Administering the EntireX RPC Server under IBM i

The EntireX RPC Server under IBM i enables you to call programs as servers, using ILE (Integrated Language Environment).

This chapter covers the following topics:

- General Rules and Conventions
- 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
- Troubleshooting
- Activating Tracing for the RPC Server

General Rules and Conventions

The following rules apply to all server and client applications written in C under IBM i.

- Only ILE (Integrated Language Environment) program objects are supported.

  The EntireX RPC Server does not support OPM (Original Program Model) and EPM (Extended Program Model) objects.

- The servers run in a multithreaded environment. Therefore your application server programs must be thread-safe. This implies that all commands and subprograms accessed in your servers must allow multithreads.

- When linking servers, the binding parameter `ACTGRP(*CALLER)` must be specified. This guarantees that a server application runs in the same activation group as the calling RPC server. Before processing a server program, the RPC Server verifies if the application was bound with `ACTGRP(*CALLER)`. If not, the RPC Server rejects the call and sends a message to the system operator.

- The IBM i terms "bind" or "binding" are equivalent to the terms "link" or "linking" in the current manual.

- Under IBM i, the term "service program" for a `*SRVPGM` program is equivalent to "shared library" under UNIX.

Warning:
When you compile ILE programs, do *not* use the PDM source compiling option 14. Use only the ILE commands `CRTBND`*`xxx`* or `CRT`*`xxx`*`MOD` with `CRTPGM`, where *`xxx`* denotes the source language such as CBL for COBOL, RPG or CL for command language.

Locating and Calling the Target Server

- Types of Target Server
- Finding a Server Program
- Passing Parameters to the Server

The library and program names that come from the client are used to locate the target server. This two-level IDL concept (library and program) has to be mapped in some way to the RPC Server environment.

Types of Target Server

The RPC Server under IBM i supports target servers as:

- **ILE-C applications**
  The servers and their stubs are implemented as IBM i service programs of type `*SRCPGM`, written according to *EntireX C Wrapper*. ILE-C applications have a two-level concept. The IDL 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 shared library/object and stub shared library/object
  - The program name is used to form the entry point names for the target server shared library/object and stub shared library/object.
- **ILE-RPG, ILE-COBOL and/or ILE-CL applications**
  For these applications, the servers are called directly without any stubs. The IDL library and program names that come from the client for this type of target servers are mapped as follows:
  - The library name is used to form the IBM i library name.
  - The program name is used to form the IBM i program name, which is implemented as a bound program of type *PGM.

Under IBM i, the RPC Server uses various methods to call the two kinds of server implementations, depending on the programming language you have used to develop your application. The next section describes how the RPC Server under IBM i distinguishes between the two implementations.

Finding a Server Program

The RPC Server searches for a server in the following order:

First Preference

The RPC Server first assumes that the server was developed in ILE-C. It therefore tries to run two service programs of type `*SRCPGM` located in the library list of the user who submitted the RPC Server. The service programs represent the shared libraries of:

- the server stub, which is named by default D*My_shared_library* and
- the server implementation, named by default *My_shared_library*, which contains the application functions.

My_shared_library corresponds to the library name in the IDL file. The program names specified in the IDL file are used to form the entry point names, e.g. `DCALC` and `CALC`. They represent C functions in the server program. Based on the IDL layout, the sources of the stub and the server frame must be generated from templates using the *EntireX Workbench* function "Generate C Server" or the corresponding Software AG IDL Compiler command.

Generating C stubs and servers is described in detail under *Using the C Wrapper*. Extensions specific to IBM i are described under *Using the C Wrapper for the Server Side (z/OS, UNIX, Windows, BS2000/OSD, IBM i)*. You can find an example of a C application server under *Step 4: Verify the RPC Server using C*.

