webMethods EntireX

EntireX C RPC Server

Version 9.12

October 2016

WEBMETHODS

This document applies to webMethods EntireX Version 9.12 and all subsequent releases.

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

Document ID: EXX-CRPC-912-20181116

Table of Contents

EntireX C RPC Server v
1 About this Documentation 1
Document Conventions 2
Online Information and Support 2
Data Protection 3
2 Introduction to the C RPC Server 5
Worker Models 6
Server Interface Objects and C Server 7
3 Administering the C RPC Server 9
Customizing the RPC Server 10
Configuring the RPC Server 10
Locating and Calling the Target Server 17
Using SSL/TLS with the RPC Server 19
Starting the RPC Server 21
Stopping the RPC Server 22
Running the RPC Server as a Windows Service 23
Activating Tracing for the RPC Server 23
4 Scenarios 25
Writing a New C Server 26

EntireX C RPC Server

The EntireX C RPC Server allows standard RPC clients to communicate with servers written in C. It works together with the C Wrapper and calls standard libraries (Windows DLLs or UNIX shared objects/libraries).

The supported C development and runtime environments are described in the prerequisites for UNIX and Windows in the Release Notes.

1 About this Documentation

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

Document Conventions

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

Online Information and Support

Software AG Documentation Website

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

Software AG Empower Product Support Website

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

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

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

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

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

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

Software AG TECHcommunity

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

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

Data Protection

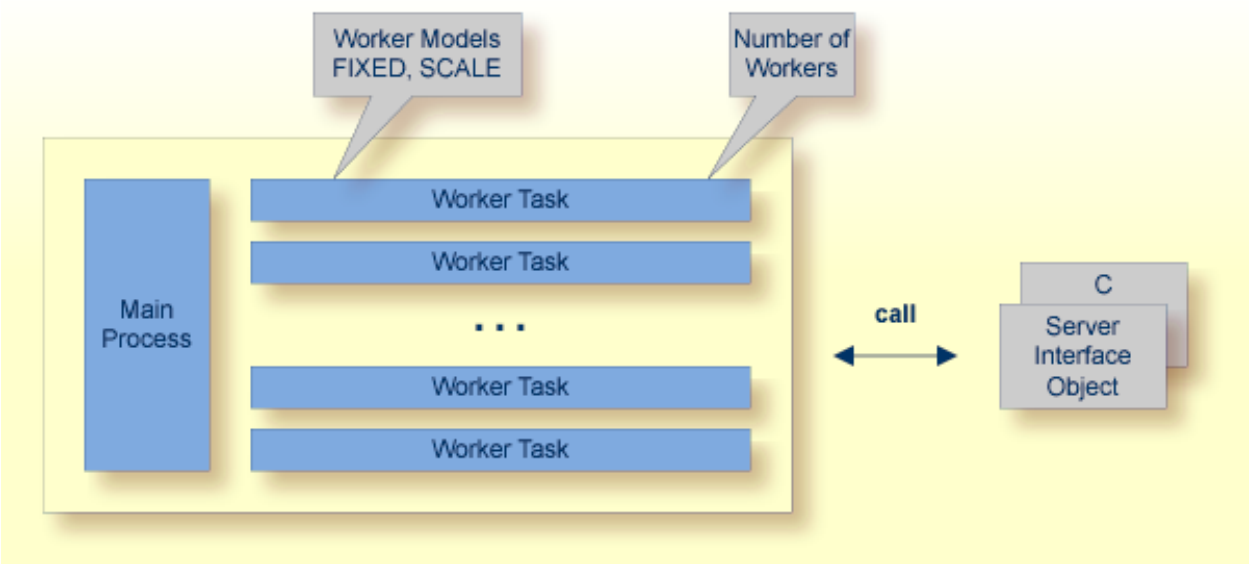
Software AG products provide functionality with respect to processing of personal data according to the EU General Data Protection Regulation (GDPR). Where applicable, appropriate steps are documented in the respective administration documentation.

