Administering the EntireX RPC Server

The Windows RPC Server, together with the C Wrapper, enables you to call DLLs as servers.

This chapter covers the following topics:

- Locating and Calling the Target Server
- Configuring the RPC Server
- Scalability of the RPC Server
- Using Internationalization with the RPC Server
- Using SSL or TLS with the RPC Server
- Starting the RPC Server
- Stopping the RPC Server
- Activating Tracing for the RPC Server

See also *Administering the EntireX RPC Servers using System Management Hub*.

Locating and Calling the Target Server

The library and program names that come from the client are used to locate the target server. This two-level concept (library and program) has to be mapped in some way to the RPC Server environment. The target servers and their stubs are implemented as Windows DLLs. Windows DLLs also have a two-level concept. The library and program names that come from the client are mapped as follows:

- The library name is used to form the file names of the target server DLL and stub DLL.
- The program name is used to form the entry point names for the target server DLL and stub DLL.

The stub DLL as well as the target server DLL must be accessible through the standard Windows DLL load mechanism.

To locate the target server, the *Possible Values for Libraries* is also used as a kind of search sequence. The default for the library parameter is set `PREFIX(D) - PREFIX()` to be compatible with server stubs and target servers written according to *C Wrapper*.

Under normal circumstances it is not necessary to change the library parameter. There may, nevertheless, be occasion to do so:

- Changing the platform default of the library parameter gives you control and independence over the library name that comes from the client.
- By changing it to a setting of `FIX(DMYLIB) - FIX(MYLIB)` and renaming the server stub and target server built according to *EntireX C Wrapper* to *DMYLIB* and *MYLIB*, you can tailor all or part of the target servers to these libraries regardless of what the client sends.

- Changing the platform default can also make sense when Natural is the client environment, since it always sends *SYSTEM* as the library name.

Example

Assume the following situations:

- A client sends *Example* as the library name and *CALC* as the program name.
- A stub DLL with DExample.dll built with the delivered makefile *Server.mak* or a corresponding one exists and can be accessed through the standard Windows DLL load mechanism.
- A target server DLL with the name *Example.dll* built with the delivered makefile *Server.mak* or a corresponding one exists and can be accessed through the standard Windows DLL load mechanism.
- The default value for Windows of `PREFIX(D) - PREFIX()` for the library parameter is not changed.

Search for Stub DLL

The RPC Server under Windows searches for a stub DLL with:

1. An entry point derived from the program name that comes from the client by adding a prefix D. For our example the entry point is DCALC. This prefix has nothing to do with any library parameter configuration and is always D.
2. Names formed by the instructions of the library parameter from left to right. The first library parameter `PREFIX(D)` means: take the library name that comes from the client and add the prefix. For our example above, the DLL name is *DExample.dll*.

If in step 1 such a DLL can be located through the normal DLL load mechanism, it is taken as the stub; otherwise the next DLL name is formed using the next library parameter entry (step 2). If all library parameter entries have been worked off and the stub is not located, an error is returned to the client.

For our example above, the stub *DExample.dll* is found with the first library parameter entry.

Search for Target Server DLL

The RPC Server under Windows searches for the target server DLL with:

1. An entry point using the program name that comes from the client request directly. For our example above, the entry point is CALC.
2. Names formed by the instructions of the library parameter from left to right. The first library parameter `PREFIX(D)` means: take the library that comes from the client and add the prefix. For our example above, the DLL name is *DExample.dll*.

If in step 1 such a DLL can be located through the normal Windows DLL load mechanism, it is taken as the target server; otherwise the next DLL name is formed using the next library parameter entry (step 2). If all library parameter entries have been worked off and the target server is not located, an error is returned to the client.

For our example above, the target server *Example.dll* is found with the second library parameter entry.

Configuring the RPC Server

This section covers the following topics:

- Configuration File Syntax
- Table of Server Parameters
- Possible Values for Endworkers
- Possible Values for Libraries

Configuration File Syntax

- 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.

Table of Server Parameters

Configuration File Parameter Syntax (UNIX, Windows, IBM i)	Value	Req. Opt.	Description	Notes
`brokerid=localhost`	string	R	Broker ID used by the server.	Corresponds to the `BROKER-ID` field of the Broker ACI control block.
`class=RPC`	case-sensitive, up to 32 characters	R	Server class used by the server.	Corresponds to the `SERVER-CLASS` field of the Broker ACI control block.
`codepage=`		O	This field exposes the Broker ACI field `LOCALE-STRING` as a parameter to users of the RPC server.	See *Using Internationalization*.
`compresslevel=0`	0-9 or Y | N	O	Enforce compression when data is transferred between broker and server.	See *Data Compression in EntireX Broker*.
`encryptionlevel=0`	0 | 1 | 2	O	Enforce encryption when data is transferred between client and server.	Corresponds to the `ENCRYPTION-LEVEL` field of the Broker ACI control block. See also *Broker Attributes*.

