

webMethods EntireX

EntireX RPC Server for Java

Version 10.5

October 2019

This document applies to webMethods EntireX Version 10.5 and all subsequent releases.

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

Copyright © 1997-2019 Software AG, Darmstadt, Germany and/or Software AG USA, Inc., Reston, VA, USA, and/or its subsidiaries and/or its affiliates and/or their licensors.

The name Software AG and all Software AG product names are either trademarks or registered trademarks of Software AG and/or Software AG USA, Inc. and/or its subsidiaries and/or its affiliates and/or their licensors. Other company and product names mentioned herein may be trademarks of their respective owners.

Detailed information on trademarks and patents owned by Software AG and/or its subsidiaries is located at <http://softwareag.com/licenses>.

Use of this software is subject to adherence to Software AG's licensing conditions and terms. These terms are part of the product documentation, located at <http://softwareag.com/licenses/> and/or in the root installation directory of the licensed product(s).

This software may include portions of third-party products. For third-party copyright notices, license terms, additional rights or restrictions, please refer to "License Texts, Copyright Notices and Disclaimers of Third-Party Products". For certain specific third-party license restrictions, please refer to section E of the Legal Notices available under "License Terms and Conditions for Use of Software AG Products / Copyright and Trademark Notices of Software AG Products". These documents are part of the product documentation, located at <http://softwareag.com/licenses> and/or in the root installation directory of the licensed product(s).

Use, reproduction, transfer, publication or disclosure is prohibited except as specifically provided for in your License Agreement with Software AG.

Document ID: EXX-JAVARPC-105-20220422

Table of Contents

1 About this Documentation	1
Document Conventions	2
Online Information and Support	2
Data Protection	3
2 Introduction to the RPC Server for Java	5
Administration using Command Central	6
Worker Models	7
3 Administering the RPC Server for Java using the Command Central GUI	9
Logging in to Command Central	10
Creating an RPC Server Instance	11
Configuring an RPC Server Instance	15
Viewing the Runtime Status	21
Starting an RPC Server Instance	22
Stopping an RPC Server Instance	24
Inspecting the Log Files	26
Changing the Trace Level Temporarily	27
Deleting an RPC Server Instance	27
4 Administering the RPC Server for Java using the Command Central Command Line	29
Creating an RPC Server Instance	30
Configuring an RPC Server Instance	31
Displaying the EntireX Inventory	47
Viewing the Runtime Status	49
Starting an RPC Server Instance	50
Stopping an RPC Server Instance	50
Inspecting the Log Files	51
Changing the Trace Level Temporarily	53
Deleting an RPC Server Instance	54
5 Administering the RPC Server for Java with Local Scripts	57
Customizing the RPC Server	58
Configuring the RPC Server	59
Using SSL/TLS with the RPC Server	62
Starting the RPC Server	63
Stopping the RPC Server	63
Pinging the RPC Server	64
Running an EntireX RPC Server as a Windows Service	64
Application Identification	65
6 Scenarios and Programmer Information	67
Writing a New Java Server	68
7 Parameter Reference	69
8 Building an EntireX RPC Server for Java Docker Image	71
Prerequisites	72
Building and Running the RPC Server for Java Image	72

Verifying the Build	76
Healthcheck for RPC Server for Java	77

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 <code>folder.subfolder.service</code> , APIs, Java classes, methods, properties.
<i>Italic</i>	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

Product Documentation

You can find the product documentation on our documentation website at <https://documentation.softwareag.com>.

In addition, you can also access the cloud product documentation via <https://www.software-ag.cloud>. Navigate to the desired product and then, depending on your solution, go to “Developer Center”, “User Center” or “Documentation”.

Product Training

You can find helpful product training material on our Learning Portal at <https://knowledge.softwareag.com>.

Tech Community

You can collaborate with Software AG experts on our Tech Community website at <https://tech-community.softwareag.com>. From here you can, for example:

- Browse through our vast knowledge base.
- Ask questions and find answers in our discussion forums.
- Get the latest Software AG news and announcements.
- Explore our communities.
- Go to our public GitHub and Docker repositories at <https://github.com/softwareag> and <https://hub.docker.com/publishers/softwareag> and discover additional Software AG resources.

Product Support

Support for Software AG products is provided to licensed customers via our Empower Portal at <https://empower.softwareag.com>. Many services on this portal require that you have an account. If you do not yet have one, you can request it at <https://empower.softwareag.com/register>. Once you have an account, you can, for example:

- Download products, updates and fixes.
- Search the Knowledge Center for technical information and tips.
- Subscribe to early warnings and critical alerts.
- Open and update support incidents.
- Add product feature requests.

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 RPC Server for Java

- Administration using Command Central 6
- Worker Models 7

The EntireX RPC Server for Java allows standard RPC clients to communicate with servers written in Java. It works together with the Java Wrapper and calls Java server interface objects.

Administration using Command Central

Software AG Command Central is a tool that enables you to manage your Software AG products remotely from one location. Command Central offers a browser-based user interface, but you can also automate tasks by using commands to remotely execute actions from a terminal or custom script (for example CI servers such as Jenkins, or generic configuration management tools such as Puppet or Chef).

The screenshot shows the 'Instances' page in the Software AG Command Central interface. The page has a blue header with navigation links: 'Installations', 'Stacks', 'Licensing', 'Repositories', 'Jobs', and 'Administrator'. Below the header, there's a breadcrumb trail 'Home > Instances > ALL' and a search bar for environments. The main content area is titled 'Instances' and contains a table of installed instances. The table has columns for Name, Component, Status, Alerts, Installation, and Host. The data rows are:

Name [Count]	Component	Status	Alerts	Installation	Host
EntireX Broker ETB001	EntireX Broker ETB001	↑		Local	localhost
CCE [1 Components]	CCE	↑		Local	localhost
IS_default [3 Components]	IS_default	↑		Local	localhost
SPM [2 Components]	SPM	↑		Local	localhost

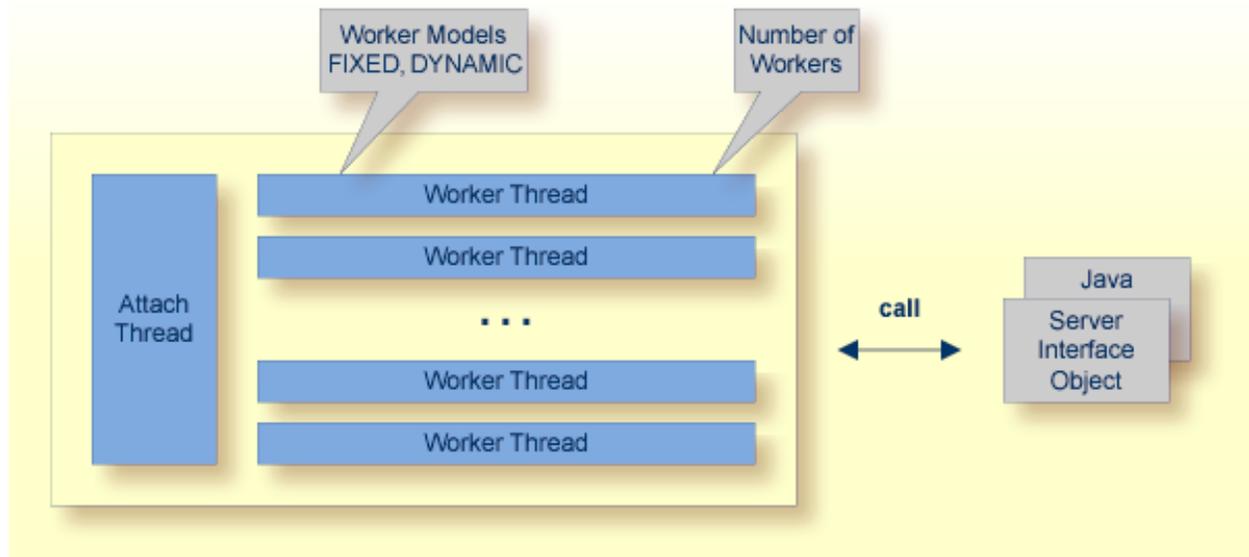
Command Central can assist with the following configuration, management, and monitoring tasks:

- Infrastructure engineers can see at a glance which products and fixes are installed, where they are installed, and compare installations to find discrepancies.
- System administrators can configure environments by using a single web user interface or command-line tool. Maintenance involves minimum effort and risk.
- Release managers can prepare and deploy changes to multiple servers using command-line scripting for simpler, safer lifecycle management.
- Operators can monitor server status and health, as well as start and stop servers from a single location. They can also configure alerts to be sent to them in case of unplanned outages.

The Command Central graphical user interface is described under [Administering the RPC Server for Java using the Command Central GUI](#). For the command-line interface, see [Administering the RPC Server for Java using the Command Central Command Line](#).

The core Command Central documentation is provided separately and is also available under **Guides for Tools Shared by Software AG Products** on the Software AG documentation website.

Worker Models



RPC requests are worked off inside the RPC server in worker threads. Every RPC request occupies during its processing a worker thread. If you are using RPC conversations, each RPC conversation requires its own thread during the lifetime of the conversation. The RPC Server for Java can adjust the number of worker threads to the number of parallel requests. The RPC server provides two worker models:

- **FIXED**
The *fixed* model creates a fixed number of worker threads. The number of worker threads does not increase or decrease during the lifetime of an RPC server instance.
- **DYNAMIC**
The *dynamic* model creates worker threads depending on the incoming load of RPC requests.

For configuration with the Command Central GUI, see [Worker Scalability](#) under *Configuration > Server*.

For technical details, see property `entirex.server.fixedservers` under *Administering the RPC Server for Java with Local Scripts*.

3 Administering the RPC Server for Java using the Command

Central GUI

- Logging in to Command Central 10
- Creating an RPC Server Instance 11
- Configuring an RPC Server Instance 15
- Viewing the Runtime Status 21
- Starting an RPC Server Instance 22
- Stopping an RPC Server Instance 24
- Inspecting the Log Files 26
- Changing the Trace Level Temporarily 27
- Deleting an RPC Server Instance 27

This chapter describes how to administer the EntireX RPC Server for Java, using the Command Central graphical user interface.

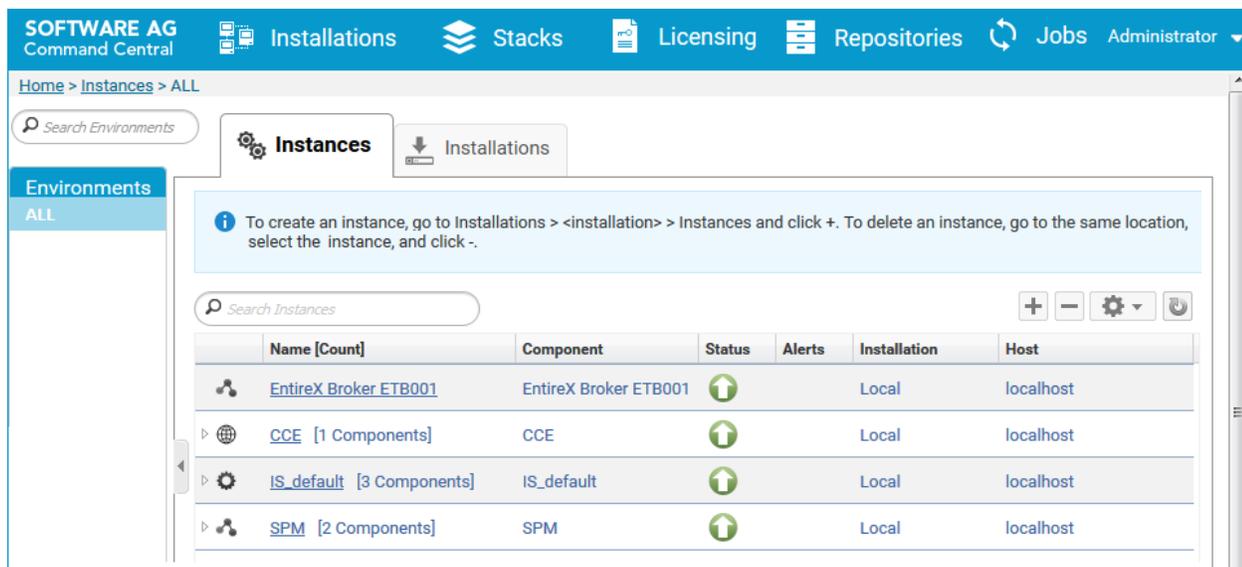
See also *Administering the RPC Server for Java using the Command Central Command Line*. The core Command Central documentation is provided separately and is also available under **Guides for Tools Shared by Software AG Products** on the Software AG documentation website.