2 Introduction to the C RPC Server

- Worker Models 6
- Server Interface Objects and C Server 7

The EntireX C RPC Server allows standard RPC clients to communicate with servers written in C. It works together with the C Wrapper and calls standard libraries (Windows DLLs or UNIX shared objects/libraries). This chapter covers the following topics:

Worker Models



RPC requests are worked off inside the RPC server in worker tasks, which are controlled by a main task. Every RPC request occupies during its processing a worker task. If you are using RPC conversations, each RPC conversation requires its own task during the lifetime of the conversation. The C RPC Server provides two worker models:

- `FIXED`
  The *fixed* model creates a fixed number of worker tasks. The number of worker tasks (defined with `minworkers`) does not increase or decrease during the lifetime of an RPC server instance. It is configured by setting the `endworkers` to value `"NEVER"`. Example:

  ```
  endworkers=NEVER
  minworkers=4
  ```

- `SCALE`
  The *scale* model creates worker tasks depending on the incoming load of RPC requests.

  A maximum number (`maxworkers`) of the worker tasks created can be set to restrict the system load. The minimum number (`minworkers`), allows you to define a certain number of tasks - not used by the currently executing RPC request - to wait for new RPC client requests to process. In this way the RPC server is ready to handle many RPC client requests arriving at the same time. It is configured by setting the `endworkers` to value `"TIMEOUT"` or `"IMMEDIATE"`.

- With value `IMMEDIATE`, worker tasks shrink fast, that is, a worker task not used is stopped immediately as soon as it has finished its conversation, except for the number of workers specified as minimum being active.
- With value `TIMEOUT`, worker tasks shrink slowly, that is, all worker tasks not used are stopped in the time specified by the `timeout`, except for the number of workers specified as minimum being active.

Example:

```
endworkers=TIMEOUT
minworkers=2
maxworkers=6
timeout=60
```

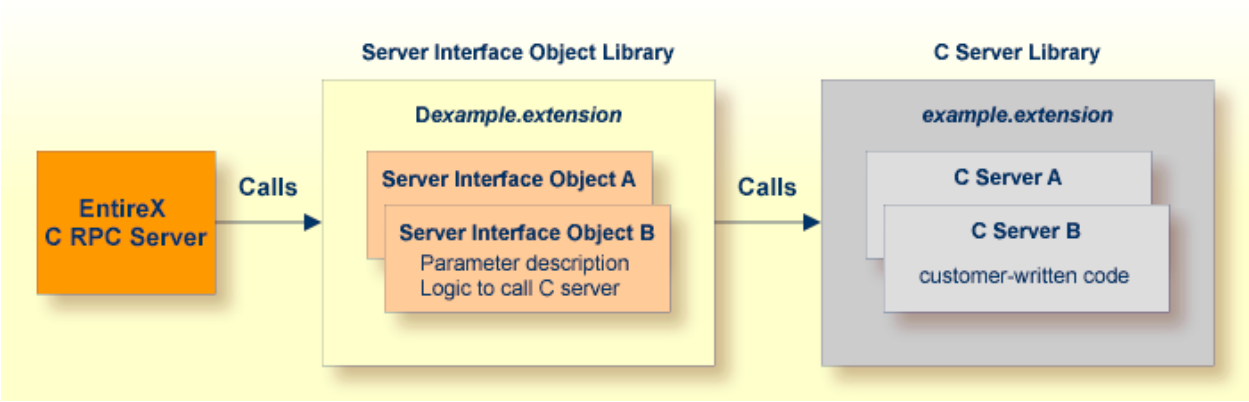
Server Interface Objects and C Server

The C RPC Server uses server interface objects to call the customer-written C server. The server interface objects contain, for example, version information, parameter descriptions and logic to call the C server. Both the server interface objects and C server skeletons are generated with the C Wrapper. All server interface objects are linked together to the library named *Dlibrary-name*[(1)]*.extension*[(2)]. Similarly all C Server are linked together to the library with name *library-name*[(1)]*.extension*[(2)].

See also *Locating and Calling the Target Server*.

Notes:

1. *library-name* is formally the `library-name` of the `library-definition` of the Software AG IDL.
2. The extension is .so or .sl under UNIX; .dll under Windows. Example for IDL library name *EXAMPLE*:
   - For Windows, the server interface object file name is *DEXAMPLE.dll* and the C Server file name is *EXAMPLE.dll*.
   - For UNIX, the server interface object file name is *DEXAMPLE.so* or *DEXAMPLE.sl* and the C Server file name is *EXAMPLE.so* or *EXAMPLE.sl*.


