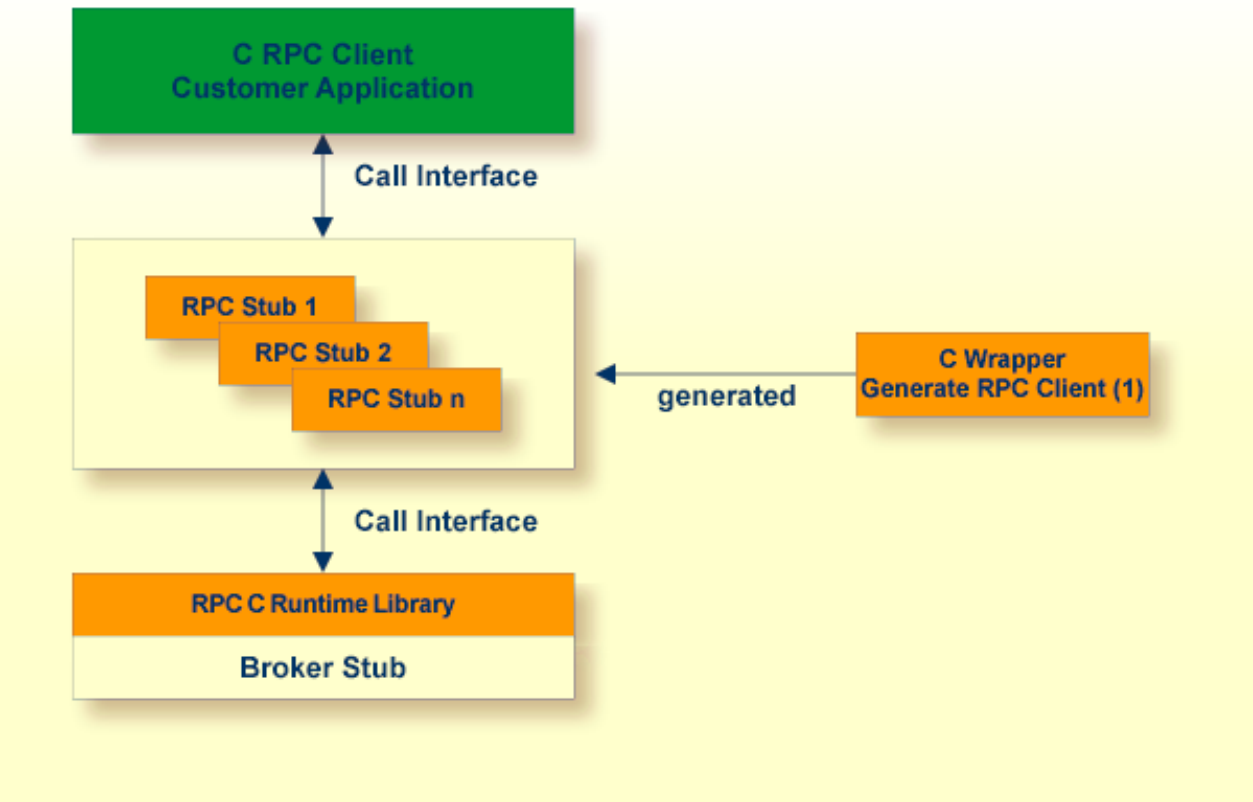
Using the C Wrapper for the Client Side

The C Wrapper provides access to RPC-based components from C applications and enables users to develop both clients and server. This section introduces the various possibilities for RPC-based client applications written in C.

- Using the C Wrapper in Single-threaded Environments (UNIX, Windows)
- Using the C Wrapper in Multithreaded Environments (UNIX, Windows)

Using the C Wrapper in Single-threaded Environments (UNIX, Windows)

This mode applies to UNIX and Windows.



(1) For generation, see *Generate C Source Files from Software AG IDL Files*.

In this scenario, the C RPC client customer application, every generated interface object and the RPC C runtime library (erx) are linked (bound) together to an executable application.