Logging in to Command Central

Open an Internet browser and specify the URL of the Command Central Server as follows: *http://<Command_Central_host>:<Command_Central_port>*. This takes you to the Command Central **Login** page.

On Windows you can also get to the **Login** page from the Command Central Start Menu entry.

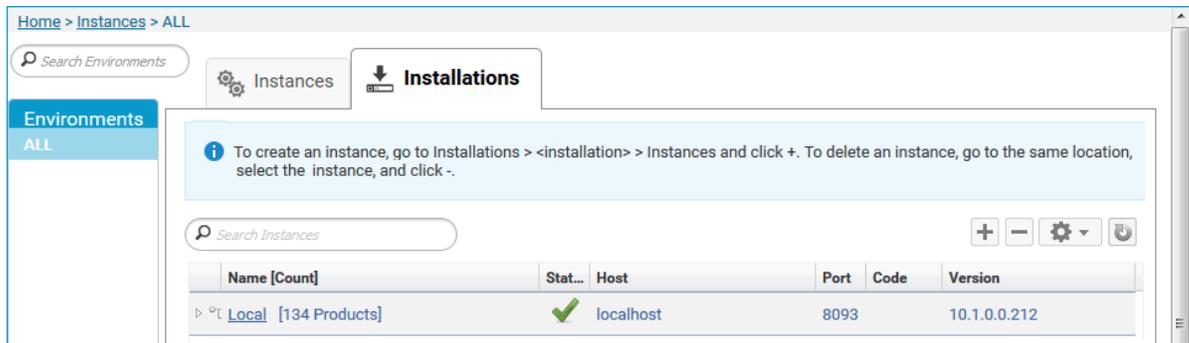
Provide your user credentials in the **Login** page and click **Log In**. This takes you to the page **Home > Instances**:



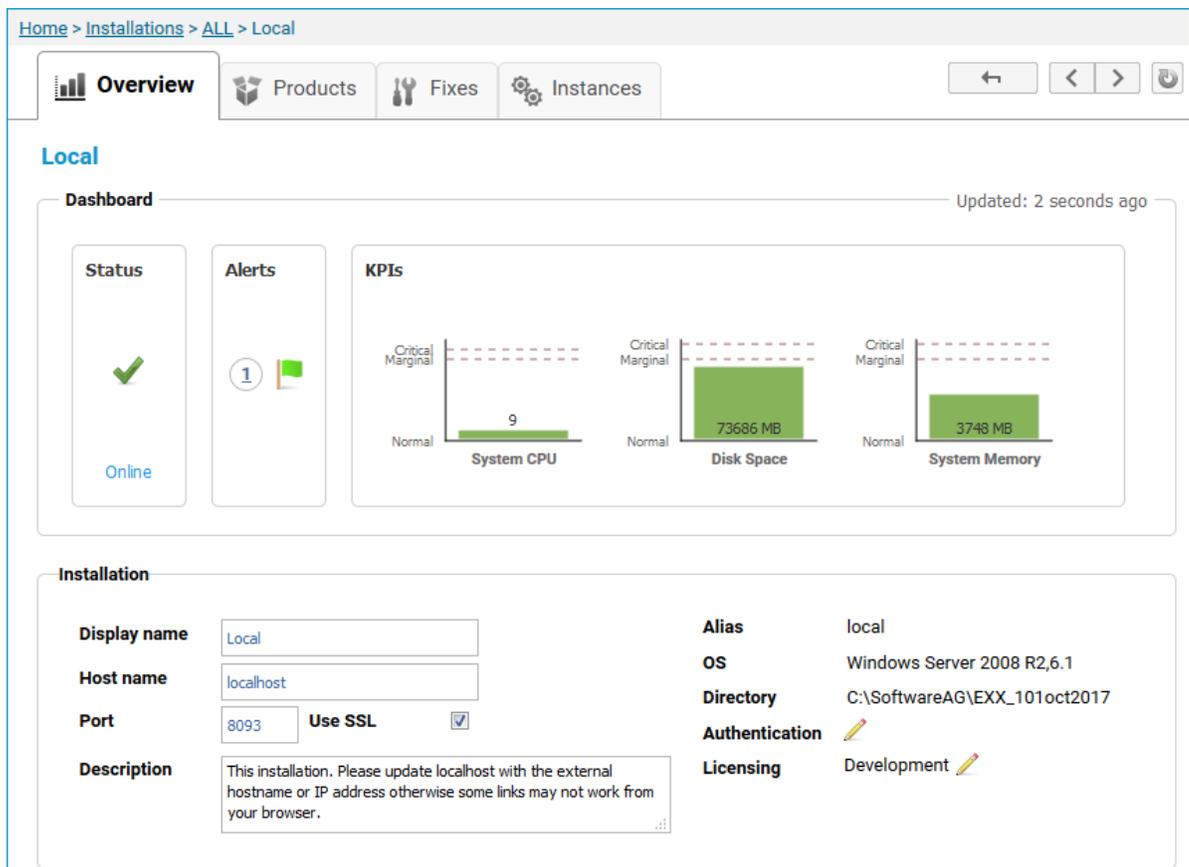
Creating an RPC Server Instance

➤ To create an RPC Server for Java instance

- 1 In the Command Central home page, click the **Installations** tab.



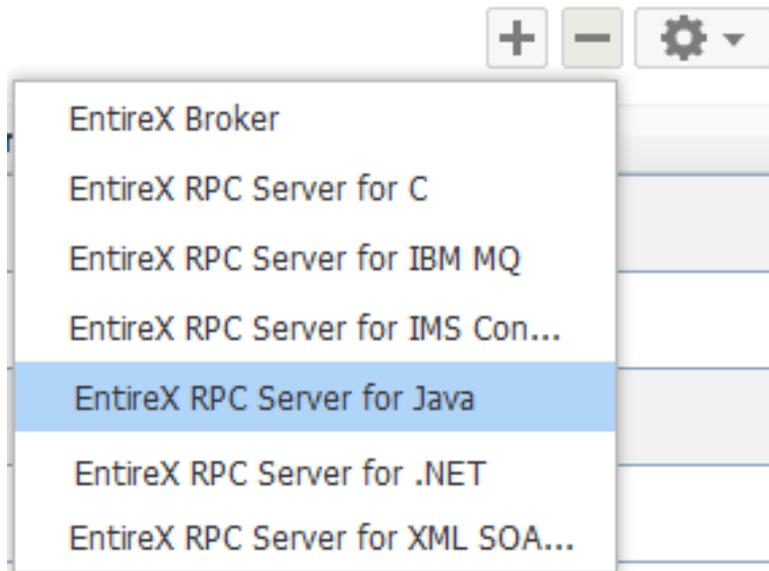
- 2 Click on the desired installation, for example **Local**, where you want to add an RPC Server for Java instance.



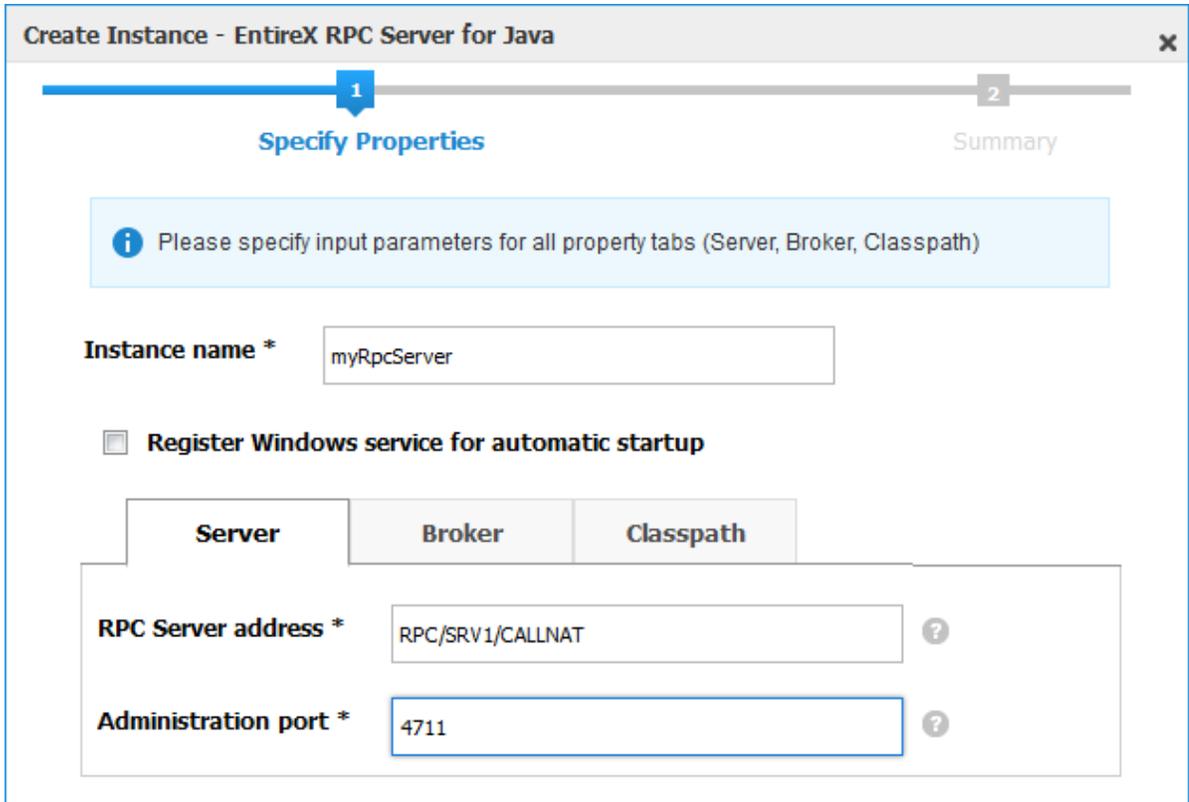
- 3 Click the **Instances** tab.

Local						
Search Instances						
Name [Count]	Component	Status	Alerts	Installation	Host	
EntireX Broker ETB001	EntireX Broker ETB001	↑		Local	localhost	
▶ CCE [1 Components]	CCE	↑		Local	localhost	
▶ IS_default [3 Components]	IS_default	↑		Local	localhost	
▶ SPM [2 Components]	SPM	↑		Local	localhost	

- 4 Click the  button in the upper right corner above the list and choose **EntireX RPC Server for Java**.



- 5 In the **Create Instance** wizard, fill in the fields in the main screen and in the **Server**, **Broker** and **Classpath** tabs.



Main Screen

Parameter	Description
Instance name	Required. Name of the runtime component, for example "MyRpcServer".
Register Windows Service for automatic startup	Optional. Register Windows Service for automatic startup. Default is not checked. If this parameter is checked, the RPC server can be controlled by the Windows Service Control Manager.

Server Tab

Parameter	Description
RPC Server address	Required. The case-sensitive RPC server address has the format: CLASS/SERVER/SERVICE.
Administration port	Required. The administration port in range from 1025 to 65535.