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

Second Preference

If no appropriate server written in C can be found in the RPC Server job's library list, the RPC Server assumes that the server was developed in ILE-COBOL, ILE-RPG or ILE-CL. It tries to execute a stubless ILE program of type `*PGM`, where

- the IBM i program name is derived from the IDL program name that comes from the client, and
- the IBM i library name is derived from the IDL library name that comes from the client. Changing the default setting `FIX() file` gives you control and independence over the library name that comes from the client. You can reroute the call to a target library of your choice (currently restricted to 8-character names) by setting the server parameter `Library=FIX(`*`MyLib`*`)` in the configuration file of the RPC Server. In this case, the library name sent with the client request is ignored.

See *Using EntireX RPC for CL under IBM i* for how to use RPC servers in RPG, *Using the COBOL Wrapper* on how to use RPC servers in COBOL and *Using EntireX RPC for CL under IBM i* for how to use RPC servers in CL.

You will find an IBM i example of a COBOL application server under *Step 3: Verify the RPC Server using COBOL*.

If both access approaches fail, an error message is sent back to the client.

Passing Parameters to the Server

The number of Level 01 parameters passed is predetermined by the client and the IDL definition. If this number does not match the number specified in the linkage section of your ILE server program, the operating system will reject the call.

Note:
For stubless access to servers written in ILE-RPG, ILE-COBOL and ILE-CL, currently up to 16 level-01 parameters are supported. More parameters are ignored. Parameter fields of type floating point are not supported.

Configuring the RPC Server

- Configuration File Syntax
- Table of Server Parameters
- runoption
- 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.

Configuration File Parameter Syntax (UNIX, Windows, IBM i)	Value	Req. Opt.	Description	Notes
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*.
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.	

Configuration File Parameter Syntax (UNIX, Windows, IBM i)	Value	Req. Opt.	Description	Notes
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.
Kernelsecurity = level	Y yes | N no | U user	O	Dynamic enablement if EntireX security is active.	Corresponds to the KERNELSECURITY 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=		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:	

Configuration File Parameter Syntax (UNIX, Windows, IBM i)	Value	Req. Opt.	Description	Notes
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.
runoption=		O	Provides EntireX RPC Server with additional information when calling target servers without stubs.Default: runoption= P_SIGNED N_SIGNED DIRECT_CALL	

runoption

The runoptions are important for EntireX RPC Servers when the servers are called directly without server stubs, that is on z/VSE Batch, z/OS IMS, CICS, IBM i platforms or with a callable RPC server. When there is no server stub information, it is not known how to align and unmarshal to the target data type (and marshal the data back). With the `runoption` parameter, it is possible to provide the EntireX RPC Server with such information.

The runoptions are normally set to meet the platform’s requirements. In most cases it should not be necessary to modify them.

Keyword	Description
`I2_ALIGNED` `NOT_I2_ALIGNED`	Aligns Integer (medium) data types. See *Mapping of Software AG IDL Data Type I2* to your target programming language COBOL | RPG.
`I4_ALIGNED` `NOT_I4_ALIGNED`	Aligns Integer (large) data types. See *Mapping of Software AG IDL Data Type I4* to your target programming language COBOL | RPG.
`F4_ALIGNED` `NOT_F4_ALIGNED`	Aligns Floating point (small) data types. See *Mapping of Software AG IDL Data Type F4* to your target programming language COBOL | RPG.
`F8_ALIGNED` `NOT_F8_ALIGNED`	Aligns Floating point (large) data types. See *Mapping of Software AG IDL Data Type F8* to your target programming language COBOL | RPG.
`P_SIGNED` `NOT_P_SIGNED`	Treats Packed decimal data types as signed packed. See *Mapping of Software AG IDL Data Type P number1 [. number2]* to your target programming language COBOL | RPG.
`N_SIGNED` `NOT_N_SIGNED`	Treats Unpacked decimal data types as signed numeric. See *Mapping of Software AG IDL Data Type N number1 [. number2]* to your target programming language COBOL | RPG.
`COBOL_TIME` `NOT_COBOL_TIME`	Maps Time and Date data types to the COBOL format PIC 9(21). See *Mapping of Software AG IDL Data Type T* to your target programming language COBOL | RPG.
`DIRECT_CALL` `NOT_DIRECT_CALL`	Forces calling server directly without server stub.
`C_STRING` `NOT_C_STRING`	Provides string variables (A) in C style: The string is guaranteed to be limited with a terminating byte, thus the size is increased. For example, A10 will be mapped to a maximum of 11 bytes.
`NUMERIC_DOUBLE` `NON_NUMERIC_DOUBLE`	Maps numeric data types (N, P) to double.