To use the C Wrapper in single-threaded environments

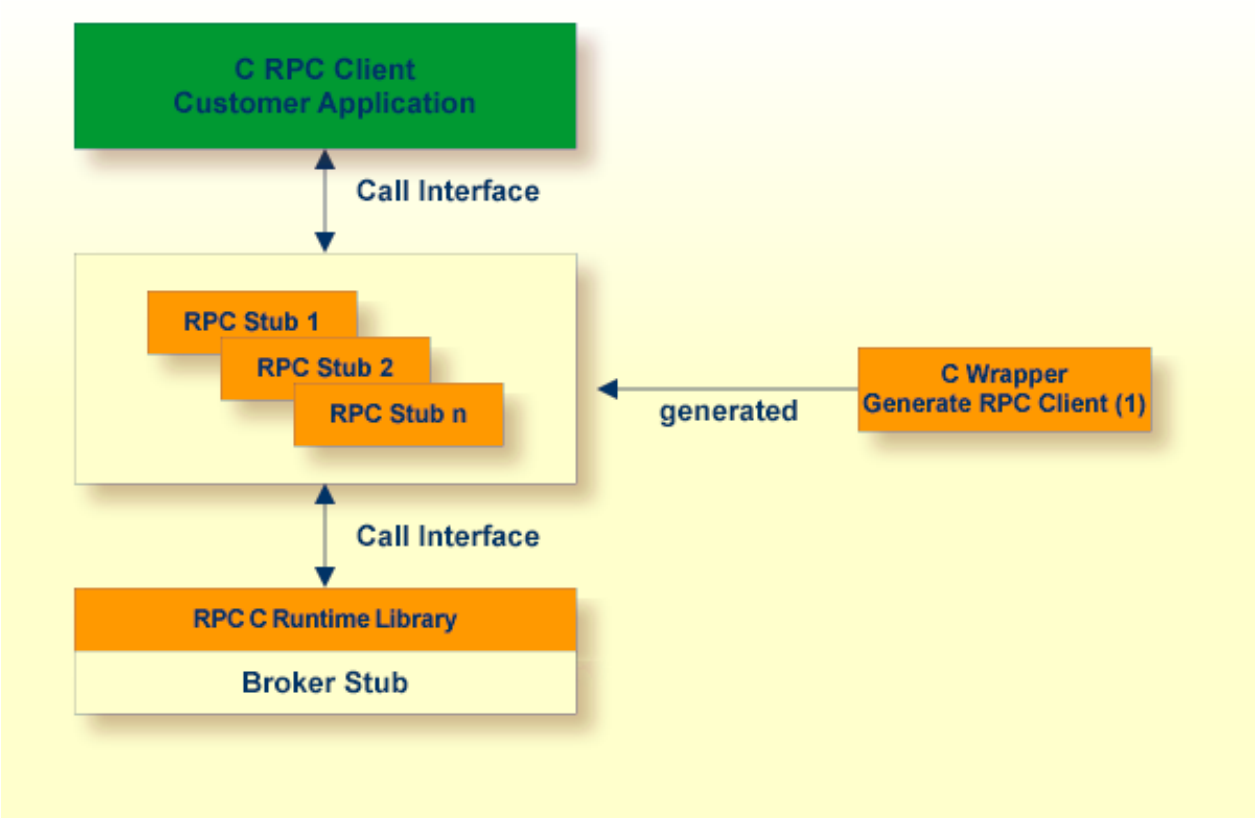
1. Generate the RPC client, see *Generate C Source Files from Software AG IDL Files*
   - and select the *Mapping Options* according to your needs.
   - Do *not* switch on the check box **Multithreaded Client**, see *Generate RPC Client*.
2. If necessary, use FTP to transfer your application and the interface object(s) to the target platform where you write your application.
3. Write your application. See *Writing a Single-threaded C RPC Client Application*.
4. If necessary, transfer your application and the client interface object(s) to the target platform where you compile your application, using FTP.
5. Using a C compiler supported by the C Wrapper and compile
   - the generated client interface object(s)
   - your C RPC client customer application.

   Use the standard C/C++ compiler of your target platform. Please note platform-specific settings and see also the README.TXT file of the *Delivered Examples for the C Wrapper*.
6. Using the linker (binder), link (bind)
   - the compiled client interface object(s)
   - your C RPC client customer application
   - the broker stub
   - under **Windows**: the RPC C runtime library delivered as a library and DLL named *erx.lib* and *erx.dll*
   - under **UNIX**: the RPC C runtime library delivered as a shared object or shared library named *liberx.so* or *liberx.sl*

   Use the standard C/C++ linker of your target platform. Please note platform-specific settings and see also the README.TXT file of the *Delivered Examples for the C Wrapper*.

