

## **webMethods EntireX**

### **EntireX RPC Server for .NET**

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-DOTNETRPC-105-20220422**

## Table of Contents

EntireX RPC Server for .NET .....	v
1 About this Documentation .....	1
Document Conventions .....	2
Online Information and Support .....	2
Data Protection .....	3
2 Introduction to the RPC Server for .NET .....	5
Administration using Command Central .....	6
.NET Wrapper Runtime and .NET Server Assembly .....	7
Worker Models .....	8
3 Administering the RPC Server for .NET 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 .NET using the Command Central Command Line .....	29
Creating an RPC Server Instance .....	30
Configuring an RPC Server Instance .....	31
Displaying the EntireX Inventory .....	48
Viewing the Runtime Status .....	50
Starting an RPC Server Instance .....	51
Stopping an RPC Server Instance .....	51
Inspecting the Log Files .....	52
Changing the Trace Level Temporarily .....	54
Deleting an RPC Server Instance .....	55
5 Administering the RPC Server for .NET with Local Scripts .....	57
Customizing the RPC Server .....	58
Configuring the RPC Server .....	60
Locating and Calling the Target Server .....	67
Using SSL/TLS with the RPC Server .....	67
Starting the RPC Server .....	69
Stopping the RPC Server .....	70
Pinging the RPC Server .....	71
Deploying the RPC Server .....	71
Running an EntireX RPC Server as a Windows Service .....	72
Activating Tracing for the RPC Server .....	72
6 Scenarios .....	73
Writing a New .NET Server Assembly .....	74



---

## EntireX RPC Server for .NET

---

The EntireX RPC Server for .NET allows standard RPC clients to communicate with .NET server assemblies. It works together with the .NET Wrapper.

The supported .NET development and runtime environments are described under *Windows Prerequisites* in the Release Notes.

---

# 1 About this Documentation

---

▪ Document Conventions .....	2
▪ Online Information and Support .....	2
▪ Data Protection .....	3

## Document Conventions

---

Convention	Description
<b>Bold</b>	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 .NET

---

- Administration using Command Central ..... 6
- .NET Wrapper Runtime and .NET Server Assembly ..... 7
- Worker Models ..... 8

The EntireX RPC Server for .NET allows standard RPC clients to communicate with .NET server assemblies. It works together with the .NET Wrapper.

## 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 displays the 'Instances' page in the Software AG Command Central interface. The top navigation bar includes 'SOFTWARE AG Command Central' and several menu items: 'Installations', 'Stacks', 'Licensing', 'Repositories', 'Jobs', and 'Administrator'. Below the navigation bar, there is a breadcrumb trail 'Home > Instances > ALL' and a search box for environments. The main content area features a 'Search Instances' box and a table of installed instances. The table has columns for Name, Component, Status, Alerts, Installation, and Host. The data rows are as follows:

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:

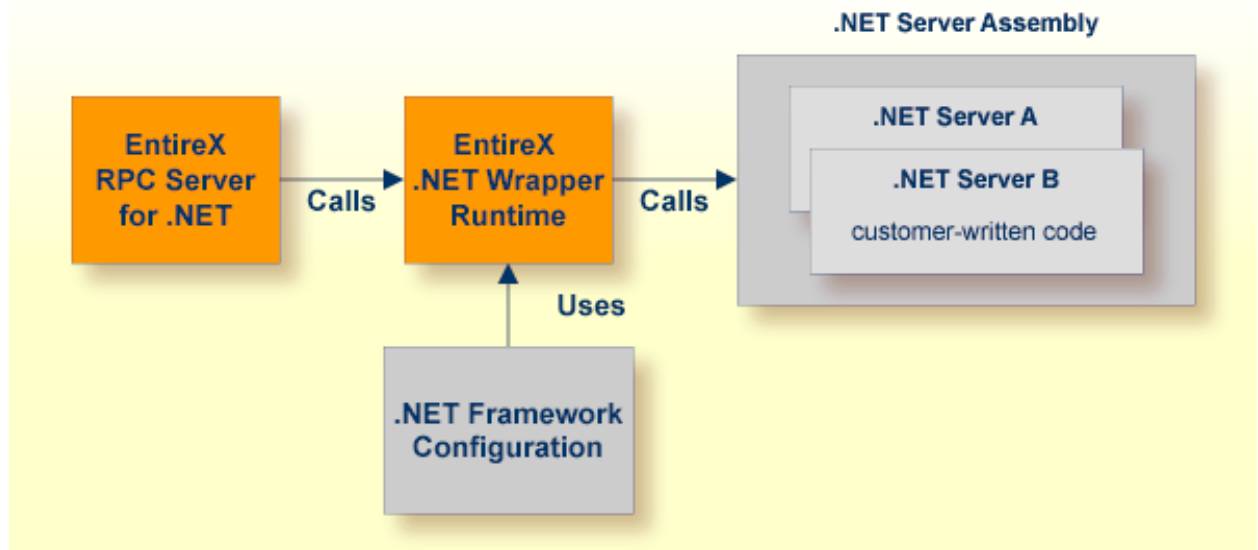
- Infrastructure engineers can see at a glance which products and fixes are installed, where they are installed, and compare installations to find discrepancies.
- System administrators can configure environments by using a single web user interface or command-line tool. Maintenance involves minimum 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 .NET using the Command Central GUI](#). For the command-line interface, see [Administering the RPC Server for .NET 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.

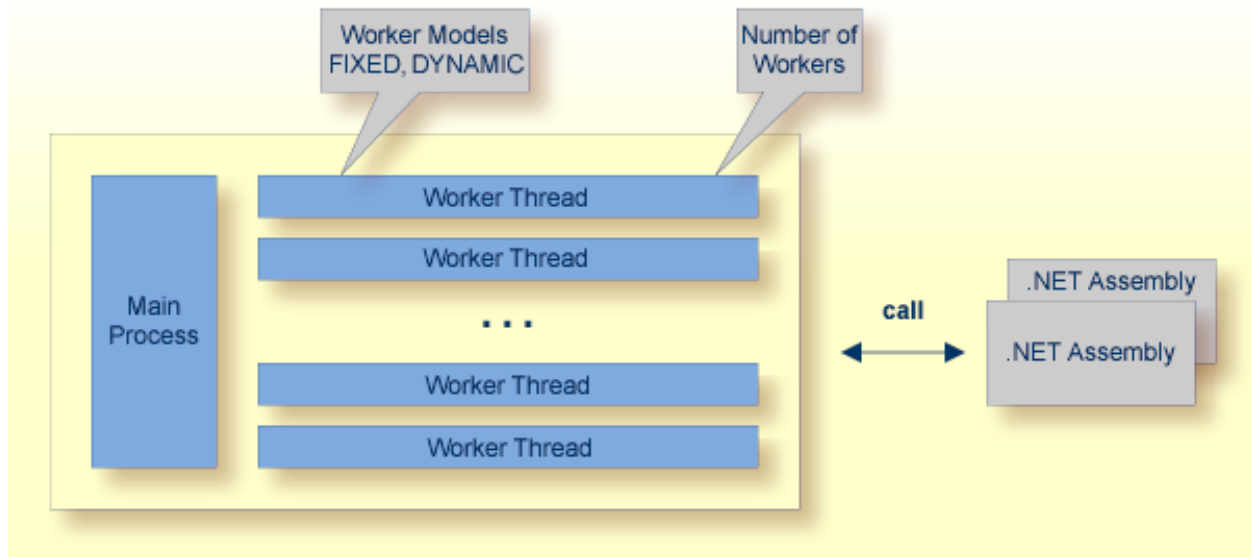
## .NET Wrapper Runtime and .NET Server Assembly

The RPC Server for .NET uses the .NET Wrapper Runtime to call the .NET Server assemblies. .NET Server assembly skeletons are generated with the .NET Wrapper.



For more information on the RPC Server for .NET, .NET Wrapper Runtime and .NET Framework Configuration, see [Customizing the RPC Server](#).

## Worker Models



RPC requests are worked off inside the RPC server in worker threads, which are controlled by a main thread. 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 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 parameter [endworkers](#) under *Administering the RPC Server for .NET with Local Scripts*.

# 3 Administering the RPC Server for .NET 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 .NET, using the Command Central graphical user interface.