The `runoption` keyword can be issued multiple times:

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 EntireX RPC Server, we suggest the following settings for scaling the server.

Endworkers=T (timeout) and a 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). The timeout parameter can be set e.g. to 2 minutes i.e. timeout=120.

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 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. Conversion or translation is done by the broker according to the codepage the RPC Server tells the broker.

The RPC Server running under IBM i

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

With the parameter codepage you can

- provide (override) 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*.
- force a locale string to be sent if communicating with broker version 7.1.x and below.

The codepage the RPC Server tells the broker must be a codepage supported by the broker, depending on the internationalization approach.

Using SSL or TLS with the RPC Server

This function is not currently supported on this platform.

Starting the RPC Server

Important:
The EntireX RPC Server under IBM i can only be started as a batch job for multithreading reasons. You must use the `SBMJOB` parameter `ALWMLTTHD=*YES` to allow multiple threads. Ensure that the subsystem you are submitting to the RPC server allows multiple threads. Use the command `WRKJOBD` to verify the relevant job description.

To start the RPC Server

1. Ensure that the EntireX product library EXX is in your library list.

2. Create a startup procedure that submits the RPC Server to batch. The RPC Server is called XSERVER (of type *PGM) and is usually located in the EntireX product library EXX.

   The following startup procedure shows the sample procedure STR_RPCSRV that is delivered in the library EXAMPLE:

```
            PGM
/*-------------------------------------------------------------------*/
/* Example start procedure for the EntireX RPC Server.               */
/*                                                                   */
/* Make sure that the RPC Server PGM and the configuration file      */
/* would be found in the library list.                               */
/*-------------------------------------------------------------------*/
            DCL        VAR(&NULL)   TYPE(*CHAR) LEN(2)  +
                         VALUE(X'0000')
            DCL        VAR(&PARM1)  TYPE(*CHAR) LEN(40) +
                         VALUE('CFG=*LIBL/QCLSRC(SERVER_CFG)')
            DCL        VAR(&PARM2)  TYPE(*CHAR) LEN(4)  VALUE('-S')
/*-------------------------------------------------------------------*/
/* C-Language binding requires each command-line parameter to be     */
/* terminated with a NULL character                                  */
/*-------------------------------------------------------------------*/
            CHGVAR     VAR(&PARM1) VALUE(&PARM1 *TCAT &NULL)
            CHGVAR     VAR(&PARM2) VALUE(&PARM2 *TCAT &NULL)
/*-------------------------------------------------------------------*/
SBMJOB      CMD(CALL PGM(*LIBL/XSERVER) +
                           PARM(&PARM1 &PARM2)) +
                           JOB(RPCSERVER) ALWMLTTHD(*YES)
            SNDPGMMSG  MSG('EntireX RPC Server job submitted to +
                         batch') MSGTYPE(*COMP)
            ENDPGM
```

   One or two parameters are passed to the RPC server:

   - Parameter 1 specifies the RPC server configuration file (for details see the next step), and
   - The optional parameter 2 "-S", which runs the server in silent mode, that is: no terminal input will be required (e.g. acknowledge error messages). This is recommended for background jobs.

   Note:
   C programming conventions require that each parameter must be terminated with a two-byte hexadecimal zero (variable &NULL in the procedure sample).