Using the C Wrapper in Multithreaded Environments (UNIX, Windows)

This mode applies to UNIX and Windows.



(1) For generation, see *Generate C Source Files from Software AG IDL Files*.

In this scenario, the C RPC client customer application, every generated client interface object and the RPC C runtime library (erx) are linked (bound) together to an executable application.

To use the C Wrapper in multithreaded environments

1. Generate the RPC Client, see *Generate C Source Files from Software AG IDL Files*
   - and select the *Mapping Options* according to your needs
   - and switch on the check box **Multithreaded Client**, see *Generate RPC Client*.
2. If necessary, transfer your application and the client interface object(s) to the target platform where you write your application, using FTP.
3. Write your multithreaded C RPC Client application, see *Programming Multithreaded RPC Clients*.
4. If necessary, transfer your application and the client interface object(s) to the target platform where you compile your application, using FTP.
5. Using a C compiler supported by the C Wrapper, compile:
   - the generated client interface object(s)
   - your C RPC client customer application

Use the standard C/C++ compiler of your target platform. Please note platform-specific settings and see also the README.TXT file of the *Delivered Examples for the C Wrapper*.

6. Using the linker (binder), link (bind)

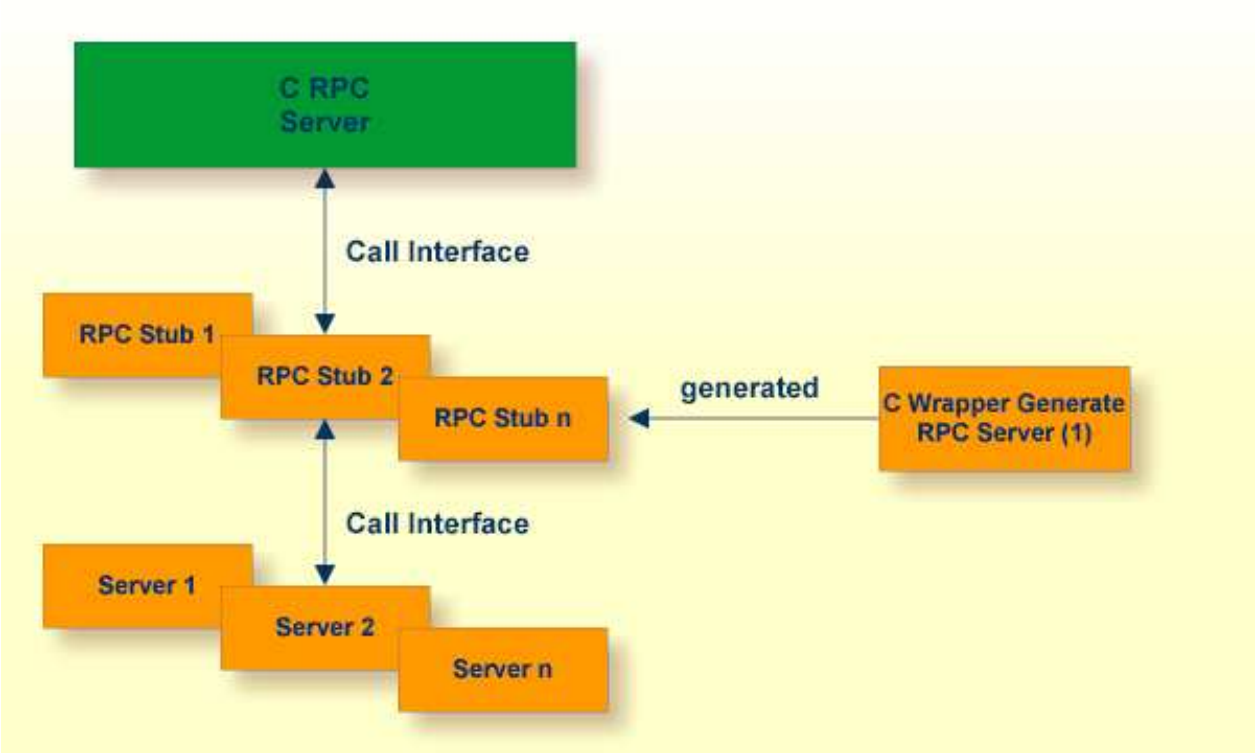
   - the compiled client interface object(s)
   - your C RPC client customer application
   - the broker stub
   - under **Windows**: the RPC C runtime library delivered as a library and DLL named *erx.lib* and *erx.dll*
   - under **UNIX**: the RPC C runtime library delivered as a shared object or shared library named *liberx.so* or *liberx.sl*.