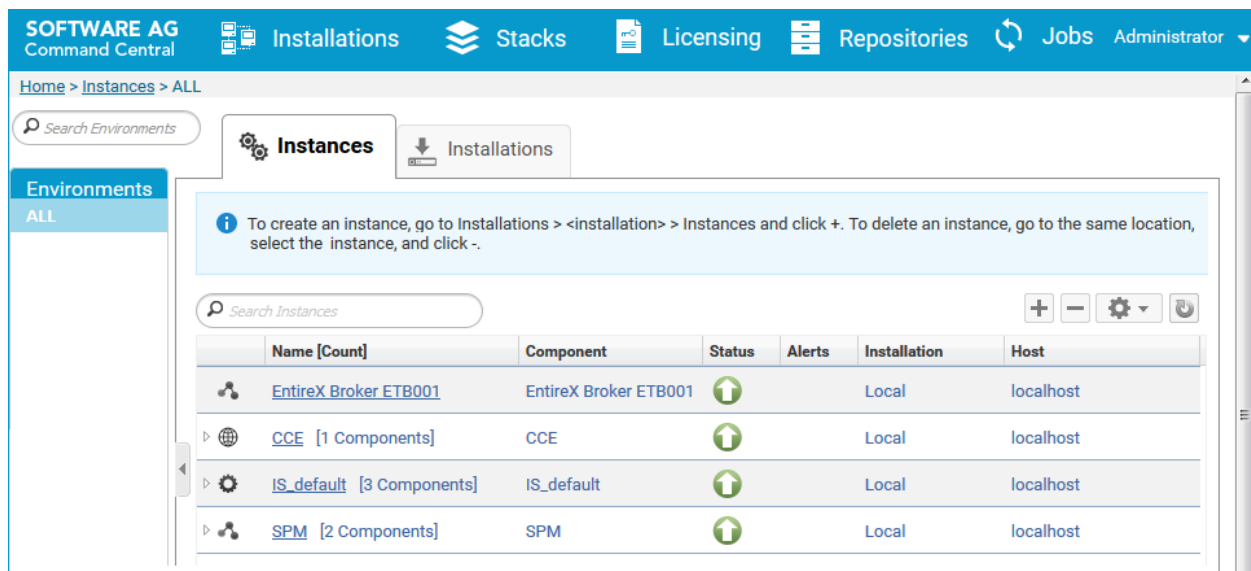
See also *Administering the RPC Server for .NET using the Command Central Command Line*. The core Command Central documentation is provided separately and is also available under **Guides for Tools Shared by Software AG Products** on the Software AG documentation website.

## 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**:

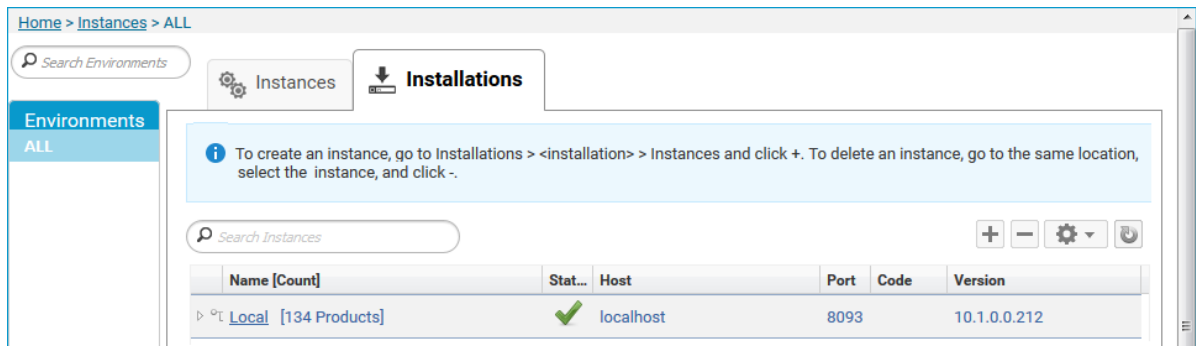




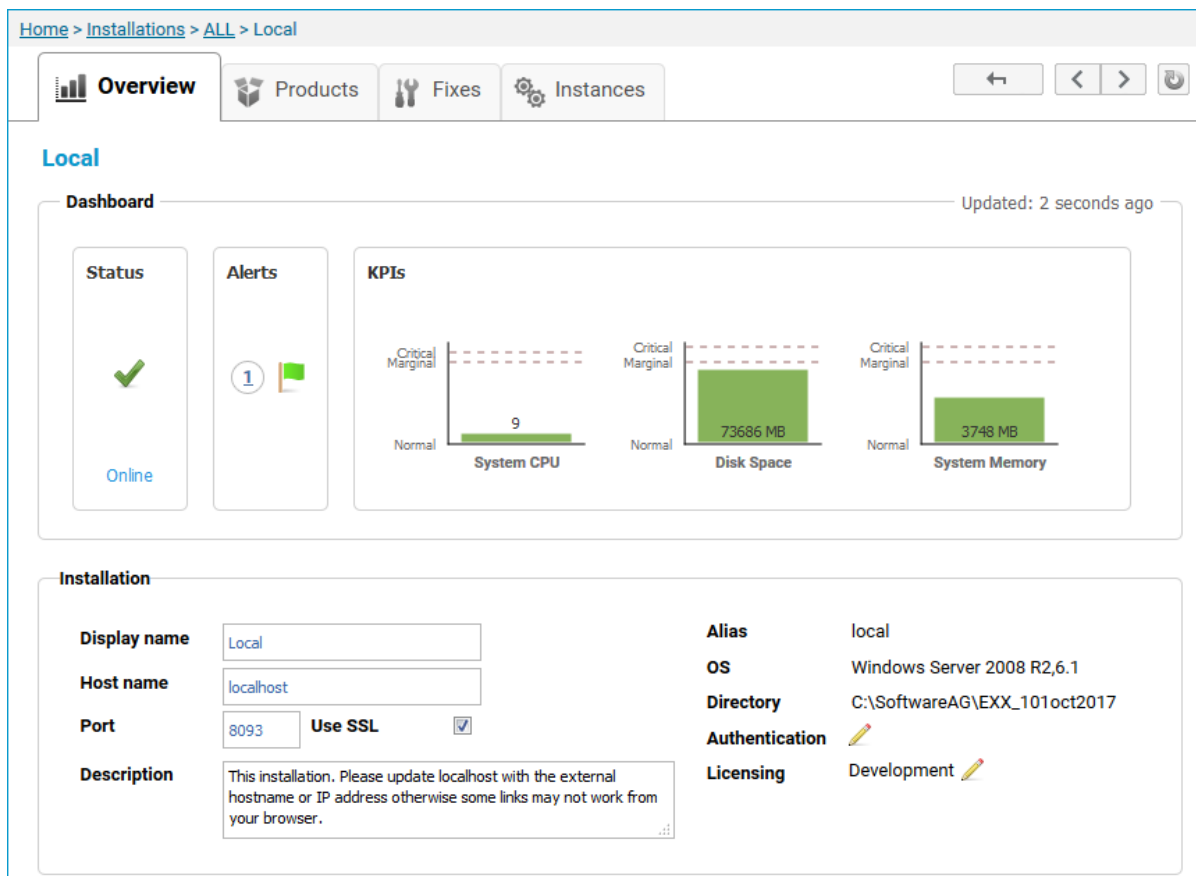
## Creating an RPC Server Instance

➤ To create an RPC Server for .NET instance

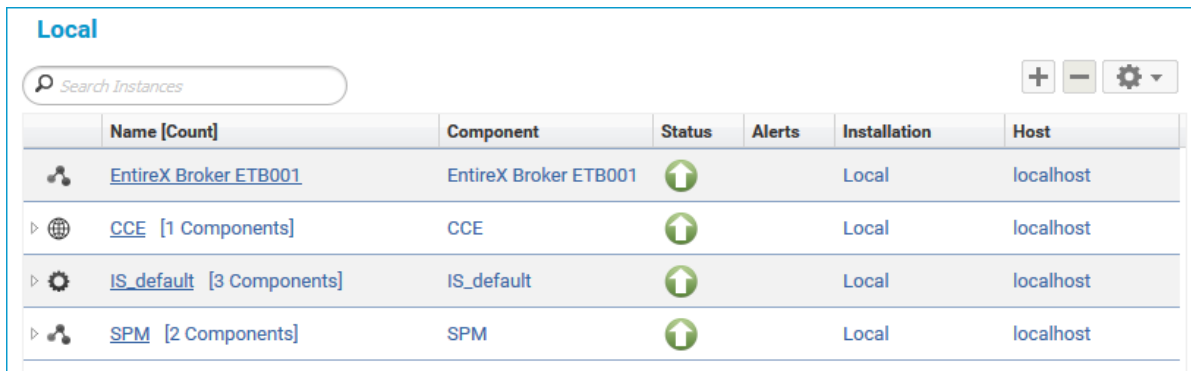
- 1 In the Command Central home page, click the **Installations** tab.