3. Create/adjust the RPC server configuration file.

   The IBM i configuration file is derived from the file used for RPC Servers under UNIX. A sample data set is provided in the text member EXAMPLE/QCLSRC(RPCSRV_CFG) delivered with the IBM i installation kit. Edit the individual parameters as described in *Configuring the RPC Server*. (For the keywords for the server parameter runoption see runoption)

   Most important are the BrokerID=Localhost:1971 of the remote Broker with which you want to register your server, and the ServerName=SRV1 that identifies your service.

   Some parameters are not supported under IBM i (see the comment line in the text member referring to this).

4. Run the startup procedure. Make sure that the configuration file is available in your library list. When the RPC server is up and running, a corresponding entry should appear in the Active Jobs list that can be verified with command `WRKACTJOB`. If you use the sample procedure described above, the server will be named `RPCSERVER` as specified in the `SBMJOB` parameter `JOB`.

Stopping the RPC Server

To stop the EntireX RPC server

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

See also *EntireX RPC Server Return Codes*.

Troubleshooting

Follow the hints and recommendations given below if the RPC Server does not return the expected results:

- If the client process receives an EntireX error message, read the relevant explanation under *Error Messages and Codes*.
- Use the command `WRKACTJOB` to determine whether the `RPCSERVER` process is still up and running.
- If the `RPCSERVER` is still running, invoke the job with option 5 (work with ...) and select command 10 to display the job log. Examine the log for system messages and exceptions.
- If the `RPCSERVER` is no longer active, search for a printed job log in the spooler of the user that submitted the RPC server. Usually, the IBM i system uses the default name `QPJOBLOG` for the print file.
- For many exceptions, the RPC Server writes messages to the system operator. Use the command `WRKMSG MSGQ(*SYSOPR)` to query the operator's message queue.
- To monitor the traffic of the RPC Server and to detect problems, restart the RPC Server with tracing activated. To do so, provide a valid log file and enable the tracing with the parameters `TraceLevel` and `TraceDestination` in the RPC Server configuration file member. See *Activating Tracing for the RPC Server* for details.

Activating Tracing for the RPC Server

To switch on tracing for the RPC Server

1. In the RPC Server configuration file, set the parameters TraceLevel and TraceDestination. See *Table of Server Parameters*.

   See also *Starting the RPC Server* for information on how to specify the parameters in the RPC Server configuration file under IBM i.

2. Evaluate the return codes. See *Error Messages and Codes*.

Supported Trace Levels

Trace Level	Description
None	No tracing.
Standard	Traces the interface parameters.
Advanced	Traces the interface parameter, Broker calls and internal information needed for support.

Trace Destination

To assign a valid log file to the RPC Server

1. Use program `EXX/EXACRTLOG` to create file `LOG` in the target library, which is usually your current library.

2. Specify the Trace Destination in the RPC Server configuration file. Trace destination is a generic name including environment variables, e.g., `%TEMP%`; `@PID` (process ID), `@TID` (thread ID), `@RANGE[n,m]`, where `m` must be greater than `n`, range is from 0 - 999. Using the `RANGE` option under IBM i, you can specify the following syntax:

   ```
   TraceDestination=MyLib/LOG(MyMember@RANGE[n-m])
   ```

   where *`MyLib`* is the target library of the file LOG and *`MyMember`* is the prefix of the member name. Every time a new RPC Server session has been started, a new log member will be created it the log file.

   Note:
   Under IBM i, a process ID (@PID) is assigned to a session and *not* changed during session lifetime.

3. When the RPC Server is up and running, use the command `WRKACTJOB` to verify that the server batch job has opened the log file successfully.

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

Example

The following excerpt from an RPC server configuration file demonstrates the usage of the `Tracelevel` and the `Tracedestination`:

```
...
 Tracelevel=Support
 TraceDestination=*CURLIB/LOG(RPC@RANGE[1-100])
 ...
```

When the RPC Server is first started, the member RPC001 will be created in file LOG. The next time, RPC002 will be created and so on.