   Use the standard C/C++ linker of your target platform. Please note platform-specific settings and see also the README.TXT file of the *Delivered Examples for the C Wrapper*.

Using the C Wrapper for the Server Side (z/OS, UNIX, Windows, BS2000/OSD, IBM i)

The C Wrapper provides access to RPC-based components from C applications and enables users to develop both clients and server. This section introduces the various possibilities for RPC-based server applications written in C.

This section applies to the operating systems z/OS, UNIX, Windows, BS2000/OSD and IBM i.



(1) For generation, see *Generate C Source Files from Software AG IDL Files*.

For C, the RPC server works with server interface objects. Your server is called dynamically using standard call interfaces.

To use the C Wrapper

1. Generate the RPC Server, see *Generate C Source Files from Software AG IDL Files*, and
   - select the *Mapping Options* according to your needs.
   - The interface objects and the server (skeleton) must be generated with the same mapping options, otherwise results will be unpredictable.

     Note:
     For z/OS, the limitation of 8 characters per (physical) member name must be considered when defining the IDL library (see `library-definition` in the Software AG IDL file. The client interface object will be generated with a prefix letter "D". Therefore an IDL library name "EXAMPLE" within the IDL file results in a physical member name "DEXAMPLE". We suggest using an IDL library name of up to 7 characters in length; if the name is longer, you will

not be able to transfer (using FTP) the generated objects to the mainframe.

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

3. Use the generated server (skeleton) and complete it by applying your application logic. To prevent loss of implementation code when re-generating, we suggest the following before you add any implementation code to the server (skeleton):

   - rename the server from *F<library>.c* to *<library>.c* (or to any other suitable name)
   - or move the server *F<library>.c* to a different directory (folder)

4. If necessary, transfer your server and the server interface object(s) to the target platform where you compile your application, using FTP. The objects to be transferred depend on the platform:

   - **z/OS**
     - your server and the interface objects to a PDS, CA Librarian etc.
   - **BS2000/OSD**
     - your server and the interface objects to your application library. Note that the header files delivered in the LMS library LMS.EXP811.CSRV are required.
   - **IBM i**
     - your server and the interface object source files to the source file QCSRC in your application library
     - header files to the source file H of your application library.
   - **Other Platforms**
     - your server and the interface objects to a suitable directory (folder).

5. With a C compiler supported by the C Wrapper, compile the following objects, depending on the platform:

   - **z/OS**
     - the generated interface object
     - your server (including your application logic)
   - **BS2000/OSD**
     - the generated interface object
     - your server (including your application logic).

   Use any C/C+ ILCS-enabled compiler on BS2000/OSD.

- **IBM i**
  - the generated interface object
  - your server (including your application logic).

  Use the standard ILE C compiler invoked by the following commands for compiling: `CRTCMOD MODULE(X) SRCFILE(..) SRCMBR(..)`.

  EPM-style C programs are not supported.

  You can find various examples of these commands in the procedures provided with the *Delivered Examples for the C Wrapper*.

- **Other Platforms**
  - the generated interface object
  - your server (including your application logic).

  Use the standard C/C++ compiler of your target platform. Please note platform-specific settings and see also the README.TXT file of the *Delivered Examples for the C Wrapper*.

  You can find various examples of these commands in the procedures provided with the *Delivered Examples for the C Wrapper*.