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

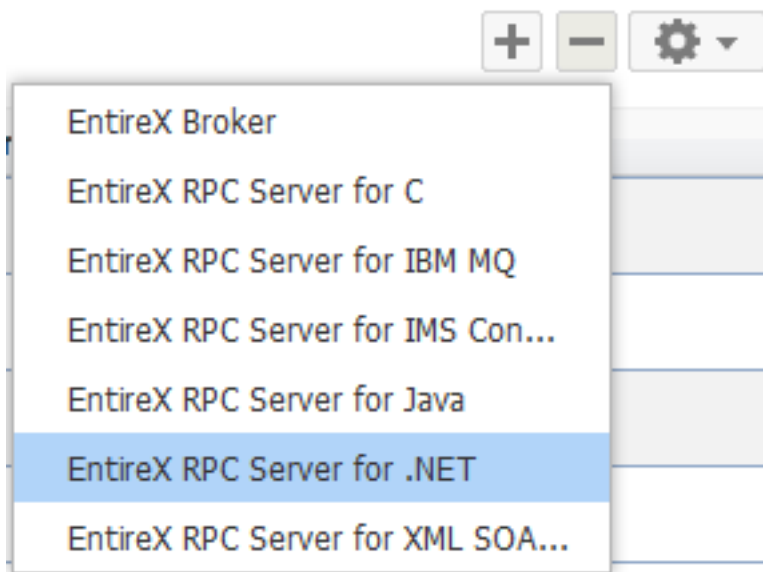


3 Click the **Instances** tab.

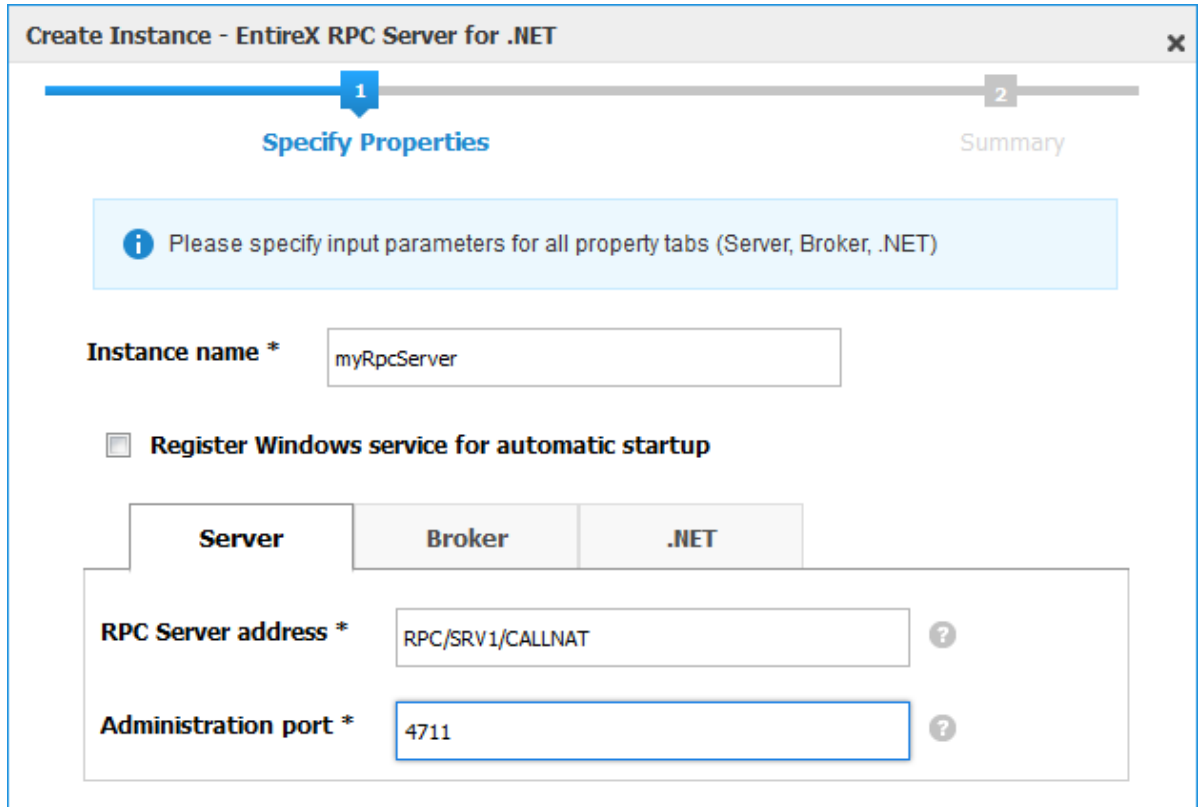


	Name [Count]	Component	Status	Alerts	Installation	Host
	<a href="#">EntireX Broker ETB001</a>	EntireX Broker ETB001	↑		Local	localhost
▶	<a href="#">CCE</a> [1 Components]	CCE	↑		Local	localhost
▶	<a href="#">IS_default</a> [3 Components]	IS_default	↑		Local	localhost
▶	<a href="#">SPM</a> [2 Components]	SPM	↑		Local	localhost

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



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

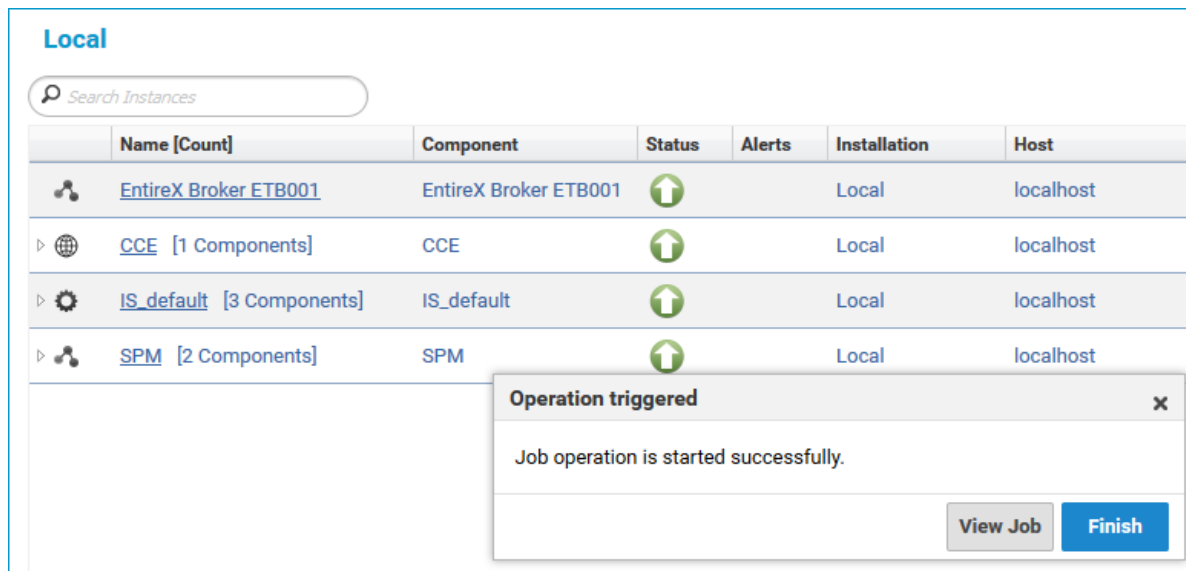
**Broker Tab**

Parameter	Description
<b>Connection</b>	
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.
<b>Credentials</b>	
User	Optional. The user ID for secured access to the broker.
Password	Optional. The password for secured access to the broker.

**.NET Tab**

Parameter	Description
IDL Library	Optional. IDL library name associated with .NET server assembly.
Assembly	Optional. Location (path including DLL name) of .NET server assembly.

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

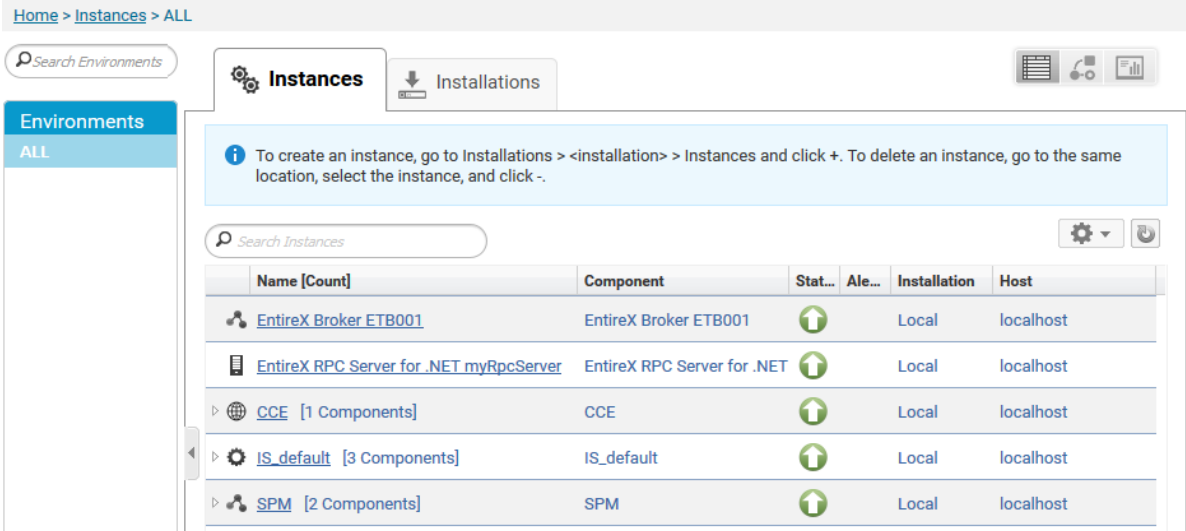


The new instance *myRpcServer* appears in the list.