Configuration File Parameter Syntax (UNIX, Windows, IBM i)	Value	Req. Opt.	Description	Notes
etb_apivers= 0	*n*	O	Determines the Broker API to use.	Corresponds to the API-VERSION field of the Broker ACI control block. We recommend either not configuring the API Version or setting it to 0. This allows the EntireX Broker and the EntireX RPC server to autodetect the best API version to use. For compatibility with older Brokers, the API version can be set manually.
logon=YES	YES | NO	O	YES executes the Broker functions LOGON/LOGOFF. NO does not.	Specify NO for compatibility with EntireX Broker prior to Version 4.1.1.
servername=SRV1	case-sensitive, up to 32 characters	R	Server Name used by the server.	Corresponds to the SERVER-NAME field of the Broker ACI control block.
service=CALLNAT	case-sensitive, up to 32 characters	R	Service used by the server.	Corresponds to the SERVICE field of the Broker ACI control block.
smhport=0	any digit within range 0 to 99999	O	If greater than zero, starts the RPC server with a separate SMH communication task and listen port <smhport> to the local TCP/IP system.	
ssl_file=		O	Set the SSL parameters.	See *Using SSL or TLS with the RPC Server*.
timeout=60	*n*	O	Timeout in seconds, used by the server to wait for Broker requests.	Corresponds to the WAIT field in the Broker ACI control block. See also *Scalability of the RPC Server*.
userid=ERX-SRV	case-sensitive, up to 32 characters	R	Used to identify the server to the broker.	Corresponds to the USER-ID field of the Broker ACI control block.
password=	case-sensitive, up to 32 characters	O	Password for Broker logon.	Corresponds to the PASSWORD field of the Broker ACI control block.
endworkers= timeout	See *Possible Values for Endworkers*	O	Defines the behavior of worker tasks on completion of client requests.	See *Scalability of the RPC Server*.
minworkers= 1	*n*	O	Minimum number of parallel worker threads started.	
maxworkers=10	*n*	O	Maximum number of parallel worker threads started.	

Configuration File Parameter Syntax (UNIX, Windows, IBM i)	Value	Req. Opt.	Description	Notes
tracelevel=None	None | Standard | Advanced | Support	O	Select the trace level for this server.	See *Activating Tracing for the RPC Server*.
tracedest=	Default: `tracedest=C:\Documents and Settings\<userid>\My Documents\Software AG\EntireX\ERXTrace`*nnn*`.log`, where *nnn* is from 001 to 005.	O	The name of the destination file for trace output.	
traceoption=	None | STUBLOG | NOTRUNC	O	Additional trace option if trace is active. **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)`	
library=	`library = PREFIX(D) - PREFIX()`	O	Specifies criteria to locate target servers and any stubs.	See *Possible Values for Libraries* and *Locating and Calling the Target Server*.
restartcycles=15	*n*	O	Number of restart cycles the RPC Server will try to connect to the Broker. A restart cycle will be repeated every <timeout> +60 seconds. When the number of cycles is reached and a connection to the Broker is not possible, the RPC Server stops.	This may occur when the RPC Server is started prior to the Broker or when the Broker is shut down before the RPC Server is shut down.

Possible Values for Endworkers

The server is able to adjust the number of worker threads to the current number of client requests. This is configured with the parameter endworkers and several others. See *Scalability of the RPC Server* for information on how the various parameters work together and what combinations can be specified.

Value	Explanation
N	**Never** The number of worker threads is fixed. No additional worker threads are started. Minworkers determines the number of workers started.
T	**Timeout is used** The number of worker threads ranges between the `minworkers` and `maxworkers` settings, depending on the number of currently active client requests. Until `maxworkers` has been reached, the server tries to maintain enough free worker threads to accept all incoming clients. The server stops all worker threads not used in the time specified by the `timeout` server parameter (see `timeout`), except for the number of workers specified in `minworkers`.
I	**Immediately** The number of worker threads ranges between the `minworkers` and `maxworkers` settings, depending on the number of client requests currently active. Until `maxworkers` has been reached, the server tries to maintain enough free worker threads to accept all incoming clients. The server stops a thread immediately as soon as it has finished its conversation. When the number of active workers falls below the number of workers specified in `minworkers`, a new thread will be started.

Possible Values for Libraries

The library parameter defines how the RPC Server locates the target server and any stubs on the platform.

The following coding rules apply to the library parameter:

- Up to five library entries can be specified as a sequence.
- Library entries are separated by a hyphen "-".
- Library entries are used from left to right by the RPC Server.