Broker Tab

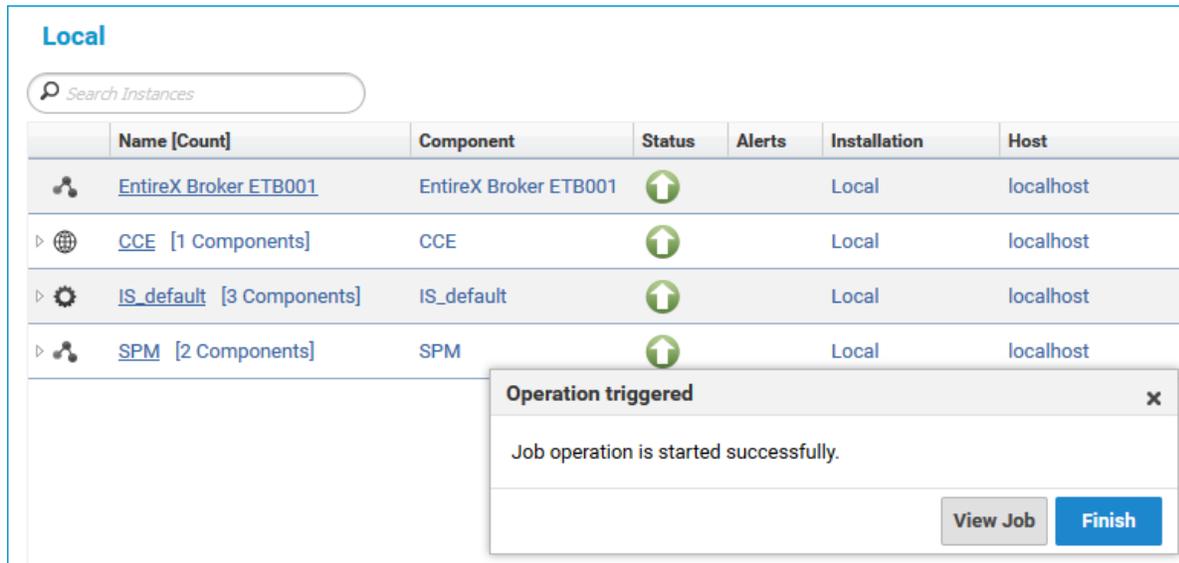
Parameter	Description
Connection	
Transport	Transport over TCP or SSL. Default is TCP.
Broker host	Required. EntireX Broker host name or IP address.
Broker port	Required. Port number in range from 1025 to 65535.
SSL trust store	Optional. Specifies the location of SSL trust store.
Credentials	
User	Optional. The user ID for secured access to the broker.
Password	Optional. The password for secured access to the broker.

Classpath Tab

Here you can modify the classpath from which the RPC Server for Java loads the server implementations.

Parameter	Description
Classpath	Required. Classpath to the RPC Server implementation.

- 6 Press **Next** to get to the **Summary** page to verify your input.
- 7 Press **Finish**.

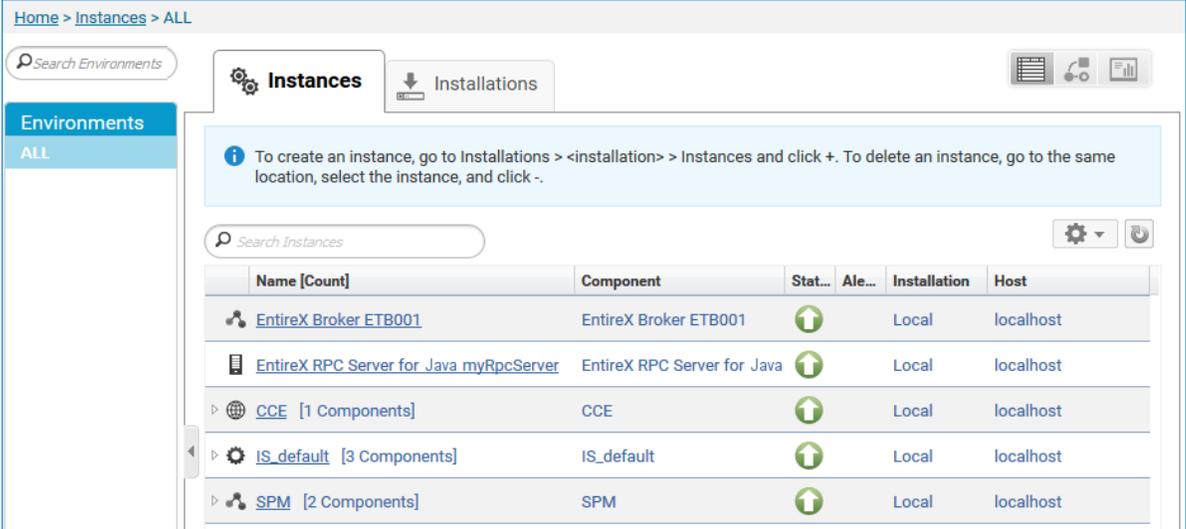


The new instance *myRpcServer* appears in the list.

Configuring an RPC Server Instance

> To configure an RPC Server for Java instance

- 1 In the Command Central home page, click the **Instances** tab.



The screenshot shows the Command Central GUI with the 'Instances' tab selected. The breadcrumb navigation is 'Home > Instances > ALL'. There is a search bar for environments and a search bar for instances. A table lists the installed components and their instances. The 'EntireX RPC Server for Java myRpcServer' instance is highlighted.

Name [Count]	Component	Stat...	Ale...	Installation	Host
EntireX Broker ETB001	EntireX Broker ETB001	↑		Local	localhost
EntireX RPC Server for Java myRpcServer	EntireX RPC Server for Java	↑		Local	localhost
▸ CCE [1 Components]	CCE	↑		Local	localhost
▸ IS_default [3 Components]	IS_default	↑		Local	localhost
▸ SPM [2 Components]	SPM	↑		Local	localhost

- 2 Click on the link associated with this instance to select the RPC server instance you want to configure.

Overview Configuration Logs Administration

Instance: EntireX RPC Server for Java myRpcServer

Dashboard Updated: 30 seconds ago

Status Online

Alerts 1

KPIs

Active Workers: 1 (Normal)

Busy Workers: 0 (Normal)

Details

Display name: EntireX RPC Server for Java myRpcSe

Component: EntireX RPC Server for Java my...

Host name: localhost

Authentication:

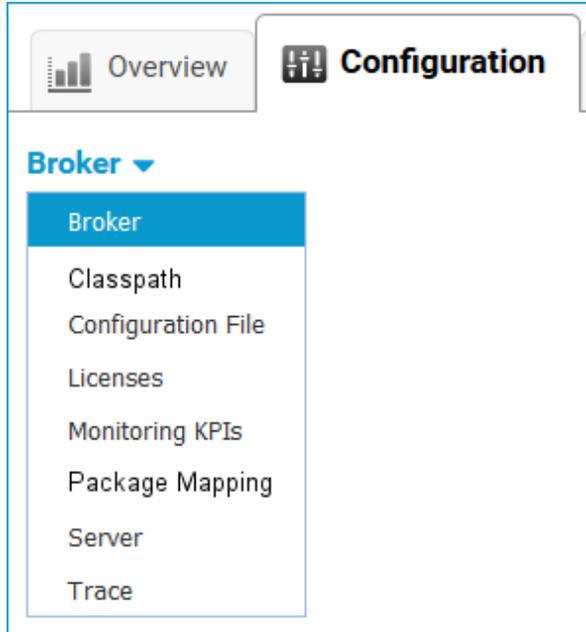
Installation name: Local

Installation alias: local

Attributes

Name	Value
------	-------

- 3 Click the **Configuration** tab. EntireX supports the following configuration types, which are presented in a drop-down box when you click the down arrow below the **Configuration** tab label:



 **Note:** All configuration changes require a restart of the instance to take effect.

- **Broker**
- **Classpath**
- **Configuration File**
- **Licenses**
- **Monitoring KPIs**
- **Package Mapping**
- **Server**
- **Trace Level**

Broker

Parameter	Description
Connection	
Transport	Transport over TCP or SSL. Default is TCP.
Broker host	Required. EntireX Broker host name or IP address.
Broker port	Required. Port number in range from 1025 to 65535.
Encoding	Required. Encoding used for the communication between the RPC server and EntireX Broker.
SSL trust store	Optional. Specifies the location of SSL trust store.
SSL verify server	Optional. The RPC server as SSL client checks the identity of the broker as SSL server.

Parameter	Description
Credentials	
User	Optional. The user ID for secured access to the broker.
Password	Optional. The password for secured access to the broker.

Classpath

Here you can specify one or more full paths to your Java server programs.

Parameter	Description
Java Archive or Directory	Required. Classpath to the RPC Server implementation.

Configuration File

Here you can view/edit the configuration file of the RPC Server for Java.

Licences

Here you can view/set the license file in the EntireX installation. For details see *Point to the License Key for an Instance or Component* under *Working with Standalone Product Installation* in the Command Central documentation.



Note: The license file is used for all EntireX instances in this installation.

Monitoring KPIs

Here you can modify margins of monitored key performance indicators (KPIs) available for the RPC Server for Java: Active Workers and Busy Workers.

Key performance indicators (KPIs) enable you to monitor the health of your RPC Server for Java. The following KPIs help you administer, troubleshoot, and resolve performance issues:

KPI	Setting
Absolute number of Active Workers	entirex.generic.kpi.1.max=20
Critical alert relative to maximum	entirex.generic.kpi.1.critical=0.95
Marginal alert relative to maximum	entirex.generic.kpi.1.marginal=0.80
Absolute number of Busy Workers	entirex.generic.kpi.2.max=20
Critical alert relative to maximum	entirex.generic.kpi.2.critical=0.95
Marginal alert relative to maximum	entirex.generic.kpi.2.marginal=0.80

Do not change the other properties!

Package Mapping

Here you can modify how the RPC Server for Java handles server programs with package names. The package name can be configured for each IDL library (see `library-definition` under *Software AG IDL Grammar* in the IDL Editor documentation).

Parameter	Description
IDL Library	Optional. IDL library name for server implementation if RPC Server for Java handles these server programs with package names.
Java Package	Optional. Java package name for server implementation if RPC Server for Java handles these server programs with package names.

Server

Here you can specify the RPC Server settings.

Parameter	Description
RPC Server	
RPC Server address	Required. The case-sensitive RPC server address has the format: CLASS/SERVER/SERVICE.
Administration port	Required. The administration port in range from 1025 to 65535.
Reconnection attempts	Required. Number of reconnection attempts to the broker. When the number of attempts is reached and a connection to the broker is not possible, the RPC Server for Java stops.
Worker Scalability	
Worker model	You can either have a fixed or dynamic number of workers. Default is <code>dynamic</code> (<code>true</code>). For more information see Worker Models .
Fixed number	Required. Fixed number of workers. Must be a number in range from 1 to 255.
Minimum number	Required. Minimum number of workers. Must be a number in range from 1 to 255.
Maximum number	Required. Maximum number of workers. Must be a number in range from 1 to 255.

Trace Level

Here you can set the trace level of the RPC Server for Java.

Parameter	Value	Description
Trace level	0 - 3	One of the following levels: 0 - None - No trace output (default). 1 - Standard - Minimal trace output. 2 - Advanced - Detailed trace output. 3 - Support - Support diagnostic. Use only when requested by Software AG support.

- 4 Click **Edit** to modify the parameters on your selected configuration type.
- 5 Click **Test** to check the correctness of your input or **Apply** to save your changes.

Viewing the Runtime Status

➤ To view the runtime status of the RPC server instance

- In the Command Central **Home** page, click the **Instances** tab and select the RPC Server for Java instance for which you want to see the runtime status (same as Step 1 under *Configuring a Broker Instance*).



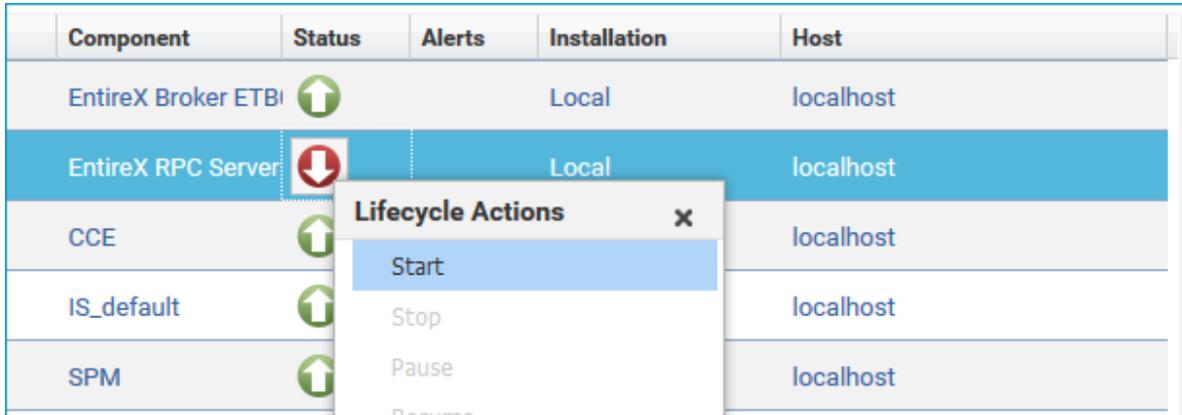
The visual key performance indicators (KPIs) and alerts enable you to monitor the RPC Server for Java's health.