## Configuring an RPC Server Instance

### > To configure an RPC Server for .NET instance

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



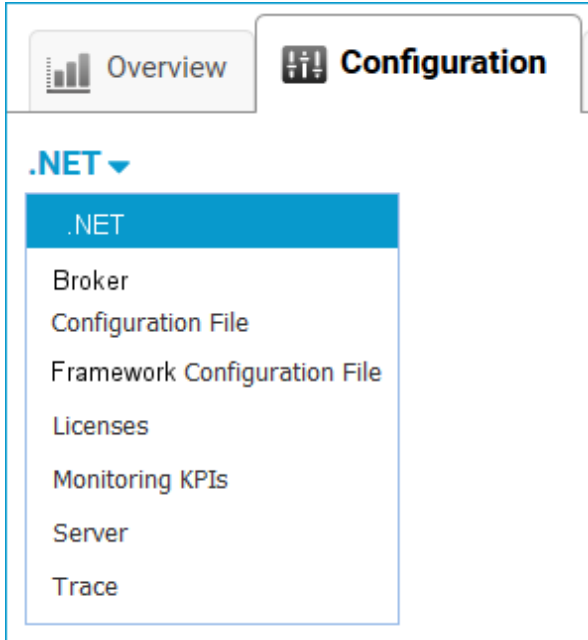
The screenshot shows the Command Central GUI with the 'Instances' tab selected. The breadcrumb navigation is 'Home > Instances > ALL'. A search bar for environments is visible. The main content area contains a table of instances with the following data:


Name [Count]	Component	Stat...	Ale...	Installation	Host
<a href="#">EntireX Broker ETB001</a>	EntireX Broker ETB001	↑		Local	localhost
<a href="#">EntireX RPC Server for .NET myRpcServer</a>	EntireX RPC Server for .NET	↑		Local	localhost
▸ <a href="#">CCE</a> [1 Components]	CCE	↑		Local	localhost
▸ <a href="#">IS_default</a> [3 Components]	IS_default	↑		Local	localhost
▸ <a href="#">SPM</a> [2 Components]	SPM	↑		Local	localhost

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

The screenshot displays the Command Central GUI interface for the 'EntireX RPC Server for .NET myRpcServer' instance. At the top, there are navigation tabs: Overview (selected), Configuration, Logs, and Administration. Below the tabs, the instance name is shown. The main area is divided into two sections: 'Dashboard' and 'Details'. The 'Dashboard' section, updated 30 seconds ago, contains three panels: 'Status' (Online), 'Alerts' (1), and 'KPIs'. The 'KPIs' section shows two bar charts: 'Active Workers' with a value of 1 and 'Busy Workers' with a value of 0. The 'Details' section lists various configuration parameters: Display name (EntireX RPC Server for .NET myRpcSe), Component (EntireX RPC Server for .NET my...), Host name (localhost), Authentication (with a pencil icon), Installation name (Local), and Installation alias (local). An 'Attributes' table is also present, currently empty.

- 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.

- **.NET**
- **Broker**
- **Configuration File**
- **Framework Configuration File**
- **Licenses**
- **Monitoring KPIs**
- **Server**
- **Trace Level**

#### **.NET**

Parameter	Description
IDL Library	Optional. IDL library name associated with .NET server assembly.
Assembly	Optional. Location (path including DLL name) of .NET server assembly.

## Broker

Parameter	Description
<b>Connection</b>	
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.
<b>Credentials</b>	
User	Optional. The user ID for secured access to the broker.
Password	Optional. The password for secured access to the broker.

## Configuration File

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

## Framework Configuration File

Here you can view/edit the framework configuration file of the RPC Server for .NET. See [.NET Framework Configuration](#).

## 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 .NET: Active Workers and Busy Workers.

Key performance indicators (KPIs) enable you to monitor the health of your RPC Server for .NET. 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!

### Server

Here you can specify the RPC Server settings.

Parameter	Description
<b>RPC Server</b>	
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 .NET stops.
<b>Worker Scalability</b>	
Worker model	You can either have a fixed or dynamic number of workers. Default is <i>dynamic</i> ( <i>true</i> ). For more information see <a href="#">Worker Models</a> .
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 .NET.

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 .NET instance for which you want to see the runtime status (same as Step 1 under *Configuring a Broker Instance*).



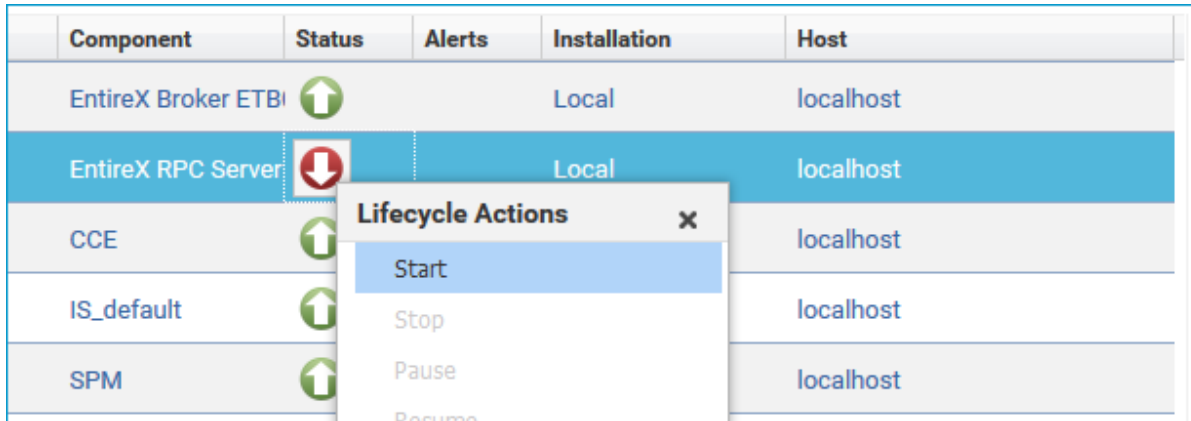
The visual key performance indicators (KPIs) and alerts enable you to monitor the RPC Server for .NET's health.

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 .NET instance from the Instances tab

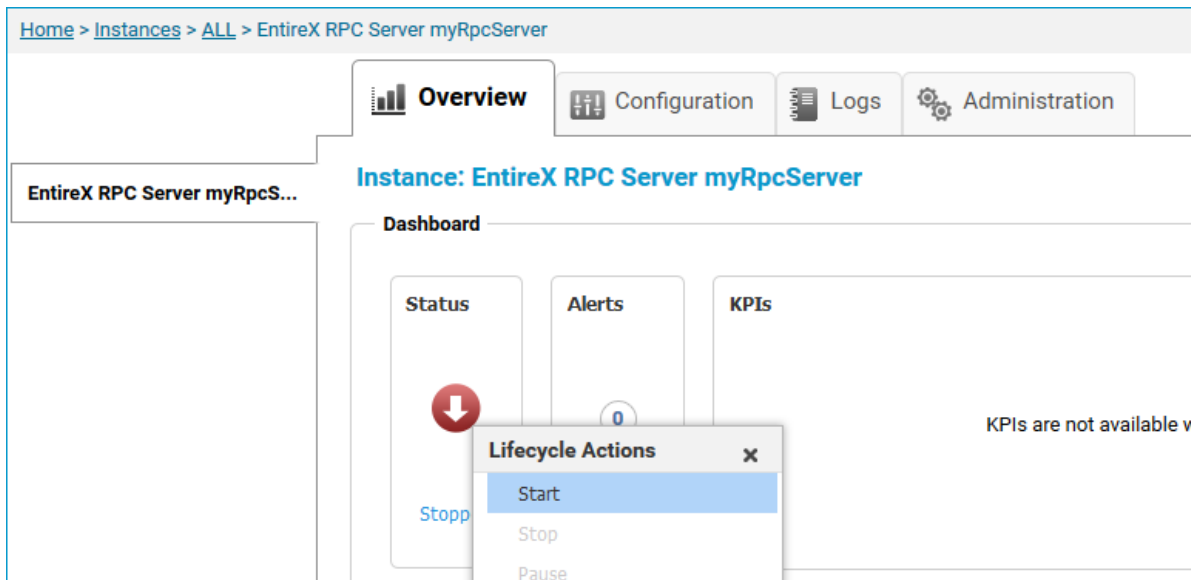
- 1 In the Command Central home page, click the **Instances** tab.



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

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

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

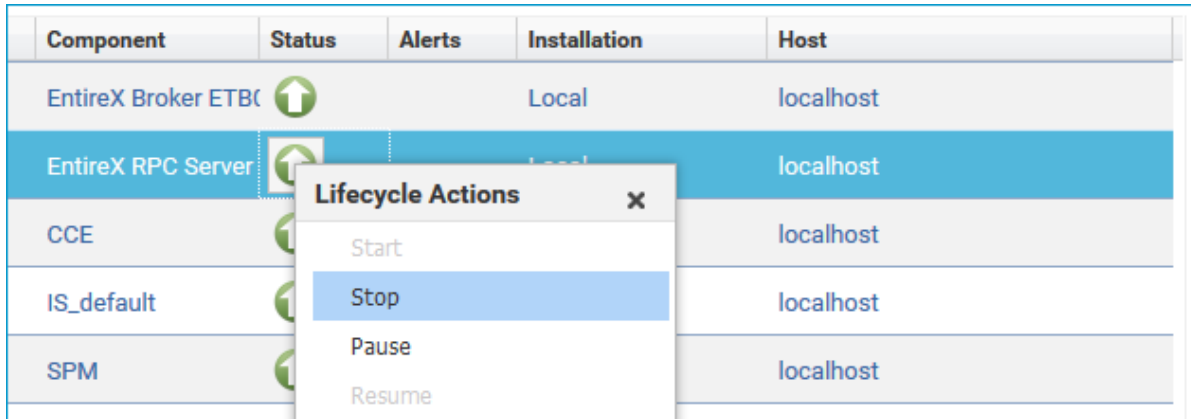


- 2 Select the status, and from the context menu choose **Start**.