The meaningful combinations vary per platform and the type of target server:

Operating System	Type of Target Server	Configuration	Description
IBM i	Target servers in ILE COBOL compatible with *Mapping IDL Data Types to COBOL Data Types* in the COBOL Wrapper documentation or Target servers ILE RPG compatible with *Using EntireX RPC for RPG under IBM i* or Target servers ILE CL compatible with *Software AG IDL to CL Mapping*.	`FIX(`*`library`*`)` `F(`*`library`*`)`	The library sent with the client request is ignored. The configured library *`library`* is used to locate the target server.
UNIX Windows IBM i	Target servers and their stubs compatible with EntireX C Wrapper.	`FIX()` or `F()`	The library name sent with the client request is ignored. The program name sent with the client request is used to locate the target server.

		`FIX(`*`library`*`)`
		or
		`F(`*`library`*`)`
			The library sent with the client request is ignored. The configured library *`library`* is used to locate the target server and any stubs on the platform.

		`PREFIX()` or `P()`	The library name sent with the client request is used to locate the target server and any stubs on the platform.

		`PREFIX(`*`prefix`*`)`
		or
		`P(`*`prefix`*`)`
			The library name sent with the client request is prefixed with the value in "`prefix`" before locating the target server and any stubs on the platform.

Example: `library = PREFIX(D) - PREFIX()`

The default for the `library` parameter is set to satisfy the environment specifics best. Under normal circumstances it is not necessary to change the `library` parameter.

For an explanation of the approach to locating the target server on your platform, see *Locating and Calling the Target Server*.

Scalability of the RPC Server

- Parameters

- Configuration Examples
- Suggested Configuration on First Usage

Parameters

The RPC server can be configured to adjust the number of worker threads to the current number of client requests. When more clients are active, more worker threads are needed to achieve the best throughput. Depending on the configuration, worker threads are started on demand and stopped as soon as they are no longer needed.

This mechanism can be configured with the following parameters:

EntireX RPC Server under operating system:	**Configuration**	**endworkers**	**minworkers**	**maxworkers**	**timeout**
UNIX Windows IBM i	Fixed number of workers.	Never.	Determines the number of workers started.	Unused.	Not used with this configuration.
UNIX Windows IBM i	Scaling number of workers between `minworkers` and `maxworkers` without any idle time.	Immediately.	Determines the minimum number of workers started.	The upper limit of workers started.	Not used with this configuration.
UNIX Windows IBM i	Scaling number of workers between `minworkers` and `maxworkers` with configurable idle time.	Timeout.			The idle time for workers can be configured, i.e. a worker is stopped when, for the period defined by timeout, no client request has to be served and the minimum number of workers has not been reached.

Configuration Examples

Configuration 1: Medium Lifespan of Worker Threads

- `endworkers=T` (timeout)

- `timeout=600`
- `minworkers=1`
- `maxworkers=10`

The endworkers parameter determines the condition under which a worker will be stopped. The value is the period of time specified by the parameter `timeout` (600 seconds, i.e. 10 minutes). Active workers will be stopped if no client requests arrive within the timeout period, except for the number of threads specified in `minworkers`.

Minworkers specifies the minimum number of workers that must be available to handle incoming client requests. The server is started (manually) and the first worker (`minworkers=1`) waits for client requests. When the first client request arrives, a second worker is started. This ensures that there will be at least one free worker (`minworkers=1`) to handle the next incoming client request.

When the first client request has been worked off (in conversational mode when the conversation has been ended, and in non-conversational mode when the request has been answered), there will be two workers active. For the next incoming client request (second request) no additional worker will be started because the second worker is still free. A third worker will only be started if a third client request arrives before the second request has been finished, in which case there will be three active workers, and so on.

The `maxworkers` parameter specifies the maximum number of active worker tasks permitted (default is 10).

Configuration 2: Shortest Lifespan of Worker Threads

- `endworkers=I` (immediately)
- `timeout=600`
- `minworkers=1`
- `maxworkers=10`

In this example the `endworkers` parameter has been set to "`I`" (immediately). This setting will stop worker threads immediately when client requests are completed, except for the number of threads specified in minworkers. All other behavior is the same as for *Configuration 1: Medium Lifespan of Worker Threads*.

Configuration 3: Fixed Number of Workers

- `endworkers=N` (never)
- `timeout=600`
- `minworkers=10`
- `maxworkers=`

This configuration determines a fixed number of workers. The `maxworkers` parameter is ignored and the `endworkers` parameter is set to "`N`" (never). All worker threads are started immediately with the server and will never stop. This method is useful in minimizing system resources.