KPI	Description
Active Workers	Number of active workers.
Busy Workers	Number of busy workers.

Starting an RPC Server Instance

➤ To start an RPC Server for Java instance from the Instances tab

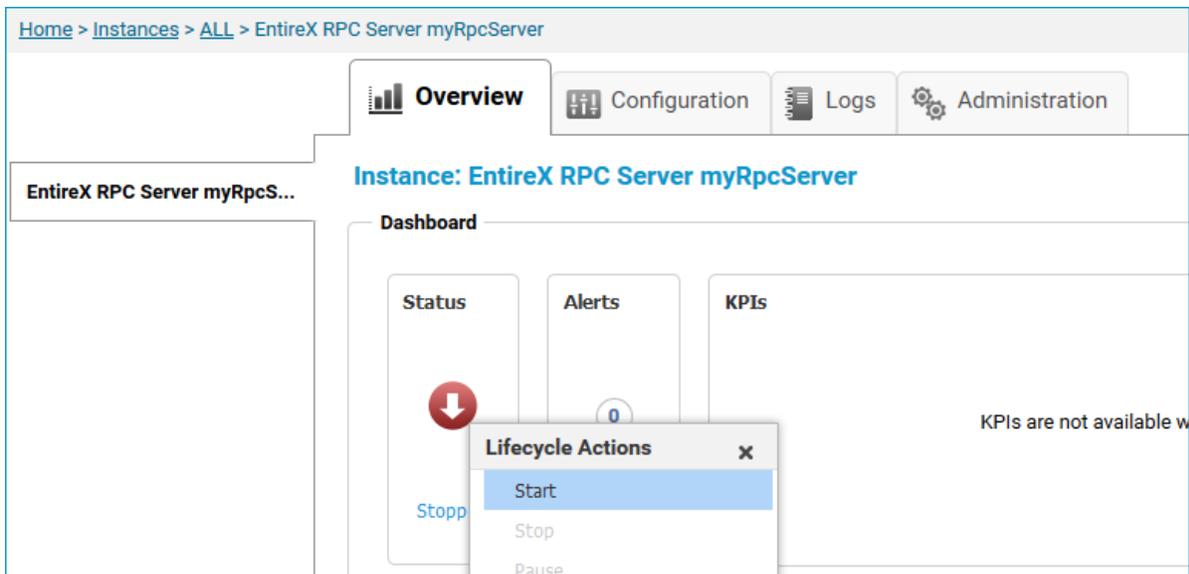
- 1 In the Command Central home page, click the **Instances** tab.



- 2 Select the status, and from the context menu choose **Start**.

➤ To start an RPC Server for Java instance from its Overview tab

- 1 In the Command Central home page, click the **Instances** tab and select the RPC Server for Java instance you want to start (same as Step 1 under *Configuring a Broker Instance*).

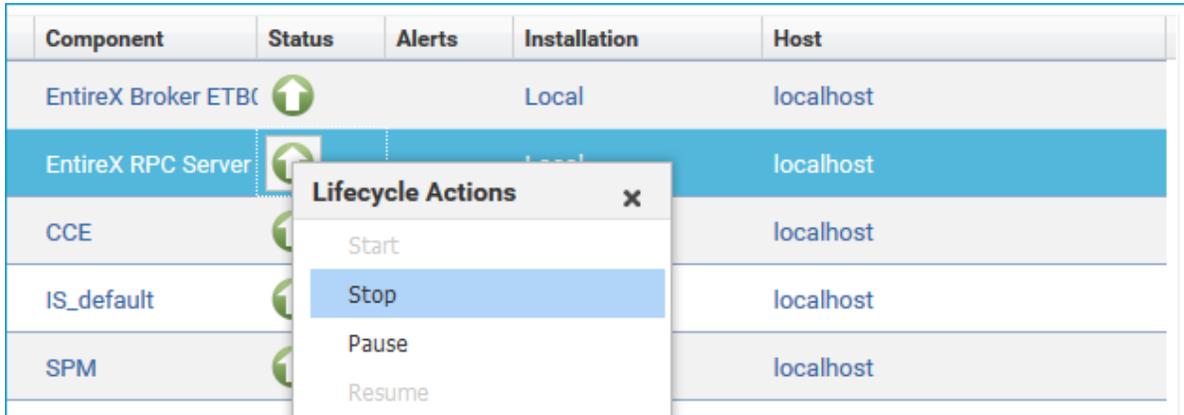


- 2 Select the status, and from the context menu choose **Start**.

Stopping an RPC Server Instance

➤ To stop an RPC Server for Java instance from the Instances tab

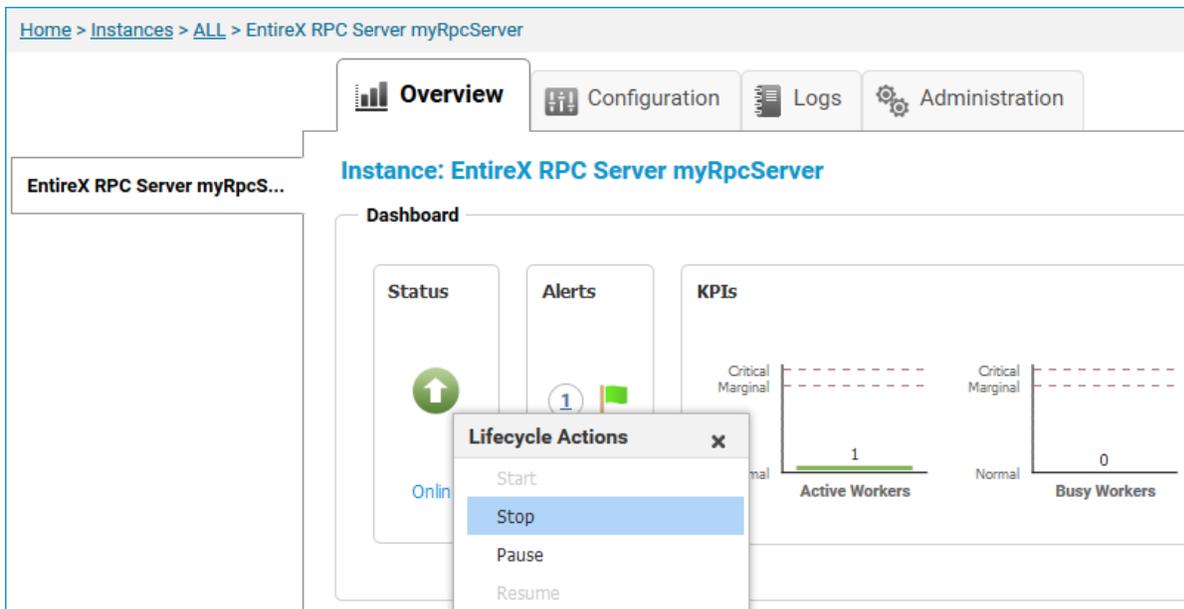
- 1 In the Command Central home page, click the **Instances** tab.



- 2 Select the status, and from the context menu choose **Stop**.

➤ To stop an RPC Server for Java instance from its Overview tab

- 1 In the Command Central home page, click the **Instances** tab and select the RPC Server for Java instance you want to stop (same as Step 1 under *Configuring a Broker Instance*).

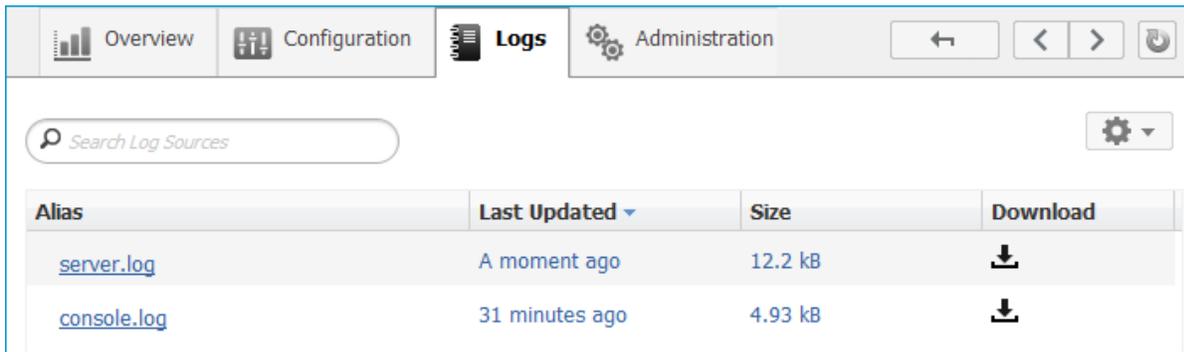


- 2 Select the status, and from the context menu choose **Stop**.

Inspecting the Log Files

➤ To inspect the log files of an RPC Server for Java instance

- 1 In the Command Central home page, click the **Instances** tab, then click the link associated with the RPC Server for Java instance for which you want to inspect the log files (same as Step 1 under *Configuring a Broker Instance*).
- 2 Click the **Logs** tab:



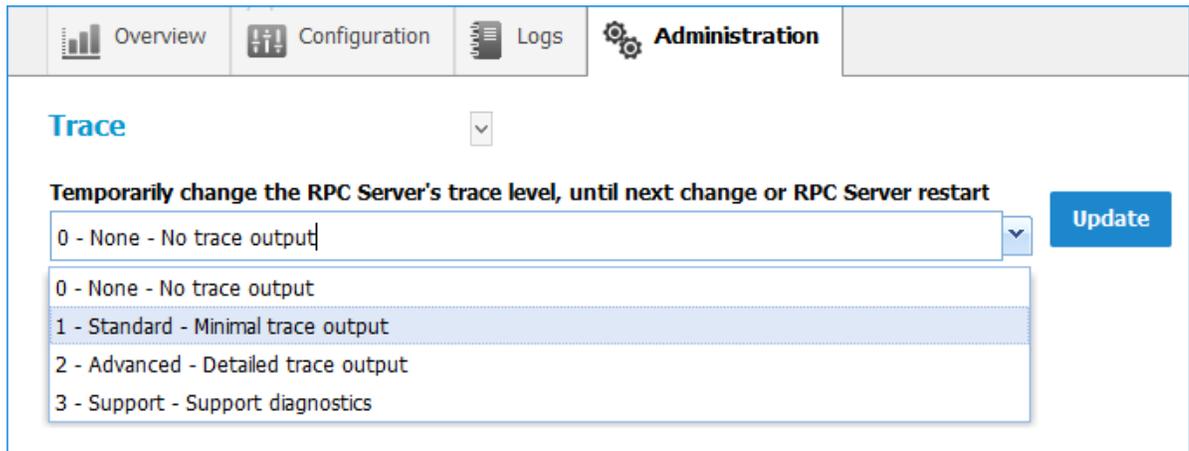
- 3 In the **Alias** column, click the link of the log file you want to inspect, for example *server.log*:



Changing the Trace Level Temporarily

➤ To temporarily change the trace level of an RPC Server for Java instance

- 1 In the Command Central home page, click the **Instances** tab then click the link associated with the RPC Server for Java instance for which you want change the trace level temporarily (same as Step 1 under *Configuring a Broker Instance*).
- 2 In the **Administration** tab, select the trace level and press **Update**.