## Stopping an RPC Server Instance

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

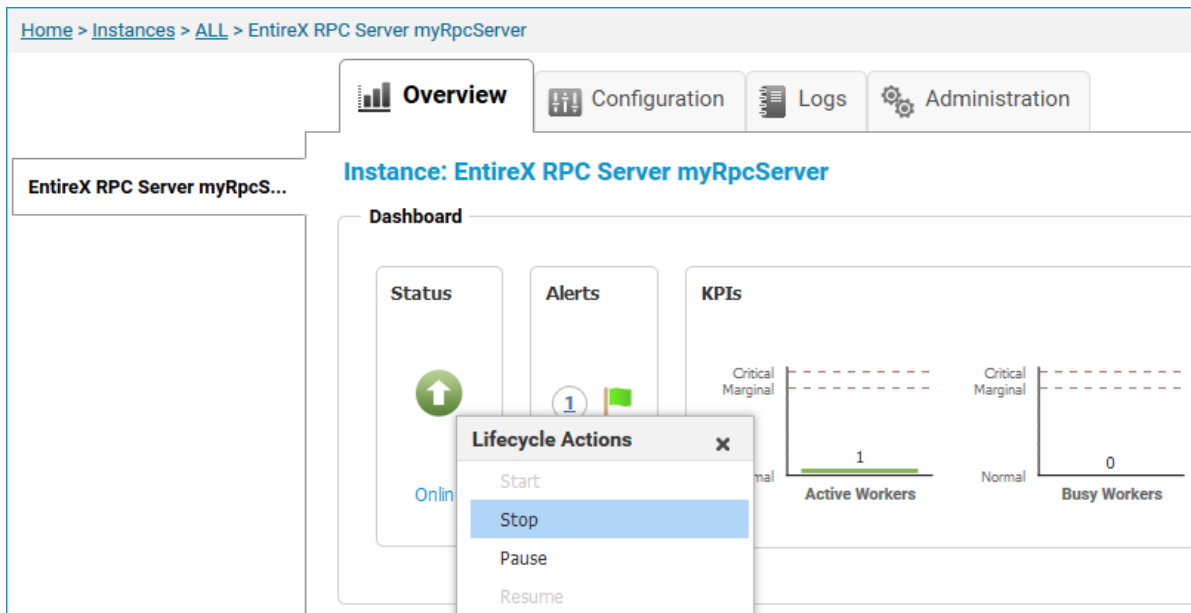
- 1 In the Command Central home page, click the **Instances** tab.



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

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

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

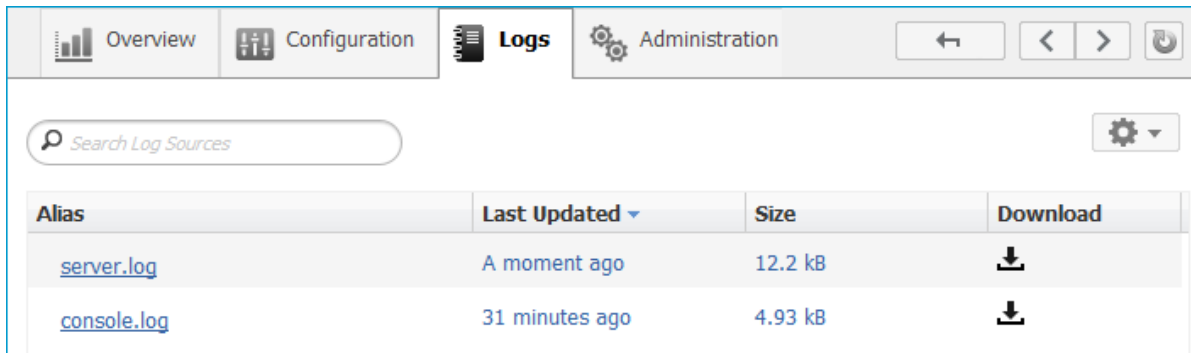


- 2 Select the status, and from the context menu choose **Stop**.

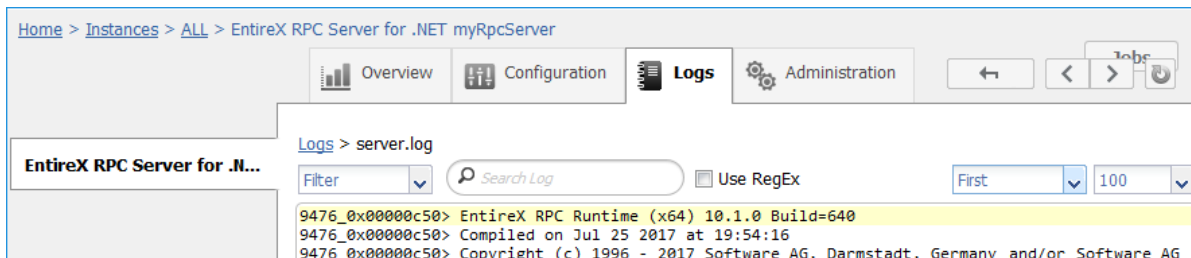
## Inspecting the Log Files

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

- 1 In the Command Central home page, click the **Instances** tab, then click the link associated with the RPC Server for .NET 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*:

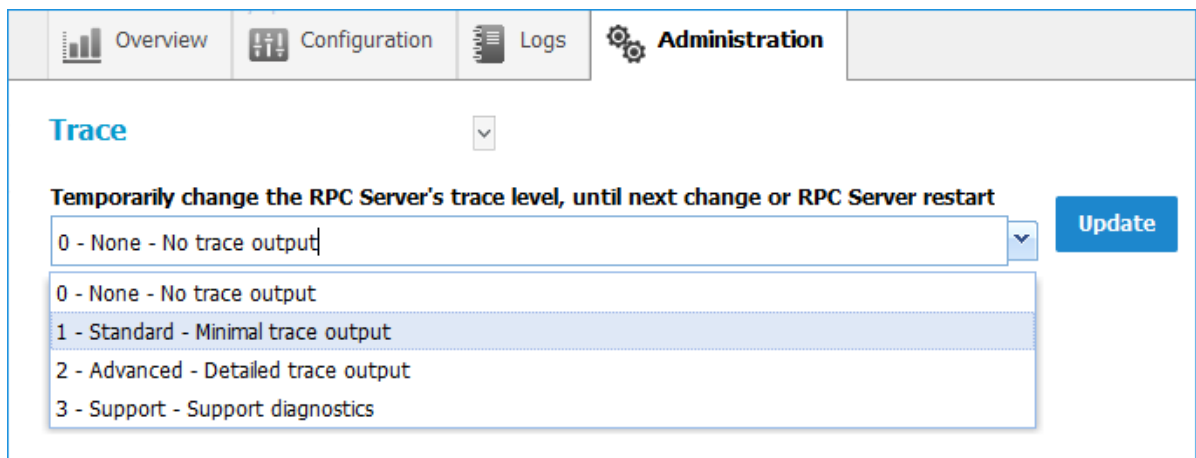




## Changing the Trace Level Temporarily

### > To temporarily change the trace level of an RPC Server for .NET instance


- 1 In the Command Central home page, click the **Instances** tab then click the link associated with the RPC Server for .NET 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**.

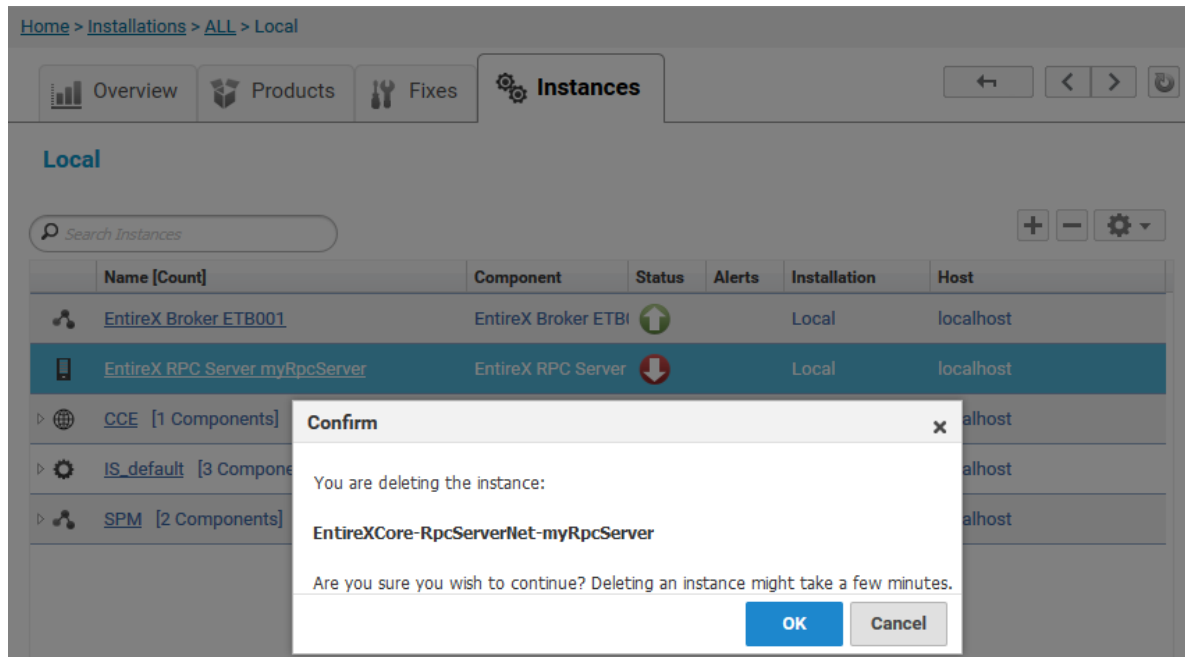


**Note:** If you want to set the trace level permanently, see [Trace Level](#) under *Configuring an RPC Server Instance*.