6. Using the linker (binder), link (bind) the following objects, depending on the platform:

- **z/OS**
  - Create dynamic-link libraries (DLLs) for the client interface objects and RPC server. See *Building and Using Dynamic-link Libraries* (DLLs) in the *z/OS C/C++ Programming Guide*, Order No. SC09-2362-03 or later, available through IBM and *Architecture and Software Support in IBM S/390 Parallel Enterprise Servers for IEEE Floating-Point Arithmetic - References* under *http://www.research.ibm.com/journal/rd/435/abbotref.html* (subscription required).
  - There are various possibilities to combine the client interface objects and RPC server together and create dynamic-link libraries (DLLs). We suggest you keep the generated client interface object DLLs separate from RPC server DLLs:
    - Create two larger DLLs, one containing all your client interface objects and one containing all your RPC servers, and use the `FIX(ddlname)` configuration of the parameter `library` of the *Batch RPC Server*.
    - Create separate DLLs, one for each client interface object and each RPC server and use the `PREFIX(prefix)` configuration with prefix "`D`", that is, `PREFIX(D)-PREFIX()` of the parameter `library` of the *Batch RPC Server*.
- **UNIX**
  - the interface object as a shared object or shared library. If, for example, the library name within the Software AG IDL file is HUGO, the standard name of the dynamically callable interface object is *DHUGO.so* or *DHUGO.sl*. The standard name can be changed (see the

Parameter Libraries of the RPC Server).

- your server as a shared object or shared library. If, for example, the library name within the Software IDL file is HUGO, the standard name of the dynamically callable server is *HUGO.so* or *HUGO.sl*.

Use the standard C/C++ linker of your target platform. Please note platform-specific settings and see also the README.TXT file of the *Delivered Examples for the C Wrapper*.

- **Windows**
  - the interface object as a DLL. If, for example, the library name within the Software IDL file is HUGO, the name standard name of the dynamically callable interface object is *DHUGO.dll*. The standard name can be changed (see the Parameter Libraries of the RPC Server).
  - your server as a DLL. If, for example, the library name within the Software IDL file is HUGO, the standard name of the dynamically callable server is *HUGO.dll*.

  Use the standard C/C++ linker of your target platform. Please note platform-specific settings and see also the README.TXT file of the *Delivered Examples for the C Wrapper*.
- **BS2000/OSD**
  - the interface object as an LLM, using `BINDER`
  - your server as an LLM, using `BINDER`

  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.
- **IBM i**
  - the interface object, the RPC server (`EXPRUNTIME`) and the broker stub (EXA) to a service program (type *SRVPGM)
  - your server, the RPC server (`EXPRUNTIME`) and the broker stub (EXA) to a service program (type `*SRVPGM`)

  Use the standard binder invoked by the following commands for binding: `CRTSRVPGM SRVPGM(X) MODULE(X Y Z)`

  The activation group must be `ACTGRP(*CALLER)`. This guarantees the server application runs in the same activation group as the RPC server.

  You can find various examples of these commands in the procedures provided with the *Delivered Examples for the C Wrapper*.
- **Other Platforms**
  - Use the standard linker of your target platform.

7. Provide the interface object library and the server library accessible to the RPC server according to the rules of your operating system.