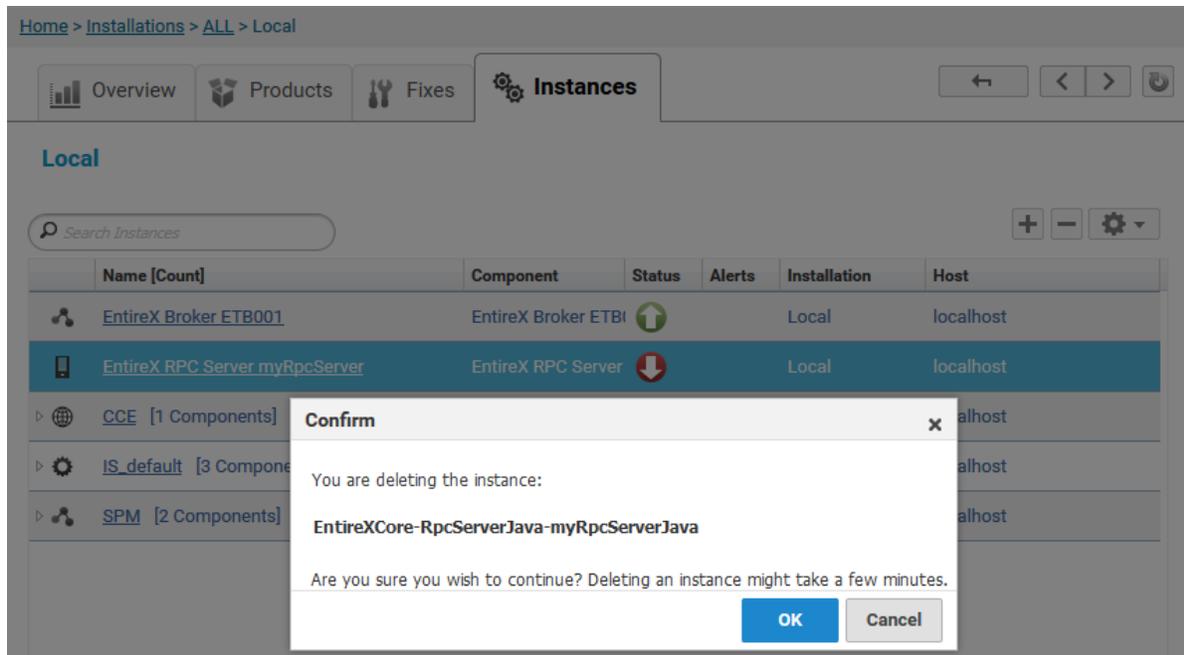


 **Note:** If you want to set the trace level permanently, see [Trace Level](#) under *Configuring an RPC Server Instance*.

Deleting an RPC Server Instance

➤ To delete an RPC Server for Java instance

- 1 In the list of EntireX RPC Server for Java instances for your selected installation (for example Local), select the instance you want to delete and click the  button in the upper right corner above the list.



- 2 Click **OK** to confirm the uninstall of this RPC Server for Java instance.
- 3 In the next window, click **Finish**. The selected instance is removed from the list.

4 Administering the RPC Server for Java using the Command

Central Command Line

▪ Creating an RPC Server Instance	30
▪ Configuring an RPC Server Instance	31
▪ Displaying the EntireX Inventory	47
▪ Viewing the Runtime Status	49
▪ Starting an RPC Server Instance	50
▪ Stopping an RPC Server Instance	50
▪ Inspecting the Log Files	51
▪ Changing the Trace Level Temporarily	53
▪ Deleting an RPC Server Instance	54

This chapter describes how to administer the EntireX RPC Server for Java, using the Command Central command-line interface.

Administering the RPC Server for Java using the Command Central GUI is described under [Administering the RPC Server for Java using the Command Central GUI](#). The core Command Central documentation is provided separately and is also available under **Guides for Tools Shared by Software AG Products** on the Software AG documentation website.

Creating an RPC Server Instance

The following table lists the parameters to include when creating an EntireX RPC instance, using the Command Central `create instances` commands.

Command	Parameter	Value	Description
sagcc create instances	<i>node_alias</i>	<i>name</i>	Required. Specifies the alias name of the installation in which the runtime component is installed.
	<i>type</i>	RpcServerJava	Required. EntireXCore instance type of RPC server. Must be "RpcServerJava".
	<i>product</i>	EntireXCore	Required. Must be set to "EntireXCore".
	<i>instance.name</i>	<i>name</i>	Required. Name of the runtime component, for example "MyRpcServer".
	<i>install.service</i>	true <u>false</u>	Optional. Register Windows Service for automatic startup. Default is false. If this parameter is true, the RPC server can be controlled by the Windows Service Control Manager.
	<i>server.address</i>	<i>class/server/service</i>	Required. The case-sensitive RPC server address has the format: CLASS/SERVER/SERVICE.
	<i>server.adminport</i>	1025-65535	Required. The administration port in range from 1025 to 65535.
	<i>broker.transport</i>	ssl <u>tcp</u>	Transport over TCP or SSL. Default is TCP.
	<i>broker.host</i>	<i>name</i>	Required. EntireX Broker host name or IP address.
	<i>broker.port</i>	1025-65535	Required. Port number in range from 1025 to 65535.
	<i>broker.user</i>	<i>user</i>	Optional. The user ID for secured access to the broker.
	<i>broker.password</i>	<i>password</i>	Optional. The password for secured access to the broker.

Command	Parameter	Value	Description
	classpath	<i>name</i>	Required. Classpath to the RPC Server implementation.

Example

- To create a new instance for an installed EntireX of the type "RpcServerJava", with name "MyRpcServer", with server address "RPC/SRV1/CALLNAT", using administration port 5757, with broker host name "localhost", listening on broker port 1971, with classpath "c:/myServer", in the installation with alias name "local":

```
sagcc create instances local EntireXCore type=RpcServerJava
instance.name=MyRpcServer server.address=RPC/SRV1/CALLNAT server.adminport=5757
broker.host=localhost broker.port=1971 classpath=c:/myServer
```

Information about the creation job - including the job ID - is displayed.

Configuring an RPC Server Instance

Here you can administer the parameters of the RPC Server for Java. Any changes to parameters will be used the next time you start the RPC server.

- [Broker](#)
- [Classpath](#)
- [Configuration File](#)
- [Monitoring KPIs](#)
- [Package Mapping](#)
- [Server](#)
- [Trace Level](#)

Broker

Here you can administer the parameters used for communication between the RPC Server for Java and EntireX Broker.

- [Parameters](#)
- [Displaying the Broker Settings of the RPC Server](#)

- Updating the Broker Settings of the RPC Server

Parameters

Parameter	Value	Description
BrokerTransport	TCP SSL	Transport over TCP or SSL. Default is TCP.
BrokerHost	<i>name</i>	Required. EntireX Broker host name or IP address.
BrokerPort	1025-65535	Required. Port number in range from 1025 to 65535.
BrokerUser	<i>user</i>	Optional. The user ID for secured access to the broker.
BrokerPassword	<i>password</i>	Optional. The password for secured access to the broker.
BrokerEncoding	<i>codepage</i>	Required. Encoding used for the communication between the RPC server and EntireX Broker.
BrokerSslTrustStore	<i>filename</i>	Optional. Specifies the location of SSL trust store.
BrokerSslVerifyServer	true false	Optional. The RPC server as SSL client checks the identity of the broker as SSL server.

Displaying the Broker Settings of the RPC Server

Command	Parameter	Description
sagcc get configuration data	<i>node_alias</i>	Required. Specifies the alias name of the installation in which the runtime component is installed.
	<i>componentid</i>	Required. The component identifier. The prefix is "EntireXCore-RpcServerJava-".
	<i>instanceid</i>	Required. Must be "BROKER".
	<i>-o file</i>	Optional. Specifies the file where you want the output written.

Example 1

- To display the Broker parameters of the RPC Server for Java "MyRpcServer" in the installation with alias name "local":

```
sagcc get configuration data local EntireXCore-RpcServerJava-MyRpcServer BROKER
```

Example 2

- To store the Broker parameters in the file *broker.json* in the current working directory:

```
sagcc get configuration data local EntireXCore-RpcServerJava-MyRpcServer BROKER
-o broker.json
```

Resulting output file in JSON format:

```
{
  "BrokerHost": "localhost",
  "BrokerPort": "1971",
  "BrokerTransport": "TCP",
  "BrokerUser": "testuser",
  "BrokerPassword": "",
  "BrokerEncoding": "Cp1252",
  "BrokerSslTrustStore": "",
  "BrokerSslVerifyServer": "true"
}
```

Updating the Broker Settings of the RPC Server

Command	Parameter	Description
sagcc update configuration data	<i>node_alias</i>	Required. Specifies the alias name of the installation in which the runtime component is installed.
	<i>componentid</i>	Required. The component identifier. The prefix is "EntireXCore-RpcServerJava-".
	<i>instanceid</i>	Required. Must be "BROKER".
	<i>-i file</i>	Optional. Specifies the file from where you want the input read.

Example

- To load the Broker parameters of the RPC Server for Java "MyRpcServer" in the installation with alias name "local" from the file *broker.json* in the current working directory:

```
sagcc update configuration data local EntireXCore-RpcServerJava-MyRpcServer BROKER
-i broker.json
```

See [Example 2](#) above for sample input file.

Classpath

Here you can modify the classpath from which the RPC Server for Java loads the server implementations.

- [Parameters](#)
- [Displaying the Classpath of the Server Implementation](#)
- [Updating the Classpath of the Server Implementation](#)

Parameters

Parameter	Description
ClasspathList	Enclosing parameter for list of items. The parameter has no value.
Classpath	Classpath to the RPC Server implementation. Note: The list of Classpath items is enclosed by parameter ClasspathList.

Displaying the Classpath of the Server Implementation

Command	Parameter	Description
sagcc get configuration data	<i>node_alias</i>	Required. Specifies the alias name of the installation in which the runtime component is installed.
	<i>componentid</i>	Required. The component identifier. The prefix is "EntireXCore-RpcServerJava-".
	<i>instanceid</i>	Required. Must be "CLASSPATH".
	<i>-o file</i>	Optional. Specifies the file where you want the output written.

Example 1

- To display the classpath parameters of the RPC Server for Java "MyRpcServer" in the installation with alias name "local":

```
sagcc get configuration data local EntireXCore-RpcServerJava-MyRpcServer CLASSPATH
```

Example 2

- To store the classpath parameters in the file *classpath.json* in the current working directory:

```
sagcc get configuration data local EntireXCore-RpcServerJava-MyRpcServer CLASSPATH
-o classpath.json
```

Resulting output file in JSON format:

```
{"ClasspathList":[
{"Classpath":"file:/c:/sampleImpl111"},
{"Classpath":"file:/c:/exampleImpl222"}
]}
```

Updating the Classpath of the Server Implementation

Command	Parameter	Description
sagcc update configuration data	<i>node_alias</i>	Required. Specifies the alias name of the installation in which the runtime component is installed.
	<i>componentid</i>	Required. The component identifier. The prefix is "EntireXCore-RpcServerJava-".
	<i>instanceid</i>	Required. Must be "CLASSPATH".
	<i>-i file</i>	Optional. Specifies the file from where you want the input read.

Example

- To load the classpath parameters of the RPC Server for Java "MyRpcServer" in the installation with alias name "local" from the file *classpath.json* in the current working directory:

```
sagcc get configuration data local EntireXCore-RpcServerJava-MyRpcServer CLASSPATH
-i classpath.json
```

See [Example 2](#) above for sample output file.

Configuration File

Here you can administer the configuration file of the RPC Server for Java. Any changes will take effect after the next restart.

- [Displaying the Content of the RPC Server Configuration File](#)
- [Updating the Content of the RPC Server Configuration File](#)

Displaying the Content of the RPC Server Configuration File

Command	Parameter	Description
sagcc get configuration data	<i>node_alias</i>	Required. Specifies the alias name of the installation in which the runtime component is installed.
	<i>componentid</i>	Required. The component identifier. The prefix is "EntireXCore-RpcServerJava-".
	<i>instanceid</i>	Required. Must be "CONFIGURATION".
	<i>-o file</i>	Optional. Specifies the file where you want the output written.