## Deleting an RPC Server Instance

### > To delete an RPC Server for .NET instance

- 1 In the list of EntireX RPC Server for .NET 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 .NET instance.
- 3 In the next window, click **Finish**. The selected instance is removed from the list.

# 4 Administering the RPC Server for .NET using the Command

## Central Command Line

---

▪ Creating an RPC Server Instance .....	30
▪ Configuring an RPC Server Instance .....	31
▪ Displaying the EntireX Inventory .....	48
▪ Viewing the Runtime Status .....	50
▪ Starting an RPC Server Instance .....	51
▪ Stopping an RPC Server Instance .....	51
▪ Inspecting the Log Files .....	52
▪ Changing the Trace Level Temporarily .....	54
▪ Deleting an RPC Server Instance .....	55

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

Administering the RPC Server for .NET using the Command Central GUI is described under [Administering the RPC Server for .NET 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	<code>node_alias</code>	<i>name</i>	Required. Specifies the alias name of the installation in which the runtime component is installed.
	<code>type</code>	RpcServerNet	Required. EntireXCore instance type of RPC server. Must be "RpcServerNet".
	<code>product</code>	EntireXCore	Required. Must be set to "EntireXCore".
	<code>instance.name</code>	<i>name</i>	Required. Name of the runtime component, for example "MyRpcServer".
	<code>install.service</code>	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.
	<code>server.address</code>	<i>class/server/service</i>	Required. The case-sensitive RPC server address has the format: CLASS/SERVER/SERVICE.
	<code>server.adminport</code>	1025-65535	Required. The administration port in range from 1025 to 65535.
	<code>broker.transport</code>	ssl   <u>tcp</u>	Transport over TCP or SSL. Default is TCP.
	<code>broker.host</code>	<i>name</i>	Required. EntireX Broker host name or IP address.
	<code>broker.port</code>	1025-65535	Required. Port number in range from 1025 to 65535.
	<code>broker.user</code>	<i>user</i>	Optional. The user ID for secured access to the broker.
	<code>broker.password</code>	<i>password</i>	Optional. The password for secured access to the broker.

Command	Parameter	Value	Description
	broker.truststore	<i>name</i>	Optional. Specifies the location of SSL trust store.
	idllibrary	<i>name</i>	Optional. IDL library name associated with .NET server assembly.
	assembly	<i>path/dllname</i>	Optional. Location (path including DLL name) of .NET server assembly.

### Example

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

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

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 .NET. Any changes to parameters will be used the next time you start the RPC server.

- [Broker](#)
- [Configuration File](#)
- [Framework Configuration File](#)
- [.NET](#)
- [Monitoring KPIs](#)
- [Server](#)

- [Trace Level](#)

## Broker

Here you can administer the parameters used for communication between the RPC Server for .NET 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-RpcServerNet-".
	<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 .NET "MyRpcServer" in the installation with alias name "local":

```
sagcc get configuration data local EntireXCore-RpcServerNet-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-RpcServerNet-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-RpcServerNet-".
	<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 .NET "MyRpcServer" in the installation with alias name "local" from the file *broker.json* in the current working directory:

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

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



## Configuration File

Here you can administer the configuration file of the RPC Server for .NET. 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-RpcServerNet-".
	<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 .NET "MyRpcServer" in the installation with alias name "local":

```
sagcc get configuration data local EntireXCore-RpcServerNet-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-RpcServerNet-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-RpcServerNet-".
	<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-RpcServerNet-MyRpcServer
CONFIGURATION -i configuration.json
```

## Framework Configuration File

Here you can overwrite the framework configuration file of the RPC Server for .NET.

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

### Displaying the Content of the RPC Server Framework 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-RpcServerNet-".
	<i>instanceid</i>	Required. Must be "FRAMEWORK".
	<i>-o file</i>	Optional. Specifies the file where you want the output written.

#### Example 1

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

```
sagcc get configuration data local EntireXCore-RpcServerNet-MyRpcServer FRAMEWORK
```

#### Example 2

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

```
sagcc get configuration data local EntireXCore-RpcServerNet-MyRpcServer FRAMEWORK
-o framework.cfg.txt
```

### Updating the Content of the RPC Server Framework 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-RpcServerNet-".
	<i>instanceid</i>	Required. Must be "FRAMEWORK".
	<i>-i file</i>	Optional. Specifies the file from where you want the input read.

### Example

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

```
sagcc update configuration data local EntireXCore-RpcServerNet-MyRpcServer
FRAMEWORK -i framework.cfg.txt
```

### .NET

Here you can modify how the RPC Server for .NET handles assemblies. An assembly is configured for each IDL library (see `library-definition` under *Software AG IDL Grammar* in the IDL Editor documentation).

- [Parameters](#)
- [Displaying the Assemblies](#)
- [Updating the Assemblies](#)

#### Parameters

Parameter	Description
AssemblyList	Enclosing parameter for list of (IdlLibrary, Assembly) parameter pairs. The parameter has no value.
IdlLibrary	IDL library associated with the assembly.
Assembly	Path of the assembly file (path including DLL name).

#### Displaying the Assemblies

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-RpcServerNet-".
	<i>instanceid</i>	Required. Must be "NET".
	<i>-o file</i>	Optional. Specifies the file where you want the output written.

#### Example 1

- To display the assembly parameters of RPC Server for .NET "MyRpcServer" in the installation with alias name "local":

```
sagcc get configuration data local EntireXCore-RpcServerNet-MyRpcServer NET
```

## Example 2

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

```
sagcc get configuration data local EntireXCore-RpcServerNet-MyRpcServer NET -o
assemblies.json
```

Resulting output file in JSON format:

```
{
  "AssemblyList":
  [
    {
      "IdLibrary": "Calculator",
      "Assembly": "c:\\assemblies\\calc\\calc.dll"
    },
    {
      "IdLibrary": "Hello",
      "Assembly": "C:\\assemblies\\hello\\Hello.dll"
    }
  ]
}
```

## Updating the Assemblies

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-RpcServerNet-".
	<i>instanceid</i>	Required. Must be "NET".
	<i>-i file</i>	Optional. Specifies the file from where you want the input read.

## Example

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

```
sagcc update configuration data local EntireXCore-RpcServerNet-MyRpcServer NET  
-i assemblies.json
```

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

## Monitoring KPIs

Here you can administer margins of monitored key performance indicators (KPIs) available for the RPC Server for .NET: 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 .NET. 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-RpcServerNet-".
	<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 .NET "MyRpcServer" in the installation with alias name "local" on stdout:

```
sagcc get configuration data local EntireXCore-RpcServerNet-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-RpcServerNet-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-RpcServerNet-".
	<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-RpcServerNet-MyRpcServer  
MONITORING-KPI -i my.properties
```

## Server

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

- [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 .NET 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 <a href="#">Worker Models</a> .
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-RpcServerNet-".
	<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 .NET "MyRpcServer" in the installation with alias name "local" on stdout:

```
sagcc get configuration data local EntireXCore-RpcServerNet-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-RpcServerNet-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-RpcServerNet-".
	<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-RpcServerNet-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 .NET.

- [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-RpcServerNet-".
	<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 .NET "MyRpcServer" in the installation with alias name "local" on stdout:

```
sagcc get configuration data local EntireXCore-RpcServerNet-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-RpcServerNet-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-RpcServerNet-".
	<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-RpcServerNet-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-RpcServerNet-".

### 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-RpcServerNet-".

### Example

- To display state information about the RPC Server for .NET:

```
sagcc get monitoring state local EntireXCore-RpcServerNet-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 .NET, 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-RpcServerNet-".

### Example

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

```
sagcc exec lifecycle start local EntireXCore-RpcServerNet-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 .NET, 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-RpcServerNet-".

### Example

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

```
sagcc exec lifecycle stop local EntireXCore-RpcServerNet-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 .NET. 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-RpcServerNet-".

### Example

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

```
sagcc list diagnostics logs local EntireXCore-RpcServerNet-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-RpcServerNet-".
	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-RpcServerNet-MyRpcServer server.log  
tail
```

### Example 2

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

```
sagcc get diagnostics logs local EntireXCore-RpcServerNet-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 .NET "MyRpcServer" in the installation with alias name "local" in JSON format on stdout:

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

#### Example 2

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

```
sagcc exec administration component local EntireXCore-RpcServerNet-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-RpcServerNet-".
	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-RpcServerNet-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-RpcServerNet-".