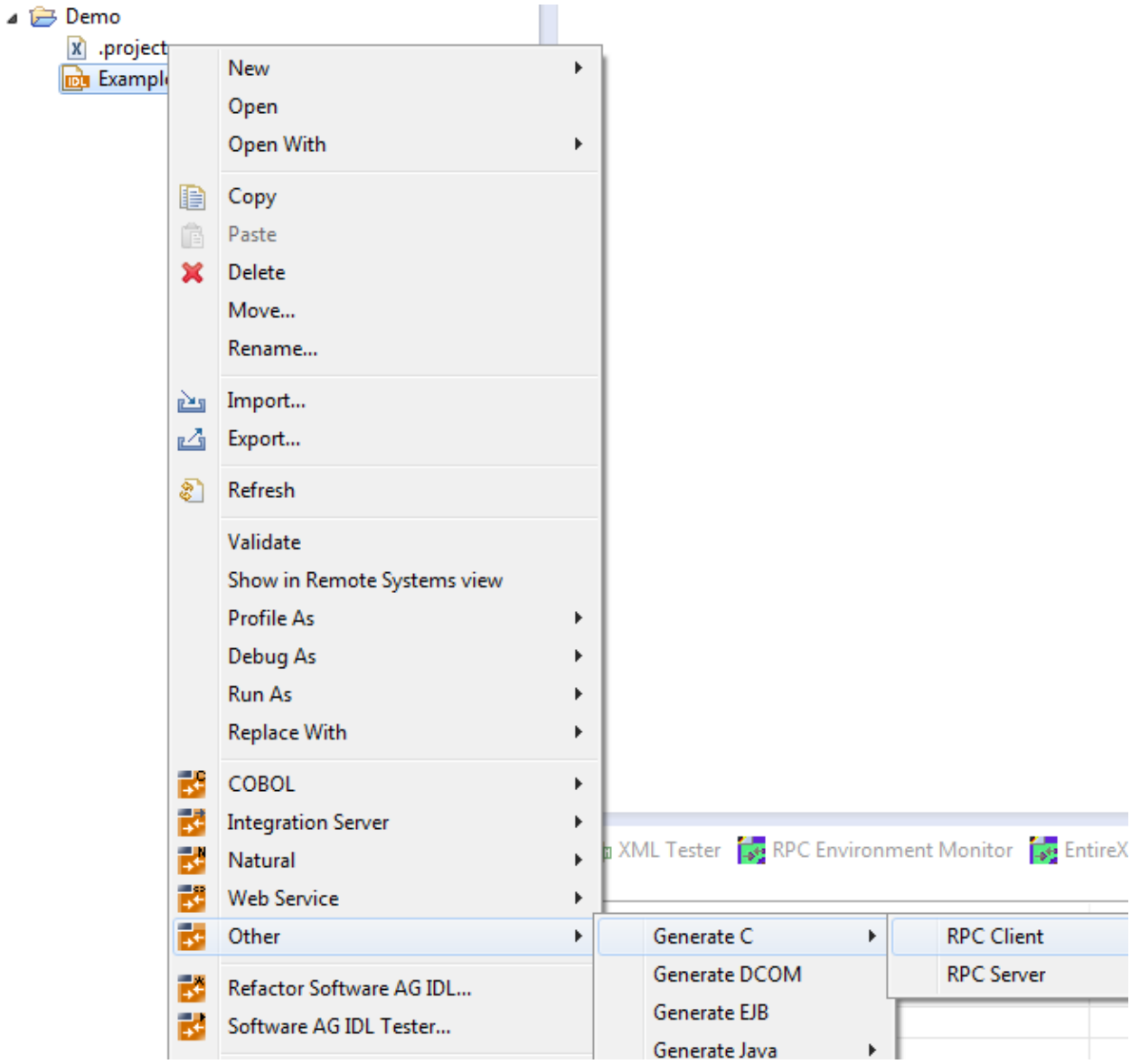
Generate C Source Files from Software AG IDL Files

This section describes how to generate C source files from Software AG IDL files. It covers the following topics:

- Select an IDL File and Generate RPC Client or RPC Server
- Settings
- Mapping Options
- Generate RPC Client
- Generate RPC Server

Select an IDL File and Generate RPC Client or RPC Server

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



For the **RPC client**

- this creates for each library defined in the IDL file the client interface object and its associated header file. All files will be stored in parallel to the IDL file.
- In command-line mode, use the command `-c:client`. See *Using the C Wrapper in Command-line Mode*.

For the **RPC server**

- this creates for each library defined in the IDL file, the server interface object, its associated header file and the server skeleton file for your server implementation. All files will be stored in parallel to the IDL file.
- In command-line mode, use the command `-c:server`. See *Using the C Wrapper in Command-line Mode*.

Warning:
Take care not to overwrite an existing server implementation with a server skeleton.
We recommend you move your server implementation to a different folder, or rename the server implementation.