Server Interface Object Library
C Server Library
Dexample.extension
example.extension
EntireX
C RPC Server
Calls
Server Interface Object A
Server Interface Object B
Parameter description
Logic to call C server
Calls
C Server A
C Server B
customer-written code

3 Administering the C RPC Server

- Customizing the RPC Server 10
- Configuring the RPC Server 10
- Locating and Calling the Target Server 17
- Using SSL/TLS with the RPC Server 19
- Starting the RPC Server 21
- Stopping the RPC Server 22
- Running the RPC Server as a Windows Service 23
- Activating Tracing for the RPC Server 23

Customizing the RPC Server

The following elements are used for setting up the C RPC Server:

- Configuration File
- Start Script

Configuration File

The name of the delivered example configuration file is *cserver.cfg* provided in the *config* folder. The configuration file contains the configuration for the C RPC Server. The following settings are important:

- connection information such as broker ID, server address (class, name, service)
- scalability parameters
- trace settings
- etc.

For more information see ***Configuring the RPC Server***.

Start Script

The start script for the C RPC Server is called *cserver.bsh* (UNIX) or *cserver.bat* (Windows) and is provided in the *bin* folder of the installation directory. You may customize this file. The start script contains the following:

- paths to the called C server
- the configuration file used; see ***Configuration File***
- etc.

Configuring the RPC Server

The following rules apply:

- In the configuration file:
  - Comments must be on a separate line.
  - Comment lines can begin with '*', '/' and ';'.
  - Empty lines are ignored.
  - Headings in square brackets [<topic>] are ignored.

- Keywords are not case-sensitive.

- Underscored letters in a parameter indicate the minimum number of letters that can be used for an abbreviated command.

  For example, in brokerid=localhost, brok is the minimum number of letters that can be used as an abbreviation, that is, the commands/parameters broker=localhost and brok=localhost are equivalents.

Parameter	Default	Values	Req/ Opt
brokerid	localhost	Broker ID used by the server. See *Using the Broker ID in Applications*. Example: brokerid=myhost.com:1971	R
class	RPC	Server class part of the server address used by the server. The server address must be defined as a service in the broker attribute file (see *Service-specific Attributes*). Case-sensitive, up to 32 characters. Corresponds to CLASS. Example: class=MyRPC	R
codepage		Depending on the internationalization approach, the codepage (locale string) where incoming data is provided to the called C server. Conversely, the called C server must provide outgoing data in the given codepage, otherwise unpredictable results occur. See *What is the Best Internationalization Approach to use?* under *Internationalization with EntireX* for information on which internationalization approach requires a codepage (locale string). For the most popular internationalization approach, *ICU Conversion*, the correct codepage (locale string) must be provided. This means it must: ■ follow the rules described under *Locale String Mapping* ■ be a codepage supported by the broker ■ be the codepage used in your environment for file and terminal IO, otherwise unpredictable results may occur. Default behavior depends on the operating system: ■ Under UNIX no codepage is transferred to the broker. This means you must configure it, considering the rules above.	R/O

Parameter	Default	Values	Req/ Opt
		■ Under Windows, the codepage defined with the Windows locale settings is transferred to the broker. This is suitable for most situations. If you specify a codepage here, the rules above must be considered. Example: `codepage=iso-8859-1`	

compresslevel	N	Enforce compression when data is transferred between broker and server. See *Data Compression in EntireX Broker*.
		
		`compresslevel= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8| 9 | Y | N`
		
		`0-9` 0=no compression
		9=max. compression
		`N` No compression.
		`Y` Compression level 6.
		
		Example:
		`compresslevel=6`
			O

endworkers	timeout	`NEVER` Defines worker model `FIXED` with a fixed number of worker threads. The number of active workers is defined with `minworkers`. `TIMEOUT` Defines slow-shrinking worker model `SCALE`, where the number of worker threads is adjusted to the current number of client requests. With value `TIMEOUT`, all worker threads not used are stopped in the time specified by the `timeout`, except for the minimum number of active workers specified with `minworkers`. The upper limit of workers parallel active is restricted with `maxworkers`. `IMMEDIATE` Defines fast-shrinking worker model `SCALE`, where the number of worker threads is adjusted to the current number of client requests. With value `IMMEDIATE`, worker threads not used are stopped immediately as soon as they have finished their conversation, except for the minimum number of active workers defined with `minworkers`. The upper limit of workers active in parallel is restricted with `maxworkers`.	O