### Example

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

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

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

# 5

## Administering the RPC Server for .NET with Local Scripts

---

▪ Customizing the RPC Server .....	58
▪ Configuring the RPC Server .....	60
▪ Locating and Calling the Target Server .....	67
▪ Using SSL/TLS with the RPC Server .....	67
▪ Starting the RPC Server .....	69
▪ Stopping the RPC Server .....	70
▪ Pinging the RPC Server .....	71
▪ Deploying the RPC Server .....	71
▪ Running an EntireX RPC Server as a Windows Service .....	72
▪ Activating Tracing for the RPC Server .....	72

The EntireX RPC Server for .NET allows standard RPC clients to communicate with .NET server assemblies. It works together with the .NET Wrapper.

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

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

## Customizing the RPC Server

---

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

- [Configuration File](#)
- [RPC Server for .NET](#)
- [.NET Wrapper Runtime](#)
- [.NET Framework Configuration](#)
- [Start Script](#)

### Configuration File

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

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

For more information see [Configuring the RPC Server](#).

### RPC Server for .NET

Technically the RPC Server for .NET consists of the *rpcserver.exe* and *dotNetServer.dll* provided in the *bin* folder. The modules must always be kept together in the same folder.



## .NET Wrapper Runtime

The .NET Wrapper Runtime is required by the RPC Server for .NET. It is implemented in the assembly *SoftwareAG.EntireX.NETWrapper.Runtime.dll* provided in the *bin* folder. It contains, for example, marshalling code for .NET data types to Software AG IDL data types and unmarshalling code for the opposite direction.

## .NET Framework Configuration

For complex installations and if the .NET Server assemblies are deployed in different folders<sup>(1)</sup>, a .NET Framework Configuration file called *rpcserver.exe.config* is required. It defines in XML format several parameters of the RPC Server for .NET, such as the dependent assemblies, version and location and others. The file *rpcserver.exe.config* must be located in the same folder as the RPC Server for .NET itself, which by default is the *bin* folder of the EntireX installation. If there are multiple .NET servers assemblies, each deployed in different folders which need to be called by the RPC Server for .NET, they all must be defined in the *rpcserver.exe.config* file.

```
<configSections>
  <!-- EntireX Configuration Section Group Definition -->
  <sectionGroup name="EntireX">
    <section name="Assemblies" type="System.Configuration.NameValueSectionHandler, ←
System, Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089, ←
Custom=null" />
  </sectionGroup>
</configSections>
<EntireX>
  <!-- EntireX Assembly Configuration -->
  <Assemblies>
    <add key="SoftwareAG.EntireX.NETWrapper.Runtime" ←
value="C:\SoftwareAG\EntireX\bin\SoftwareAG.EntireX.NETWrapper.Runtime.dll" />
  </Assemblies>
</EntireX>
```

where the location of the *.NET Wrapper Runtime* is replaced by the location used in your EntireX installation. Add an entry in the Assemblies section for each of your called .NET Server assemblies:

```
<add key="MyAssembly", value="MyLocation"/>
```

where *MyAssembly* and *MyLocation* represent the name and location of your .NET server assembly. In this context, the .NET server assembly must have a strong name. Refer to the Microsoft documentation for the .NET Framework.

If versioning is required for your assemblies, follow the rules under *Assembly Versioning* in the .NET Wrapper documentation.



### Notes:

1. See also *Locating and Calling the Target Server* and *Deploying the RPC Server*.

2. Due to an incompatibility of the .NET Framework 2.0 to the earlier versions, an additional section in the `configSections` part of file `rpcserver.exe.config` is required.

### Start Script

The start script for the RPC Server for .NET is called `dotNetServer.bat` and is provided in the `bin` folder of the installation directory. You may customize this file. The start script contains the following:

- paths to the called .NET server
- the configuration file used; see [Configuration File](#)
- etc.

## Configuring the RPC Server

---

The following rules apply:

- In the configuration file:
  - Comments must be on a separate line.
  - Comment lines can begin with `'*`, `'/` and `';`.
  - Empty lines are ignored.
  - Headings in square brackets [`<topic>`] are ignored.
  - Keywords are not case-sensitive.
- Underscored letters in a parameter indicate the minimum number of letters that can be used for an abbreviated command.

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

Parameter	Default	Values	Req/Opt
<code>brokerid</code>	<code>localhost</code>	Broker ID used by the server. See <i>Using the Broker ID in Applications</i> .  Example: <code>brokerid=myhost.com:1971</code>	R
<code>callexit</code>	<code>dotNetServer</code>	Do not change!	R
<code>class</code>	RPC	Server class part of the server address used by the server. The server address must be defined as a service in the	R

Parameter	Default	Values	Req/Opt
		broker attribute file (see <i>Service-specific Attributes</i> ). Case-sensitive, up to 32 characters. Corresponds to CLASS.  Example: class=MyRPC	
codepage		The codepage tells the broker the encoding of the data. The application must ensure the encoding of the data matches the codepage. The RPC server itself does not convert your application data. The application's data is shipped and received as given. Often, the codepage must also match the encoding used in the RPC server environment for file and terminal IO, otherwise unpredictable results may occur.  Under the Windows operating system: <ul style="list-style-type: none"> <li>■ By default, the Windows ANSI codepage configured for your system is automatically transferred to tell the broker how the data is encoded.</li> <li>■ If you want to adapt the Windows ANSI codepage, see the Regional Settings in the Windows Control Panel and your Windows documentation.</li> <li>■ If you want to encode the data different to your Windows ANSI codepage, convert the data in the application and provide the codepage name here. During receive, decode the data accordingly.</li> </ul> codepage=windows-1256  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> .	O
compresslevel	N	Enforce compression when data is transferred between broker and server. See <i>Data Compression in EntireX Broker</i> .  compresslevel= 0   1   2   3   4   5   6   7   8   9   Y   N  0-9 0=no compression 9=max. compression N No compression. Y Compression level 6.	O

Parameter	Default	Values	Req/ Opt
		Example: compresslevel=6	
<u>endworkers</u>	timeout	<p>NEVER Defines worker model <b>FIXED</b> with a fixed number of worker threads. The number of worker threads is defined with <b>minworkers</b>. It does not increase or decrease during the lifetime of an RPC server instance.</p> <p>TIMEOUT Defines slow-shrinking worker model <b>DYNAMIC</b>, where the number of worker threads is adjusted to the current number of client requests. With value <b>TIMEOUT</b>, all worker threads not used are stopped in the time specified by the <b>timeout</b>, except for the minimum number of active workers specified with <b>minworkers</b>. The upper limit of workers parallel active is restricted with <b>maxworkers</b>.</p> <p>IMMEDIATE Defines fast-shrinking worker model <b>DYNAMIC</b>, where the number of worker threads is adjusted to the current number of client requests. With value <b>IMMEDIATE</b>, worker threads not used are stopped immediately as soon as they have finished their conversation, except for the minimum number of active workers defined with <b>minworkers</b>. The upper limit of workers active in parallel is restricted with <b>maxworkers</b>.</p> <p>Example: endworkers=timeout minworkers=2 maxworkers=6 timeout=60</p>	O
<u>library</u>	library =PREFIX(D) - PREFIX()	<p>library = <i>search_logic</i> [- <i>library</i>]</p> <p>where <i>search_logic</i> is one of <b>FIX(<i>dllname</i>)</b>   <b>PREFIX(<i>prefix</i>)</b>   <b>PREFIX()</b>, and <i>library</i> can be repeated up to four times, that is, five entries are possible.</p>	O

Parameter	Default	Values	Req/Opt
		<p><code>FIX(<i>dllname</i>)</code> The IDL library name coming from the RPC client is ignored. You have to define the library names (Windows DLLs).</p> <p><code>PREFIX(<i>prefix</i>)</code> The IDL library name coming from the RPC client is used to form the library name (Windows DLLs). As prefix you can define any character. If an RPC client sends, for example, "SYSTEM" as the IDL library name and "D" is defined as prefix, the library name derived is "DSYSTEM".</p> <p><code>PREFIX()</code> The IDL library name coming from the RPC client is used as library name (Windows DLLs).</p> <p>Example FIX configuration:  <code>library=FIX(MYSTUBS) - FIX(MYRPCS)</code></p>	
<code>logon</code>	YES	<p>Execute broker functions LOGON/LOGOFF in worker threads. Must match the setting of the broker attribute AUTOLOGON. Reliable RPC requires logon set to YES. See <i>Reliable RPC</i>.</p> <p>NO No logon/logoff functions are executed.</p> <p>YES Logon/logoff functions are executed.</p> <p>Example:  <code>logon=no</code></p>	O
<code>minworkers</code>	1	<p>Minimum limit of worker threads.</p> <ul style="list-style-type: none"> <li>■ For worker model <b>DYNAMIC</b>: minimum number of active worker threads, even if no RPC client requests have to be processed. This allows you to define a certain number of worker threads - not used by the currently executing RPC request - to wait for new RPC client requests to process. In this way the RPC server is ready to handle many RPC client requests arriving at the same time. Do not set a value higher than <code>maxworkers</code>.</li> <li>■ For worker model <b>FIXED</b>: number of active worker threads. Do not set a value higher than 256.</li> </ul> <p>See also <code>endworkers</code>.</p> <p>Example:</p>	O