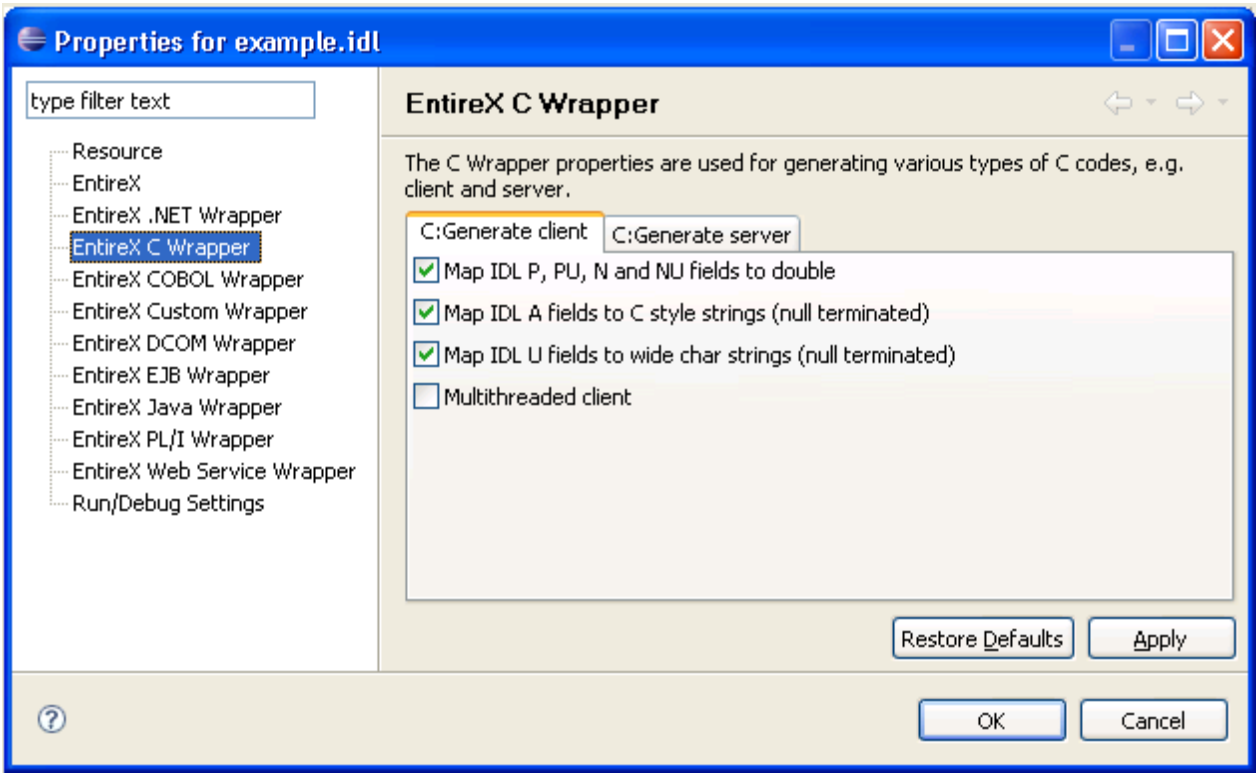
For both **RPC client** and **RPC server**

- 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.
- The header file created is the same for the RPC client as for the RPC server side and contains, for example, C structure definitions for groups in the IDL file and the prototypes for your server. Use these generated C structures in your RPC client application and server implementation as required.