Suggested Configuration on First Usage

When you first start using Micro Focus RPC Server, we suggest the following settings for scaling the server:

- `endworkers=T` (timeout)
- the `timeout` parameter can be set, for example, to 2 minutes (`timeout=120`).
- low value for `minworkers` is suggested (e.g. `minworkers=2`)
- the `maxworkers` setting depends on the expected maximum number of clients active in parallel (e.g. `maxworkers=10`)

Using Internationalization with the RPC Server

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

The RPC Server running under Windows

- does not, by default, send a codepage as part of the locale string to the broker
- assumes that the broker's locale string defaults match; see *Broker's Locale String Defaults*. If they do not match, provide the codepage explicitly.

When setting the codepage manually with the parameter `codepage`, the following rules apply:

- You can provide a codepage in the locale string sent to the broker. If a codepage is provided, it must follow the rules described under *Locale String Mapping*.
- The RPC server itself does not convert your application data (contained in RPC IDL type A, K, AV and KV fields) received from the broker before giving them to your server application. Under normal circumstances, it is not possible to configure a codepage other than the codepage used in your environment for file and terminal IO. If this is not adhered to, unpredictable results may occur.
- The codepage used must also be a codepage supported by the broker, depending on the internationalization approach.
- Before starting the RPC Server, set the locale string with the parameter `codepage`.

  Example:

  ```
  codepage=LOCAL
  ```

Using SSL or TLS with the RPC Server

There are two ways of specifying SSL or TLS, depending on the complexity of the parameters:

- as part of the Broker ID for short parameters, the simplest way

- using the SSL file, a text file containing more complex parameters.

For more information, see *SSL or TLS and Certificates with EntireX*.

Specifying the SSL or TLS Parameters as Part of the Broker ID

The simplest way to specify SSL or TLS parameters is to add them to the Broker ID.

Example:

```
ssl://ETB001?TRUSTSTORE=whatever
```

Specifying the SSL or TLS Parameters in a Separate File

For complex SSL or TLS parameters there is the SSL file, a text file containing the parameters.

The `SSL_FILE` keyword points to this text file.

To specify the SSL or TLS parameters in the SSL file

1. Set the parameters as described under *Running Broker with SSL or TLS Transport* under z/OS | UNIX | Windows.
2. Prefix/suffix the Broker ID with the SSL key.

   Example:

   ```
   brokerid=SSL://ETB001
   .
   .
   ssl_file=C:\mySSLdirectory\mySSLParms.txt
   ```

Starting the RPC Server

Before starting the EntireX RPC server, ensure that all dynamically loaded objects (server stubs and server) can be accessed using the search path.

To start the EntireX RPC server manually

- Use the format

  ```
  RPCserver CFG=<name> [-option] [Brokerid] [Class] [ServerName] [Service]
  ```

  where <*name*> determines the configuration file in use.

  Options:

  - -smhport number
    Sets the RPC server parameter smhport to number. Typically used by SMH Facility.
  - -serverlog <*file*>
    Defines an alternative log file for Window services. Typically used by Windows Services. See *Running an EntireX RPC Server as a Windows Service*.

- -s[ilent]: Run server in silent mode, that is: no terminal input will be required (e.g. acknowledge error messages). The job will terminate automatically. Recommended for background jobs.
- -TraceDestination <*file*>
  Set the trace destination parameter.
- -TraceLevel None.Standard.Advanced. Set the trace level parameter.

Note:
The server input arguments will be resolved from left to right. Thus parameters that can be applied on the command line as well in the configuration file may be overriden.

To start the EntireX RPC server using Windows services

- See *Running an EntireX RPC Server as a Windows Service*.

Note:
For reasons of compatibility with versions before 5.1.1, the old command to start the server

```
RPCserver <Brokerid> <Class> <ServerName> <Service>
```

will continue to be supported. However, a server started with this call will use the default parameters. Parameters other than `Broker ID`, `Class`, `ServerName`, `Service` require the CFG= form of the server start command.

To start the EntireX RPC server using System Mangagement Hub

1. See *Administering the EntireX RPC Servers using System Management Hub* under UNIX | Windows for information on adding an EntireX RPC server to the System Management Hub.
2. The System Management Hub facility "Adding a Local RPC Server" will use the batch script *startcserver.bat* of the EntireX Installation to apply the server parameters. Change the batch script according to your system installation or add parameters to the System Management Hub "Start Command" input property.

Stopping the RPC Server

To stop the EntireX RPC Server

- Use the function Deregister a Service or Deregister a Server of the System Management Hub. This method ensures that the deregistration from the Broker is correct.

See also *EntireX RPC Server Return Codes*.

Activating Tracing for the RPC Server

To switch on tracing for the RPC Server

- Set the parameters `tracelevel` and `tracedestination`, see *Table of Server Parameters*.

To evaluate the return codes, see *Error Messages and Codes*.

See also *Tracing the RPC Server*.