Parameter	Default	Values	Req/ Opt

		Example:
		`endworkers=timeout`
		`minworkers=2`
		`maxworkers=6`
		`timeout=60`	

library	`library =PREFIX(D) - PREFIX()`	`library = search_logic [- library]`
		
		where *search_logic* is one of `FIX(dllname)` | `PREFIX(prefix)` | `PREFIX()`, and
		library can be repeated up to four times, that is, five entries are possible.
		
		`FIX(dllname)` The IDL library name coming from the RPC client is ignored. You have to define the library names (Windows DLLs or UNIX shared objects/libraries).
		`PREFIX(prefix)` The IDL library name coming from the RPC client is used to form the library name (Windows DLLs or UNIX shared objects/libraries). As prefix you can define any character. If an RPC client sends, for example, "SYSTEM" as the IDL library name and "D" is defined as prefix, the library name derived is "DSYSTEM".
		`PREFIX()` The IDL library name coming from the RPC client is used as library name (Windows DLLs or UNIX shared objects/libraries).
		
		Example `FIX` configuration:
		`library=FIX(MYSTUBS)-FIX(MYRPCS)`
			O

logon	`YES`	Execute broker functions `LOGON/LOGOFF` in worker threads. Must match the setting of the broker attribute `AUTOLOGON`. Reliable RPC requires logon set to YES. See *Reliable RPC*. `NO` No logon/logoff functions are executed. `YES` Logon/logoff functions are executed. Example: `logon=no`	O

Parameter	Default	Values	Req/ Opt
minworkers	1	Minimum limit of tasks active in parallel. ■ For worker model SCALE: minimum number of workers active in parallel. Do not set a value higher than maxworkers. ■ For worker model FIXED: number of workers active in parallel. Do not set a value higher than 256. See also endworkers. Example: minworkers=2	O
maxworkers	10	Upper limit of tasks active in parallel for worker model SCALE. Do not set a value higher than 256. See also endworkers. Example: maxworkers=2	O
password	no default	Password for broker logon. Case-sensitive, up to 32 characters. For more information see broker ACI control block field PASSWORD. Example: password=MyPwd	O
restartcycles	15	Number of restart attempts if the broker is not available. This can be used to keep the C RPC Server running while the broker is down for a short time. A restart cycle will be repeated every 60 seconds. **Note:** Internally, the server waits in periods of 10 seconds (performing six times more cycles), which you can see in the server output. When the number of specified cycles is reached and a connection to the broker is not possible, the C RPC Server stops. Example: restartcycles=30 The server waits up to 30 minutes (30*6*10 seconds) before it terminates due to a missing broker connection.	O
servername	SRV1	Server name part of the server address used by the server. The server address must be defined as a service in the broker attribute file. See *Service-specific Attributes*. Case-sensitive, up to 32 characters. Corresponds to SERVER of the broker attribute file.	R

Parameter	Default	Values	Req/ Opt
		Example: `servername=mySrv`	
service	CALLNAT	Service part of the server address used by the server. The server address must be defined as a service in the broker attribute file. See *Service-specific Attributes*. Case-sensitive, up to 32 characters. Corresponds to `SERVICE` attribute of the broker attribute file. Example: `service=MYSERVICE`	R
ssl_file	no default	Set the SSL parameters. See ***Using SSL/TLS with the RPC Server*** for examples and more information.	O
timeout	60	Timeout in seconds, used by the server to wait for broker requests. See broker ACI control block field `WAIT` for more information. Also influences `restartcycles` and worker model `SCALE`. Example: `timeout=300`	O
tracedestination	`ERXTrace.`*`nnn`*`.log`	The name of the destination file for trace output. By default the main trace file name is `ERXTrace.`*`nnn`*`.log`, where *`nnn`* can be in the range from 001 to 005. See also ***Activating Tracing for the RPC Server***. ■ Under UNIX, the trace file is located in the current working directory. ■ Under Windows, the trace file is located in a subfolder of the windows folder *My Documents*. If the default is not used and a `tracedestination` is specified, you can use the following variables depending on the operating system: `%...%` — Windows — Environment variable. `$(...)` — UNIX — Environment variable. `@PID` — UNIX, Windows — Process ID. `@TID` — UNIX, Windows — Task ID. `@RANGE[ n,m ]` — UNIX, Windows — *`m`* must be greater than *`n`*, range is from 0 - 999	O