Example 1

- To display the configuration file of the RPC Server for Java "MyRpcServer" in the installation with alias name "local":

```
sagcc get configuration data local EntireXCore-RpcServerJava-MyRpcServer
CONFIGURATION
```

Example 2

- To store the contents of the configuration file in the text file *configuration.txt* in the current working directory:

```
sagcc get configuration data local EntireXCore-RpcServerJava-MyRpcServer
CONFIGURATION -o configuration.txt
```

Updating the Content of the RPC Server Configuration File

Command	Parameter	Description
sagcc update configuration data	<i>node_alias</i>	Required. Specifies the alias name of the installation in which the runtime component is installed.
	<i>componentid</i>	Required. The component identifier. The prefix is "EntireXCore-RpcServerJava-".
	<i>instanceid</i>	Required. Must be "CONFIGURATION".
	<i>-i file</i>	Optional. Specifies the file from where you want the input read.

Example

- To load the contents of configuration file *configuration.json* in the current working directory:

```
sagcc update configuration data local EntireXCore-RpcServerJava-MyRpcServer
CONFIGURATION -i configuration.json
```

Monitoring KPIs

Here you can administer margins of monitored key performance indicators (KPIs) available for the RPC Server for Java: Active Workers and Busy Workers.

- [Parameters](#)
- [Displaying the Monitoring KPIs](#)
- [Updating the Monitoring KPIs](#)

Parameters

Key performance indicators (KPIs) enable you to monitor the health of your RPC Server for Java. The following KPIs help you administer, troubleshoot, and resolve performance issues:

KPI	Setting
Absolute number of Active Workers	entirex.generic.kpi.1.max=20
Critical alert relative to maximum	entirex.generic.kpi.1.critical=0.95
Marginal alert relative to maximum	entirex.generic.kpi.1.marginal=0.80
Absolute number of Busy Workers	entirex.generic.kpi.2.max=20
Critical alert relative to maximum	entirex.generic.kpi.2.critical=0.95
Marginal alert relative to maximum	entirex.generic.kpi.2.marginal=0.80

Do not change the other properties!

Displaying the Monitoring KPIs

Command	Parameter	Description
sagcc get configuration data	<i>node_alias</i>	Required. Specifies the alias name of the installation in which the runtime component is installed.
	<i>componentid</i>	Required. The component identifier. The prefix is "EntireXCore-RpcServerJava-".
	<i>instanceid</i>	Required. Must be "EXX-MONITORING-KPIS".
	<i>-o file</i>	Optional. Specifies the file where you want the output written.

Example 1

- To display the monitoring KPI properties of RPC Server for Java "MyRpcServer" in the installation with alias name "local" on stdout:

```
sagcc get configuration data local EntireXCore-RpcServerJava-MyRpcServer
MONITORING-KPI
```

Example 2

- To store the monitoring KPI properties in the file *my.properties* in the current working directory:

```
sagcc get configuration data local EntireXCore-RpcServerJava-MyRpcServer
MONITORING-KPI -o my.properties
```

Resulting output file in text format:

```
entirex.entirex.spm.version=10.5.0.0.473
entirex.generic.kpi.1.critical=0.95
entirex.generic.kpi.1.id=\#1
entirex.generic.kpi.1.marginal=0.80
entirex.generic.kpi.1.max=20
entirex.generic.kpi.1.name=Active Workers
entirex.generic.kpi.1.unit=
entirex.generic.kpi.1.value=0
entirex.generic.kpi.2.critical=0.95
entirex.generic.kpi.2.id=\#2
entirex.generic.kpi.2.marginal=0.80
entirex.generic.kpi.2.max=20
entirex.generic.kpi.2.name=Busy Workers
entirex.generic.kpi.2.unit=
entirex.generic.kpi.2.value=0
```

Updating the Monitoring KPIs

Command	Parameter	Description
sagcc update configuration data	<i>node_alias</i>	Required. Specifies the alias name of the installation in which the runtime component is installed.
	<i>componentid</i>	Required. The component identifier. The prefix is "EntireXCore-RpcServerJava-".
	<i>instanceid</i>	Required. Must be "EXX-MONITORING-KPIS".
	<i>-i file</i>	Optional. Specifies the file from where you want the input read.

Example

- To load the contents of file *my.properties* in the current working directory:

```
sagcc update configuration data local EntireXCore-RpcServerJava-MyRpcServer  
MONITORING-KPI -i my.properties
```

Package Mapping

Here you can modify how the RPC Server for Java handles server programs with package names. The package name can be configured for each IDL library (see `library-definition` under *Software AG IDL Grammar* in the IDL Editor documentation).



Note: A package name can be specified when the server is generated. See *Preferences* and *Properties* under *Using the Java Wrapper*.

- [Parameters](#)
- [Displaying the Package Mapping](#)
- [Updating the Package Mapping](#)

Parameters

Parameter	Description
PackageList	Enclosing parameter for list of (<code>idlLibrary</code> , <code>javaPackage</code>) parameter pairs, the parameter has no value.
idlLibrary	IDL library name for server implementation if RPC Server for Java handles these server programs with package names.
javaPackage	Java package name for server implementation if RPC Server for Java handles these server programs with package names.

Displaying the Package Mapping

Command	Parameter	Description
sagcc get configuration data	<code>node_alias</code>	Required. Specifies the alias name of the installation in which the runtime component is installed.
	<code>componentid</code>	Required. The component identifier. The prefix is "EntireXCore-RpcServerJava-".
	<code>instanceid</code>	Required. Must be "PACKAGE-MAPPING".
	<code>-o file</code>	Optional. Specifies the file where you want the output written.

Example 1

- To display the package mapping parameters of RPC Server for Java "MyRpcServer" in the installation with alias name "local" on stdout:

```
sagcc get configuration data local EntireXCore-RpcServerJava-MyRpcServer
PACKAGE-MAPPING
```

Example 2

- To store the package mapping parameters in the file *packageMapping.json* in the current working directory:

```
sagcc get configuration data local EntireXCore-RpcServerJava-MyRpcServer
PACKAGE-MAPPING -o packageMapping.json
```

Resulting output file in JSON format:

```
{"PackageList":[
{"idLibrary":"example","javaPackage":"com.softwareag.example"},
{"idLibrary":"booking","javaPackage":"com.sample.booking"}
]}
```

Updating the Package Mapping

Command	Parameter	Description
sagcc update configuration data	<i>node_alias</i>	Required. Specifies the alias name of the installation in which the runtime component is installed.
	<i>componentid</i>	Required. The component identifier. The prefix is "EntireXCore-RpcServerJava-".
	<i>instanceid</i>	Required. Must be "PACKAGE-MAPPING".
	<i>-i file</i>	Optional. Specifies the file from where you want the input read.

Example

- To load the package mapping parameters from the file *packageMapping.json* in the current working directory:

```
sagcc update configuration data local EntireXCore-RpcServerJava-MyRpcServer
PACKAGE-MAPPING -i packageMapping.json
```

See [Example 2](#) above for sample input file.

Server

Here you can administer the parameters defining the registration name, the administration port and the behavior of the RPC Server for Java.

- [Parameters](#)
- [Displaying the Server Settings](#)
- [Updating the Server Settings](#)

Parameters

Parameter	Value	Description
ServerAddress	<i>class/server/service</i>	Required. The case-sensitive RPC server address has the format: CLASS/SERVER/SERVICE.
ServerAdminport	1025-65535	Required. The administration port in range from 1025 to 65535.
ReconnectionAttempts	<i>n</i>	Required. Number of reconnection attempts to the broker. When the number of attempts is reached and a connection to the broker is not possible, the RPC Server for Java stops.
WorkerScalability	<i>true</i> <i>false</i>	You can either have a fixed or dynamic number of workers. Default is dynamic (<i>true</i>). For more information see Worker Models .
FixNumber	1-255	Required. Fixed number of workers. Must be a number in range from 1 to 255.
MinWorkers	1-255	Required. Minimum number of workers. Must be a number in range from 1 to 255.
MaxWorkers	1-255	Required. Maximum number of workers. Must be a number in range from 1 to 255.

Displaying the Server Settings

Command	Parameter	Description
sagcc get configuration data	<i>node_alias</i>	Required. Specifies the alias name of the installation in which the runtime component is installed.
	<i>componentid</i>	Required. The component identifier. The prefix is "EntireXCore-RpcServerJava-".
	<i>instanceid</i>	Required. Must be "SERVER".
	<i>-o file</i>	Optional. Specifies the file where you want the output written.

Example 1

- To display the server parameters of RPC Server for Java "MyRpcServer" in the installation with alias name "local" on stdout:

```
sagcc get configuration data local EntireXCore-RpcServerJava-MyRpcServer SERVER
```

Example 2

- To store the server parameters in the file *server.json* in the current working directory:

```
sagcc get configuration data local EntireXCore-RpcServerJava-MyRpcServer SERVER
-o server.json
```

Resulting output file in JSON format:

```
{
  "ServerAddress": "RPC/SRV1/CALLNAT",
  "ServerAdminport": "4711",
  "ReconnectionAttempts": "15",
  "WorkerScalability": "true",
  "FixNumber": "5",
  "MinWorkers": "1",
  "MaxWorkers": "10"
}
```

Updating the Server Settings

Command	Parameter	Description
sagcc update configuration data	<i>node_alias</i>	Required. Specifies the alias name of the installation in which the runtime component is installed.
	<i>componentid</i>	Required. The component identifier. The prefix is "EntireXCore-RpcServerJava-".
	<i>instanceid</i>	Required. Must be "SERVER".
	<i>-i file</i>	Optional. Specifies the file from where you want the input read.

Example

- To load the server parameters from the file *server.json* in the current working directory:

```
sagcc update configuration data local EntireXCore-RpcServerJava-MyRpcServer SERVER  
-i server.json
```

See [Example 2](#) above for sample input file.

Trace Level

Here you can set the trace level of the RPC Server for Java.

- [Parameters](#)
- [Displaying the Trace Level](#)
- [Updating the Trace Level](#)

Parameters

Parameter	Value	Description
TraceLevel	0 1 2 3	One of the following levels: 0 - None - No trace output (default). 1 - Standard - Minimal trace output. 2 - Advanced - Detailed trace output. 3 - Support - Support diagnostic. Use only when requested by Software AG support.

Displaying the Trace Level

Command	Parameter	Description
sagcc get configuration data	<i>node_alias</i>	Required. Specifies the alias name of the installation in which the runtime component is installed.
	<i>componentid</i>	Required. The component identifier. The prefix is "EntireXCore-RpcServerJava-".
	<i>instanceid</i>	Required. Must be "TRACE".
	<i>-o file</i>	Optional. Specifies the file where you want the output written.

Example 1

- To display the trace level of RPC Server for Java "MyRpcServer" in the installation with alias name "local" on stdout:

```
sagcc get configuration data local EntireXCore-RpcServerJava-MyRpcServer TRACE
```

Example 2

- To store the trace level in the file *trace.json* in the current working directory:

```
sagcc get configuration data local EntireXCore-RpcServerJava-MyRpcServer TRACE
-o trace.json
```

Resulting output file in JSON format:

```
{
  "TraceLevel": "0"
}
```

Updating the Trace Level

Command	Parameter	Description
sagcc update configuration data	<i>node_alias</i>	Required. Specifies the alias name of the installation in which the runtime component is installed.
	<i>componentid</i>	Required. The component identifier. The prefix is "EntireXCore-RpcServerJava-".
	<i>instanceid</i>	Required. Must be "TRACE".
	<i>-i file</i>	Optional. Specifies the file from where you want the input read.

Example

- To load the trace level parameters from the file *trace.json* in the current working directory:

```
sagcc update configuration data local EntireXCore-RpcServerJava-MyRpcServer TRACE
-i trace.json
```

See [Example 2](#) above for sample input file.

Displaying the EntireX Inventory

Listing all Inventory Components

The following table lists the parameters to include, when listing all EntireX instances, using the Command Central `list inventory` commands.

Command	Parameter	Description
sagcc list inventory components	<i>node_alias</i>	Required. Specifies the alias name of the installation in which the runtime component is installed.
	<i>component_id</i>	Required. The component identifier. The prefix is "EntireXCore-RpcServerJava-".

Example

- To list inventory components of instance EntireX in the installation with alias name "local":

```
sagcc list inventory components local EntireXCore*
```

A list of all EntireX RPC Server runtime components will be displayed.