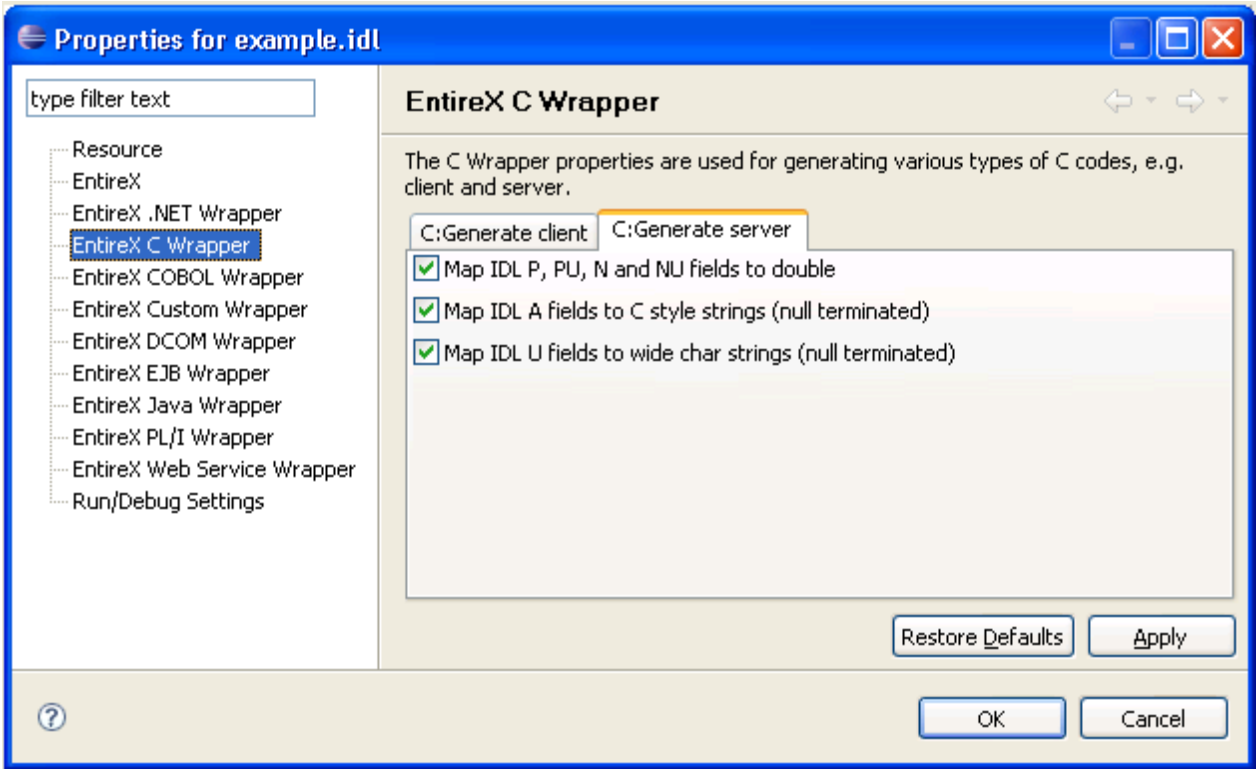
Settings

Use the properties of the IDL file - initialized from the C Wrapper preference page when used for the first time - to manipulate the mapping between Software AG IDL and the C source. A multithreaded client can be enforced.

Client settings



Server settings



Mapping Options

Select the mapping options according to your needs. All mapping options are available for RPC clients and RPC server. See also *Mapping IDL Data Types to C Data Types*.

Option	Description
Map IDL N, P, NU and PU fields to double	If the check box is *checked*, the IDL data types N, NU, P and PU are mapped to the C data type `double`. If the check box is *not checked*, the IDL data type: • N and NU are mapped to the C data type `unsigned char[..]` with unpacked (mainframe Natural, COBOL, PL/I style) contents. • P and PU are mapped to the C data type `unsigned char[..]` with packed (mainframe Natural, COBOL, PL/I style) contents.
Map IDL A fields to C style strings (null terminated)	If the check box is *marked*, the IDL data type A is mapped to a C style string (C data type char[.. + 1] with null termination). This is recommended and comfortable for C programmers and is intended to be used with the C `str...` functions. This mapping does not allow use of trailing blanks and null values to send/receive. If the check box is *not marked*, the IDL data type A is mapped to the C data type unsigned char[..] without null termination (mainframe Natural, COBOL, PL/I style). This allows the use of trailing blanks, null values and is intended to be used with the C `mem...` functions.
Map IDL U fields to wide char strings (null terminated)	If the check box is *marked*, the IDL data type U is mapped to a C style wide character string (C data type `wchar_t[.. + 1]` with null termination). This is recommended and comfortable for C programmers and is intended to be used with the C `wcs...` functions. This mapping does not allow use of trailing wide character blanks (Unicode character x3000) and null values (x0000) to send/receive. If the check box is *not marked*, the IDL data type U is mapped to the C data type `unsigned wchar_t[..]` without null termination (mainframe Natural, COBOL, PL/I style). This allows the use of trailing blanks, null values and is intended to be used with the C `mem...` functions.

If the settings for the client side need to be different from the settings for the server side, generate the RPC client in a directory other than the RPC server directory.

Generate RPC Client

Select the **Multithreaded Client** option according to your needs.

Option	Description
Multithreaded Client	If the check box is *not marked*, the generated client interface object(s) can be used in single-threaded client environments. Use this option if you want to build an RPC client application as described under *Using the C Wrapper in Single-threaded Environments (UNIX, Windows)*. If the check box is *marked*, the generated client interface object(s) are thread-safe and can be used in multithreaded client environments. Use this option if you want to build an RPC client application as described under *Using the C Wrapper in Multithreaded Environments (UNIX, Windows)*.

Generate RPC Server

If you want to build an RPC server application, follow the instructions given under *Using the C Wrapper for the Server Side (z/OS, UNIX, Windows, BS2000/OSD, IBM i)*.