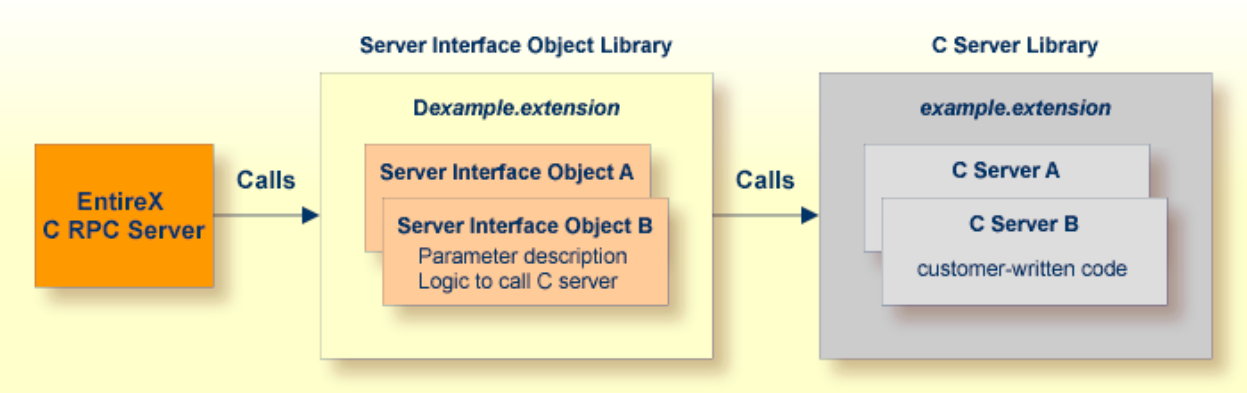
Parameter	Default	Values	Req/ Opt
		`@CSIDL_PERSONAL` Windows The user's home directory. The variable will be resolved by Windows shell functions. `@CSIDL_APPDATA` Windows The *Application Data Directory*. The variable will be resolved by Windows shell functions. `@CSIDL_LOCAL_APPDATA` Windows The *Local Application Data Directory*. The variable will be resolved by Windows shell functions. See also ***Activating Tracing for the RPC Server***. Example: `tracedestination=ERXTrace.log`	

`tracelevel`	`None`	Trace level for the server. See also ***Activating Tracing for the RPC Server***.
		
		`tracelevel = None | Standard | Advanced | ↩ Support`
		
		`None` No trace output.
		`Standard` For minimal trace output.
		`Advanced` For detailed trace output.
		`Support` This trace level is for support diagnostics and should only be switched on when requested by Software AG support.
		
		Example:
		`tracelevel=standard`
			O

Parameter	Default	Values	Req/ Opt
traceoption	None	Additional trace option if trace is active. See also *Activating Tracing for the RPC Server*. None No additional trace options. STUBLOG If tracelevel is Advanced or Support, the trace additionally activates the broker stub log. NOTRUNC Normally if a data buffer larger than 8 KB is traced, the buffer trace is truncated. Set this option to write the full amount of data without truncation. **Note:** This can increase the amount of trace output data dramatically if you transfer large data buffers. Example: traceoption=(STUBLOG,NOTRUNC)	O
userid	ERX-SRV	Used to identify the server to the broker. See broker ACI control block field USER-ID. Case-sensitive, up to 32 characters. Example: userid=MyUid	R

Locating and Calling the Target Server

The IDL library and IDL program names that come from the RPC client are used to locate the server interface objects and your customer-written C servers. See library-definition and program-definition. This two-level concept (library and program) has to be mapped to the C RPC Server environment. The called C servers and server interface objects are implemented as libraries (shared objects/libraries under UNIX; DLLs under Windows). Those libraries also have a two-level concept: file name and entry point name.



See also *Server Interface Objects and C Server* in the **Introduction**.