Viewing the Runtime Status

The following table lists the parameters to include when displaying the state of an EntireX component, using the Command Central `get monitoring` commands.

Command	Parameter	Description
sagcc get monitoring state	<i>node_alias</i>	Required. Specifies the alias name of the installation in which the runtime component is installed.
	<i>componentid</i>	Required. The component identifier. The prefix is "EntireXCore-RpcServerJava-".

Example

- To display state information about the RPC Server for Java:

```
sagcc get monitoring state local EntireXCore-RpcServerJava-MyRpcServer
```

Runtime status and runtime state will be displayed.

- Runtime *status* indicates whether a runtime component is running or not. Examples of a runtime status are ONLINE or STOPPED.
- Runtime *state* indicates the health of a runtime component by providing key performance indicators (KPIs) for the component. Each KPI provides information about the current use, marginal use, critical use and maximum use.

Starting an RPC Server Instance

The following table lists the parameters to include when starting an EntireX RPC Server for Java, using the Command Central `exec lifecycle` commands.

Command	Parameter	Description
sagcc exec lifecycle start	<i>node_alias</i>	Required. Specifies the alias name of the installation in which the runtime component is installed.
	<i>componentid</i>	Required. The component identifier. The prefix is "EntireXCore-RpcServerJava-".

Example

- To start the RPC Server for Java "MyRpcServer" in the installation with alias name "local":

```
sagcc exec lifecycle start local EntireXCore-RpcServerJava-MyRpcServer
```

Information about the job - including the job ID - will be displayed.

Stopping an RPC Server Instance

The following table lists the parameters to include when stopping an EntireX RPC Server for Java, using the Command Central `exec lifecycle` commands.

Command	Parameter	Description
sagcc exec lifecycle stop	<i>node_alias</i>	Required. Specifies the alias name of the installation in which the runtime component is installed.
	<i>componentid</i>	Required. The component identifier. The prefix is "EntireXCore-RpcServerJava-".

Example

- To stop the RPC Server for Java "MyRpcServer" in the installation with alias name "local":

```
sagcc exec lifecycle stop local EntireXCore-RpcServerJava-MyRpcServer
```

Information about the job - including the job ID - will be displayed.

Inspecting the Log Files

Here you can administer the log files of the RPC Server for Java. The following table lists the parameters to include when displaying or modifying parameters of the RPC server, using the Command Central `list` commands.

- [List all RPC Server Log Files](#)
- [Getting Content from or Downloading RPC Server Log Files](#)

List all RPC Server Log Files

Command	Parameter	Description
sagcc list diagnostics logs	<i>node_alias</i>	Required. Specifies the alias name of the installation in which the runtime component is installed.
	<i>componentid</i>	Required. The component identifier. The prefix is "EntireXCore-RpcServerJava-".

Example

- To list the log files of RPC Server for Java "MyRpcServer" in the installation with alias name "local" on stdout:

```
sagcc list diagnostics logs local EntireXCore-RpcServerJava-MyRpcServer
```

Getting Content from or Downloading RPC Server Log Files

Command	Parameter	Description
sagcc get diagnostics logs	<i>node_alias</i>	Required. Specifies the alias name of the installation in which the runtime component is installed.
	<i>componentid</i>	Required. The component identifier. The prefix is "EntireXCore-RpcServerJava-".
	full tail head	Optional. Shows full log file content, or only tail or head.
	export -o <i>file</i>	Optional. Creates a zip file of the logs.

Example 1

- To list the tail of the log file content in the current working directory:

```
sagcc get diagnostics logs local EntireXCore-RpcServerJava-MyRpcServer server.log  
tail
```

Example 2

- To create a zip file *myfile.zip* of the logs:

```
sagcc get diagnostics logs local EntireXCore-RpcServerJava-MyRpcServer export -o  
myfile.zip
```

Changing the Trace Level Temporarily

Here you can temporarily change the trace level of a running RPC server. The following table lists the parameters to include when displaying or modifying parameters of an EntireX component, using the Command Central `exec administration` command. The change is effective immediately; there is no need to restart the RPC server.



Note: If you want to set the trace level permanently, see [Trace Level](#) under *Configuring an RPC Server Instance*.

Displaying the Trace Level of a Running RPC Server

Command	Parameter	Description
sagcc exec administration	component	Required. Specifies that a component will be administered.
	node_alias	Required. Specifies the alias name of the installation in which the runtime component is installed.
	Trace	Required. Specifies what is to be administered.
	load tracelevel=?	Required. Get the trace level.
	-f xml json	Required. Specifies XML or JSON as output format.

Example 1

- To display the current trace level of the RPC Server for Java "MyRpcServer" in the installation with alias name "local" in JSON format on stdout:

```
sagcc exec administration component local EntireXCore-RpcServerJava-MyRpcServer
Trace load tracelevel=? -f json
```

Example 2

- To display the current trace level of the RPC Server for Java "MyRpcServer" in the installation with alias name "local" in XML format on stdout:

```
sagcc exec administration component local EntireXCore-RpcServerJava-MyRpcServer
Trace load tracelevel=? -f xml
```

Updating the Trace Level of a Running RPC Server

Command	Parameter	Description
sagcc exec administration	component	Required. Specifies that a component will be administered.
	<i>node_alias</i>	Required. Specifies the alias name of the installation in which the runtime component is installed.
	<i>componentid</i>	Required. The component identifier. The prefix is "EntireXCore-RpcServerJava-".
	Trace	Required. Specifies what is to be administered.
	update tracelevel	Required. Update temporarily the trace level of a running RPC server.
	-f xml json	Required. Specifies XML or JSON as output format.

Example

- To change the current trace level of the running RPC Server with the name "MyRpcServer" in the installation with alias name "local":

```
sagcc exec administration component local EntireXCore-RpcServerJava-MyRpcServer
Trace update tracelevel=2 -f json
```

Deleting an RPC Server Instance

The following table lists the parameters to include when deleting an EntireX RPC Server instance, using the Command Central `delete instances` commands.

Command	Parameter	Description
sagcc delete instances	<i>node_alias</i>	Required. Specifies the alias name of the installation in which the runtime component is installed.
	<i>componentid</i>	Required. The component identifier. The prefix is "EntireXCore-RpcServerJava-".

Example

- To delete an instance of an EntireX RPC Server for Java with the name "MyRpcServer" in the installation with alias name "local":

```
sagcc delete instances local EntireXCore-RpcServerJava-MyRpcServer
```

Information about the deletion job - including the job ID - is displayed.

5

Administering the RPC Server for Java with Local Scripts

▪ Customizing the RPC Server	58
▪ Configuring the RPC Server	59
▪ Using SSL/TLS with the RPC Server	62
▪ Starting the RPC Server	63
▪ Stopping the RPC Server	63
▪ Pinging the RPC Server	64
▪ Running an EntireX RPC Server as a Windows Service	64
▪ Application Identification	65

The EntireX RPC Server for Java allows standard RPC clients to communicate with servers written in Java. It works together with the Java Wrapper and calls Java server interface objects.

This chapter describes how to administer the RPC Server for Java with local scripts as in earlier versions of EntireX.

See also *Administering the RPC Server for Java* with the Command Central [GUI](#) | [Command Line](#).

Customizing the RPC Server

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

- [Start Script](#)
- [Properties File](#)

Start Script

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

You can set the environment variable `JAVA_HOME` for the location of the Java interpreter. Set the classpath to `entirex.jar` and the path to the Java server interface objects.

The script files that start the RPC Server for Java allow you to pass properties as command-line options as described in the table below. Alternatively, you can use system properties or a property file. The command-line option has the highest priority; the system property has second priority, and the entries of a property file have third priority.

Example:

```
java -Dentirex.server.properties=rpcserver.properties ↵  
-Dentirex.license.location=<license.xml with path> -classpath <entirex.jar with ↵  
path>:<path to your server ↵  
interface objects> com.softwareag.entirex.aci.RPCServer
```

Properties File

The default name of the properties file is `entirex.server.properties`. The file is searched for in the directory of the *Start Script*. It can be changed by assigning an arbitrary file name with a path to a property with the name `entirex.server.properties`.

A sample properties file is contained in subfolder *config* of the installation folder.

Configuring the RPC Server

Property Name	Command-line Option	Explanation
<code>entirex.rpcsrvr.packagename.library_name</code>		<p>The RPC Server for Java can handle server programs with package names if the package name of each IDL library (see <i>library-definition</i>) is configured in the properties of the server. For each library the property <code>entirex.rpcsrvr.packagename.library_name</code> has the value of the package.</p> <p>Example for the library Example (as in <i>example.idl</i>):</p> <pre>entirex.rpcsrvr.packagename.example=my.package</pre> <p>The library name must be lowercase. A package name can be specified when the server is generated. See <i>Preferences</i> and <i>Properties</i> under <i>Using the Java Wrapper</i>.</p> <p>Default: localhost.</p>
<code>entirex.server.brokerid</code>	<code>-broker</code>	Broker ID.
<code>entirex.server.codepage</code>	<code>-codepage</code>	<p>The encoding configured for the Java virtual machine (JVM) is used to convert the Unicode (UTF-16) representation within Java to the encoding sent to or received from the broker by default. This encoding is also transferred as the codepage to the broker to tell the broker the encoding of the data. Changing the default encoding of the JVM has the side effect that the encoding for terminal and file IO is affected too. This may be undesired.</p> <p>With the codepage parameter you can override the encoding without the need to change the default encoding of the JVM. The codepage must be supported by your JVM. For a list of valid encodings, see <i>Supported Encodings</i> in your Java documentation.</p> <p>Note: See your JVM documentation for how to change the default encoding of the JVM. On some JVM implementations, it can be changed with the <code>file.encoding</code> property. On some UNIX</p>

Property Name	Command-line Option	Explanation														
		<p>implementations, it can be changed with the LANG environment variable.</p> <p>Enable character conversion in the broker by setting the service-specific attribute CONVERSION to "SAGTRPC". See also <i>Configuring ICU Conversion</i> under <i>Configuring Broker for Internationalization</i> in the platform-specific Administration documentation. More information can be found under <i>Internationalization with EntireX</i>.</p>														
entirex.server.compresslevel	-compresslevel	<p>Permitted values (you can enter the text or the numeric value):</p> <table border="1"> <tr> <td>BEST_COMPRESSION</td> <td>9</td> </tr> <tr> <td>BEST_SPEED</td> <td>1</td> </tr> <tr> <td>DEFAULT_COMPRESSION</td> <td>-1, mapped to 6</td> </tr> <tr> <td>DEFLATED</td> <td>8</td> </tr> <tr> <td>NO_COMPRESSION</td> <td>0</td> </tr> <tr> <td>N</td> <td>0</td> </tr> <tr> <td>Y</td> <td>8</td> </tr> </table> <p>Default: 0 (no compression).</p>	BEST_COMPRESSION	9	BEST_SPEED	1	DEFAULT_COMPRESSION	-1, mapped to 6	DEFLATED	8	NO_COMPRESSION	0	N	0	Y	8
BEST_COMPRESSION	9															
BEST_SPEED	1															
DEFAULT_COMPRESSION	-1, mapped to 6															
DEFLATED	8															
NO_COMPRESSION	0															
N	0															
Y	8															
entirex.server.customclass	-customclass	This class is used for custom initialization and shutdown of the server. In addition, this class allows handling when closing a conversation and handling the termination of a worker thread. See <i>ServerImplementation</i> for more information.														
entirex.server.fixedservers	no	<p>NO The number of worker threads balances between what is specified in <code>entirex.server.minservers</code> and what is specified in <code>entirex.server.maxservers</code>. This is done by a so-called attach thread. At startup, the number of worker threads is the number specified in <code>entirex.server.minservers</code>. A new worker thread starts if the broker has more requests than there are worker threads waiting. If more than the number specified in <code>entirex.server.minservers</code> are waiting for requests, a worker thread stops if its receive call times out. The timeout period is configured with <code>entirex.server.waitserver</code>. See worker model DYNAMIC.</p> <p>YES The number of worker threads specified in <code>entirex.server.minservers</code> is started and the server can process this number of parallel requests. See worker model FIXED.</p>														
	-help	Display usage of the command-line parameters.														
entirex.server.logfile	-logfile	Path and name of the trace output file. Environment variables in the name are resolved only if used as command-line option.														