As an example, assume the RPC client sends the IDL library name “EXAMPLE” and IDL program “CALC”:

- **Server interface object**
  For the file name(2), the character "D" is used as a prefix to the IDL library name sent from the RPC client. The resulting name is "DEXAMPLE". For the entry point name(3), the character "D" is used as a prefix to the IDL program name send from the RPC client. The resulting name is "DCALC".
  - Under UNIX: A shared library/object with file name(1) DEXAMPLE.so|sl and entry point DCALC must be accessible through the standard UNIX shared library load mechanism.
  - Under Windows: A DLL named DEXAMPLE.DLL and entry point DCALC must be accessible through the standard Windows DLL load mechanism.
- **C Server file (written by customer)**
  For the file name(2), the IDL library name sent from the RPC client is used directly. The resulting name is "EXAMPLE". For the entry point (written by customer) name(4), the IDL program name sent from the RPC client is used directly. The resulting name is "CALC".
  - Under UNIX: A shared library/object with file name(1) EXAMPLE.so|sl and entry point CALC must be accessible through the standard UNIX shared library load mechanism.
  - Under Windows: A DLL named EXAMPLE.DLL and entry point CALC must be accessible through the standard Windows DLL load mechanism.

Notes:

1. Under UNIX, file names are case sensitive. The IDL library name must exactly match the name of the shared library/object.
2. The mechanism to form the file names can be modified with the *library* parameter. The description here assumes default settings with `PREFIX(D) - PREFIX()`. These match the standard names produced by the C Wrapper.

3. The prefix for the entry point name of the server interface object is always “D”. It does not depend on the *library* parameter.
4. The entry point name for the called C server must match the IDL program name.

Using SSL/TLS with the RPC Server

RPC servers can use Secure Sockets Layer/Transport Layer Security (SSL/TLS) as the transport medium. The term “SSL” in this section refers to both SSL and TLS. RPC-based servers are always SSL clients. The SSL server can be either the EntireX Broker, Broker SSL Agent, or Direct RPC in webMethods Integration Server (IS inbound). For an introduction see *SSL/TLS and Certificates with EntireX* in the EntireX Security documentation.

To use SSL

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

2 Specify the Broker ID, using one of the following styles:

- *URL Style*, for example:

```
ssl://localhost:2010
```

- *Transport-method Style*, for example:

```
ETB024:1609:SSL
```

If no port number is specified, port 1958 is used as default.

3 Specify SSL parameters, using one of the methods below:

- **As part of the Broker ID**
  The simplest way to specify short SSL parameter is to add them to the Broker ID.

  Example with URL-style Broker ID:

```
ssl://localhost:2010?VERIFY_SERVER=N&TRUST_STORE=c:\\certs\\CaCert.pem
```

Example with transport-method-style Broker ID:

```
ETB024:1609:SSL?VERIFY_SERVER=N&TRUST_STORE=c:\\certs\\CaCert.pem
```

- **In the SSL file**
  Complex SSL parameters can be specified in a so-called SSL file, a text file containing the parameters.

  1. Define the SSL file with the SSL parameters, for example file *mySSLParms.txt* with the following contents:

     ```
     VERIFY_SERVER=N
     TRUST_STORE=c:\\certs\\CaCert.pem
     ```

  2. Define the SSL file in the configuration file of the C RPC Server. See parameter `ssl_file` under ***Configuring the RPC Server***. Example:

     ```
     brokerid=ssl://localhost:2010
     .
     .
     ssl_file=C:\mySSLdirectory\mySSLParms.txt
     ```

If the SSL client checks the validity of the SSL server only, this is known as *one-way SSL*. The mandatory `trust_store` parameter specifies the file name of a keystore that must contain the list of trusted certificate authorities for the certificate of the SSL server. By default a check is made that the certificate of the SSL server is issued for the hostname specified in the Broker ID. The common name of the subject entry in the server's certificate is checked against the hostname. If they do not match, the connection will be refused. You can disable this check with SSL parameter `verify_server=no`.

If the SSL server additionally checks the identity of the SSL client, this is known as *two-way SSL*. In this case the SSL server requests a client certificate (the parameter `verify_client=yes` is defined in the configuration of the SSL server). Two additional SSL parameters must be specified on the SSL client side: `key_store` and `key_passwd`. This keystore must contain the private key of the SSL client. The password that protects the private key is specified with `key_passwd`.

The ampersand (&) character cannot appear in the password.

SSL parameters are separated by ampersand (&). See also *SSL/TLS Parameters for SSL Clients*.

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

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

Starting the RPC Server

Before starting, make sure all your server interface objects and (customer-written) C servers are accessible through the standard Windows DLL or UNIX shared library/object load mechanism. See also *Locating and Calling the Target Server*.

To start the C RPC Server

- Use the *Start Script*.

  Or:

  Use the following format:

```
rpcserver CFG=name [-option] [brokerid] [class] [servername] [service]
```

  Here are some sample options. See *Configuring the RPC Server* for full list.

`-serverlog file`	Defines an alternative log file. Under Windows, this is typically used by Windows Services. See *Running the RPC Server as a Windows Service*.
`-s[ilent]`	Run the RPC server in silent mode, that is, no terminal input will be required (for example to acknowledge error messages). The batch scripts will terminate automatically. Under UNIX, this is recommended when running in background mode.
`-TraceDestination file`	Set the trace destination parameter.
`-TraceLevel None|Standard|Advanced`	Set the trace level parameter.

  Note: The server input arguments are resolved from left to right. Parameters defined in the configuration file may be overridden by parameters applied on the command line and vice versa. See *Configuring the RPC Server* for full list of options.

  Or:

Under Windows you can use the C RPC Server as a Windows Service. See *Running the RPC Server as a Windows Service*.

Stopping the RPC Server

To stop the C RPC Server

- Use the command `stopServer`. See Stop EntireX Broker Server Instances in Command Central's Command-line Interface.

  Or:

  Stop the server instance using Command Central's Graphical User Interface. See *Stopping a Server Instance*.

  Or:

  Use the command `stopService`. See Stop Running EntireX Broker Services in Command Central's Command-line Interface.

  Or:

  Stop the service using Command Central's Graphical User Interface. See *Stopping a Service*.

  Or:

  Use the command-line utility `etbcmd`. See `etbcmd` under *Broker Command-line Utilities* in the platform-specific Administration documentation.

  Or:

  Use `CTRL-C` in the session where you started the RPC server instance.

  Or:

  Under UNIX, enter command `kill -process-id`.

Running the RPC Server as a Windows Service

For general information see *Running an EntireX RPC Server as a Windows Service*.

To run the C RPC Server as a Windows Service

1 Customize the ***Start Script*** according to your system installation.

Note: The script file must pass external parameters to the RPC server and use the option `-silent`:

```
rpcserver CFG=..\config\cserver.cfg  -s %*
```

See also ***Starting the RPC Server***.

2 Test your RPC server to see whether it will start if you run your script file.

3 Use the *EntireX RPC Service Tool* and install the `RPCService` with some meaningful extension, for example `MyServer`. If your ***Start Script*** is *cserver.bat*, the command will be

```
RPCService -install -ext MyServer -script install_path\EntireX\bin\cserver.bat
```

The log file will be called *RPCservice_MyServer.log*.

4 In **Windows Services** menu (**Control Panel** > **Administrative Tools** > **Services**) select the service: `Software AG EntireX RPC Service [MyServer]` and change the property `Startup Type` from "Manual" to "Automatic".

Activating Tracing for the RPC Server

To switch on tracing for the C RPC Server

1 Set the parameters `tracelevel`, `traceoption` and `tracedestination`. See ***Configuring the RPC Server***.

2 Start the C RPC Server. See ***Starting the RPC Server***.

3 To evaluate the return codes, see Component Return Codes in EntireX.

To switch off tracing

- Set the `tracelevel` parameter to `None`.

4 Scenarios

- Writing a New C Server 26

Writing a New C Server

To write a new C server

1 Use the C Wrapper to generate a C server skeleton and a C server interface object. Write your C server and proceed as described under *Using the C Wrapper for the Server Side (z/OS, UNIX, Windows, BS2000, IBM i)*.

2 Build an EntireX RPC client using any EntireX wrapper. For a quick test you can:

- use the IDL Tester; see *EntireX IDL Tester* in the EntireX Workbench documentation
- generate an XML mapping file (XMM) and use the XML Tester for verification; see *EntireX XML Tester* in the XML/SOAP Wrapper documentation