Property Name	Command-line Option	Explanation
entirex.server.maxservers		Maximum number of worker threads. Default: 32.
entirex.server.minservers		Minimum number of server threads. Default: 1.
entirex.server.name		The name of the server.
entirex.server.password	-password	The password for secured access to the broker. The password is encrypted and written to the property <code>entirex.server.password.e</code> . To change the password, set the new password in the properties file. To disable password encryption, set <code>entirex.server.passwordencrypt=no</code> . Default: yes.
entirex.server.properties	-propertyfile	The name of the property file. Default: <code>entirex.server.properties</code> .
entirex.server.restartcycles	-restartcycles	Number of restart attempts if the Broker is not available. This can be used to keep the RPC Server for Java running while the Broker is down for a short time. Default: 15.
entirex.server.security	-security	Values: <code>no yes auto name of BrokerSecurity object</code> . Default: no.
entirex.server.serveraddress	-server	Server address. Default: <code>RPC/SRV1/CALLNAT</code> .
entirex.server.serverlog	-serverlog	Name of the file where start and stop of worker threads is logged. Used by the Windows RPC Service.
entirex.server.userid	-user	The user ID for access to the broker. Default: <code>JavaServer</code> .
entirex.server.waitattach		Wait timeout for the attach server thread. Default: 600S.
entirex.server.waitserver		Wait timeout for the worker threads. Default: 300S.
entirex.timeout		TCP/IP transport timeout. See <i>Setting the Transport Timeout</i> under <i>Writing Advanced Applications - EntireX Java ACI</i> . Default: 20
entirex.trace	-trace	Trace level: 0 1 2 3. Default: 0.

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 or Broker SSL Agent. For an introduction see *SSL/TLS and Certificates with EntireX* in the Platform-independent Administration 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 Set up the RPC Server for Java for an SSL connection.

Use the *URL-style Broker ID* with protocol `ssl://` for the Broker ID. If no port number is specified, port 1958 is used as default. Example:

```
ssl://localhost:22101?trust_store=C:\SoftwareAG\EntireX\etc\ExxCACert.jks&verify_server=no
```

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*.

- 3 Make sure the SSL server to which the RPC Server for Java connects is prepared for SSL connections as well. The SSL server can be EntireX Broker or Broker SSL Agent. See:
 - *Running Broker with SSL/TLS Transport* in the platform-specific Administration documentation
 - *Broker SSL Agent* in the UNIX and Windows Administration documentation

Starting the RPC Server

> To start the RPC Server for Java

- Use the *Start Script*.

Or:

At the command prompt, enter:

```
java com.softwareag.entirex.aci.RPCServer
```

You can pass command-line options and customize your environment as described under *Start Script*.

Stopping the RPC Server

> To stop the RPC Server for Java

- Use the command `stopService`. See *Stop Running 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`.

Pinging the RPC Server

➤ To ping the RPC Server for Java

- Enter the following command:

```
java -classpath "$EXXDIR/classes/entirex.jar" ↵  
com.softwareag.entirex.rpcping.RPCServerPing -p <admin_port>
```

where *admin_port* is the number of the administration port.

The ping command returns "0" if the server is reachable, and "1" if the server cannot be accessed.



Note: This command is particularly useful in a high availability cluster context. See *Setting up your Environment for High Availability with Container Orchestration* in the High Availability documentation.

Running an EntireX RPC Server as a Windows Service

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

➤ To run the RPC Server for Java 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 reduced signalling of the JVM (option `-Xrs`):

```
java -Xrs com.softwareag.entirex.aci.RPCServer %*
```

If `-Xrs` is not used, the JVM stops and an entry 10164002 is written to the event log when the user logs off from Windows.

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 `jrpservice.bat`, the command will be

```
RPCService -install -ext MyServer -script install_path\EntireX\bin\jrpcserver.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".

Application Identification

The application identification is sent from the RPC server to the Broker. It is visible with Broker Command and Info Services.

The identification consists of four parts: name, node, type, and version. These four parts are sent with each Broker call and are visible in the trace information.

For the RPC Server for Java these values are:

Identification Part	Value
Application name	ANAME=RPC Server for Java
Node name	ANODE= <i>host_name</i>
Application type	ATYPE=Java
Version	AVERS=10.5.0.0

6 Scenarios and Programmer Information

- Writing a New Java Server 68

Writing a New Java Server

➤ **To write a new Java server**

- 1 Use the Java Wrapper to generate a Java server interface object. See *Generating a Java Server Interface Object*. Write your Java server and proceed as described under *Generating a Java Server Interface Object*.
- 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 Designer documentation
 - generate an XML mapping file (XMM) and use the XML Tester for verification; see *EntireX XML Tester* in the XML/SOAP Wrapper documentation

7

Parameter Reference

encoding

The encoding configured for the Java virtual machine (JVM) is used to convert the Unicode (UTF-16) representation within Java to the encoding sent to or received from the broker by default. This encoding is also transferred as the codepage to the broker to tell the broker the encoding of the data. Changing the default encoding of the JVM has the side effect that the encoding for terminal and file IO is affected too. This may be undesired.

With the codepage parameter you can override the encoding without the need to change the default encoding of the JVM. The codepage must be supported by your JVM. For a list of valid encodings, see *Supported Encodings* in your Java documentation.



Note: See your JVM documentation for how to change the default encoding of the JVM. On some JVM implementations, it can be changed with the `file.encoding` property. On some UNIX implementations, it can be changed with the `LANG` environment variable.

Enable character conversion in the broker by setting the service-specific attribute `CONVERSION` to "SAGTRPC". See also *Configuring ICU Conversion* under *Configuring Broker for Internationalization* in the platform-specific Administration documentation. More information can be found under *Internationalization with EntireX*.

8

Building an EntireX RPC Server for Java Docker Image

- Prerequisites 72
- Building and Running the RPC Server for Java Image 72
- Verifying the Build 76
- Healthcheck for RPC Server for Java 77

You can also build a Docker image and run the Docker container using Command Central. See *Building an EntireX Docker Image* for more information.

Prerequisites

- Operating system Linux
- Docker installation 1.13.1 or compatible
- Software AG EntireX installation containing the package EntireX > Core Files

Building and Running the RPC Server for Java Image

The scripts provided with EntireX support the following three methods of building a Docker image and running the Docker container.

- [Configuring with Modified Dockerfile](#)
- [Configuring during Image Start, using Default File Names](#)
- [Configuring during Image Start, using Custom File Names](#)

Configuring with Modified Dockerfile

➤ **To copy license, configuration and customer service implementation files into Docker container**

- 1 Set your working directory to `<install_dir>/EntireX/docker/JavaRpcServer`.
- 2 Create the TAR file containing all the necessary files with the following command:

```
./CreateEntireXJavaRpcServerTar.sh
```

- 3 Provide your configuration files into the current working directory, for example:

- *myLicense*
- *myConfiguration*
- *myData/myClassFiles*



Note: All files are required if you are using this method.

- 4 Update the Dockerfile, for example:

```

# Possibility to add a valid license file already to the image instead of
# providing it during start up
# e.g.:
ADD myLicense /licenses/license.xml

# Possibility to add a different configuration file already to the image instead of
# providing it during start up
# e.g.:
ADD myConfiguration /configs/entirex.javarpserver.properties

# Possibility to add server implementation to the image
# e.g.:
ADD myData/. /data

```

5 Build the server image:

```
docker build -t exx_java_rpc_server_image_1 .
```

In this case the Docker `build` command copies the configuration into the image.

6 Start the container:

```
docker run -d -e ACCEPT_EULA=Y --name exx_java_rpc_server_container_1
exx_java_rpc_server_image_1
```

■ Advantages

The complete configuration is in the image. For troubleshooting, Software AG Support will require only the image and the command you entered.

■ Disadvantage

If the configuration changes, you will have to build a new image before you rerun the container.

Configuring during Image Start, using Default File Names

➤ **To copy license, configuration and customer service implementation files into container, using default file names**

(Customer service implementation classes are provided in directory mounted to `/data`.)

- 1 Set your working directory to `<install_dir>/EntireX/docker/JavaRpcServer`.
- 2 Create the TAR file containing all the necessary files with the following command:

```
./CreateEntireXJavaRpcServerTar.sh
```

3 Build the server image:

```
docker build -t exx_java_rpc_server_image_2 .
```

4 Provide your license, configuration and server implementation files with the default file names, for example:

- *<my-license-dir>/license.xml*
- *<my-config-dir>/entirex.javarpserver.properties*
- *<my-data-dir>/<custom classes>*



Note: All files are required if you are using this method.

In this case the license and configuration files are mounted during startup.

5 Start the container:

```
docker run -d
    -e ACCEPT_EULA=Y
    -v <my-license-dir>:/licenses
    -v <my-config-dir>:/configs
    -v <my-data-dir>:/data
    --name exx_java_rpc_server_container_2 exx_java_rpc_server_image_2
```

■ **Advantages**

If the configuration changes, you do not need to rebuild the image; you only need to rerun the container.

■ **Disadvantage**

The configuration, license and data files are mounted to the container. For troubleshooting, Software AG Support will require the image, configuration, license, data files and the command you entered.

Configuring during Image Start, using Custom File Names

➤ **To copy license, configuration and customer service implementation files into container, using custom file names**

(Customer service implementation JAR file are provided in directory mounted to */data*.)

- 1 Set your working directory to *<install_dir>/EntireX/docker/JavaRpcServer*.
- 2 Create the TAR file containing all the necessary files with the following command:

```
./CreateEntireXJavaRpcServerTar.sh
```

3 Build the server image:

```
docker build -t exx_java_rpc_server_image_3 .
```

4 Provide your configuration files with your own file names, for example:

- *<my-license-dir>/myLicense*
- *<my-config-dir>/my.entirex.javarpserver.properties*
- *<my-data-dir>/<custom jar file>*



Note: All files are required if you are using this method.

In this case the license and configuration files are mounted during startup.

5 Start the container:

```
docker run -d
    -e ACCEPT_EULA=Y
    -e "EXX_LICENSE_KEY=myLicense.xml"
    -e ↵
    "EXX_JAVA_SERVER_CONFIGURATION=my.entirex.javarpserver.properties"
    -e "EXX_JAVA_SERVER_CLASSPATH=/data/custom.jar"
    -v <my-license-dir>:/licenses
    -v <my-configuration-dir>:/configs
    -v <my-data-dir>:/data
    --name exx_java_rpc_server_container_3 exx_java_rpc_server_image_3
```

■ Advantages

If the configuration changes, you do not need to rebuild the image; you only need to rerun the container. You can choose your own file names.

■ Disadvantage

The configuration, license and data files are mounted to the container. For troubleshooting, Software AG Support will require the image, configuration, license, data files and the command you entered.

Verifying the Build

> **To verify the build**

- 1 Show the image with command

```
docker images
```

- 2 Start the docker image to be verified as described above, for example:

```
docker run -d -e ACCEPT_EULA=Y ↵  
--name exx_java_rpc_server_container_1 exx_java_rpc_server_image_1
```

- 3 Show the log:

```
docker logs -f exx_java_rpc_server_container_1
```

- 4 Show the containers:

```
docker ps
```

- 5 Stop the container:

```
docker stop exx_java_rpc_server_container_1
```

- 6 Delete the container:

```
docker rm exx_java_rpc_server_container_1
```

- 7 Remove the image:

```
docker rmi exx_java_rpc_server_image_1
```

Healthcheck for RPC Server for Java

The *docker* directory for RPC Server for Java contains a script `healthcheck.sh`. Execution of this script pings the RPC server and returns the result of the ping command:

- 0 success
- 1 ping failure

In the Docker context, this `healthcheck.sh` is put into the Docker container and enabled by setting the `HEALTHCHECK` instruction in the Dockerfile.

You can also use the `healthcheck.sh` script in the context of an orchestration tool (e.g. Kubernetes) to enable healthcheck functionality.