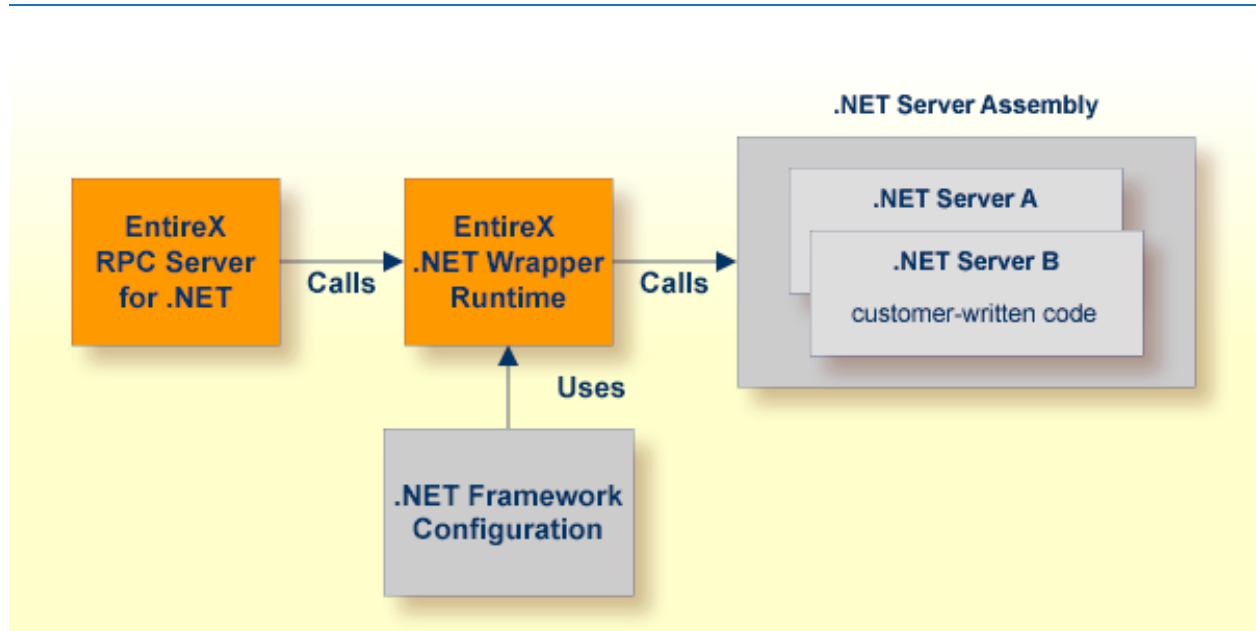
Parameter	Default	Values	Req/ Opt
		minworkers=2	
maxworkers	10	Upper limit of all tasks concurrently. Do not set a value higher than 256. See also <a href="#">endworkers</a> .  Example: maxworkers=2	O
password	no default	The password for secured access to the broker. If possible (write access) the password is encrypted and written to parameter password.e. The parameter password is removed. To change the password, add the parameter password with the new password as value.  Example: password=MyPwd	O
restartcycles	15	Number of restart attempts if the broker is not available. This can be used to keep the RPC Server for .NET running while the broker is down for a short time. A restart cycle will be repeated every 60 seconds.  <b>Note:</b> Internally, the server waits in periods of 10 seconds (performing six times more cycles), which you can see in the server output.  When the number of specified cycles is reached and a connection to the broker is not possible, the RPC Server for .NET stops.  Example: restartcycles=30  The server waits up to 30 minutes (30*6*10 seconds) before it terminates due to a missing broker connection.	O
runoption	Reset	Do not change! Do not add other run options	R
servername	SRV1	Server name part of the server address used by the server. The server address must be defined as a service in the broker attribute file. See <i>Service-specific Attributes</i> . Case-sensitive, up to 32 characters. Corresponds to SERVER of the broker attribute file.  Example: servername=mySrv	R
service	CALLNAT	Service part of the server address used by the server. The server address must be defined as a service in the broker attribute file. See <i>Service-specific Attributes</i> . Case-sensitive, up to 32 characters. Corresponds to SERVICE attribute of the broker attribute file.	R

Parameter	Default	Values	Req/Opt
		Example: service=MYSERVICE	
ssl_file	no default	Set the SSL parameters. See <a href="#">Using SSL/TLS with the RPC Server</a> for examples and more information.	O
timeout	60	Timeout in seconds, used by the server to wait for broker requests. See broker ACI control block field WAIT for more information. Also influences <a href="#">restartcycles</a> and worker model <a href="#">DYNAMIC</a> .  Example: timeout=300	O
tracedestination	ERXTrace.nnn.log	The name of the destination file for trace output. By default the main trace file name is ERXTrace.nnn.log, where <i>nnn</i> can be in the range from 001 to 005. See also <a href="#">Activating Tracing for the RPC Server</a> .  ■ Under Windows, the trace file is located in a subfolder of the windows folder <i>My Documents</i> .  If the default is not used and a tracedestination is specified, you can use the following variables depending on the operating system:  %...%                    Windows    Environment variable. @PID                    UNIX, Win    Process ID. @TID                    UNIX, Win    Thread ID. @RANGE[ <i>n, m</i> ]       UNIX, Win <i>m</i> must be greater than <i>n</i> , range is from 0 - 999 @CSIDL_PERSONAL       Windows    The user's home directory. The variable will be resolved by Windows shell functions. @CSIDL_APPDATA       Windows    The <i>Application Data Directory</i> . The variable will be resolved by Windows shell functions. @CSIDL_LOCAL_APPDATA    Windows    The <i>Local Application Data Directory</i> . The variable will be resolved by Windows shell functions.	O

Parameter	Default	Values	Req/ Opt
		See also <a href="#">Activating Tracing for the RPC Server</a> .  Example: tracedestination=ERXTrace.log	
<u>tracelevel</u>	None	Trace level for the server. See also <a href="#">Activating Tracing for the RPC Server</a> .  <pre>tracelevel = None   Standard   Advanced   ↔ Support</pre> <p>None No trace output. Standard For minimal trace output. Advanced For detailed trace output. Support This trace level is for support diagnostics and should only be switched on when requested by Software AG support.</p> <p>Example: tracelevel=standard</p>	O
<u>traceoption</u>	None	Additional trace option if trace is active. See also <a href="#">Activating Tracing for the RPC Server</a> .  <p>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.</p> <p><b>Note:</b> This can increase the amount of trace output data dramatically if you transfer large data buffers.</p> <p>Example: traceoption=(STUBLOG,NOTRUNC)</p>	O
<u>userid</u>	ERX - SRV	The user ID for access to the broker. The default ERX - SRV will be used if this parameter is omitted or specified without a value: "userid=".  Example: userid=MyUid	O



## Locating and Calling the Target Server



The approach depends on whether the called .NET Server assemblies are in the same folder as the *RPC Server for .NET* and *.NET Wrapper Runtime* or in a different folder:

- **Assemblies in same folder**

Having your .NET Server assemblies in the same folder as the RPC Server for .NET and .NET Wrapper Runtime makes sense for test and development purposes, and also for small applications. The .NET Server assemblies are loaded from the respective folder. No extra .NET Framework configuration is required. See also *Deploying the RPC Server*.

- **Assemblies in different folder**

If your .NET Server assemblies are not in the same folder as the RPC Server for .NET and .NET Wrapper Runtime, you need to configure your .NET Framework. See *.NET Framework Configuration*.

## Using SSL/TLS with the RPC Server

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

- *URL Style*, for example:

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

- *Transport-method Style*, for example:

```
ETB024:1609:SSL
```

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

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

- **As part of the Broker ID**

The simplest way to specify short SSL parameter is to add them to the Broker ID.

Example with URL-style Broker ID:

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

Example with transport-method-style Broker ID:

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

- **In the SSL file**

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

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

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

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

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

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

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

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

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

- 4 Make sure the SSL server to which the RPC Server for .NET connects is prepared for SSL connections as well. The SSL server can be EntireX Broker, Broker SSL Agent, or Direct RPC in webMethods Integration Server (IS inbound). See:
  - *Running Broker with SSL/TLS Transport* in the platform-specific Administration documentation
  - Broker SSL Agent in the UNIX and Windows Administration documentation
  - *Support for SSL/TLS* in the EntireX Adapter documentation (for Direct RPC)

## Starting the RPC Server

Before starting, make sure all your .NET server assemblies are accessible through the standard Windows DLL load mechanism. See also [Locating and Calling the Target Server](#).

### ➤ To start the RPC Server for .NET

- Use the [Start Script](#).

Or:

Use the following format:

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

Here are some sample options. See [Configuring the RPC Server](#) for full list.

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



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

Or:

You can use the RPC Server for .NET as a Windows Service. See [Running an EntireX RPC Server as a Windows Service](#).

## Stopping the RPC Server

---

### > To stop the RPC Server for .NET

- 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.

See also *Component Return Codes in EntireX*.

## Pinging the RPC Server

---

### > To ping the RPC Server for .NET

- 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.

## Deploying the RPC Server

---

The easiest way to deploy and run an RPC Server for .NET with its .NET server assemblies is the so-called XCOPY-deployment<sup>(1)</sup>. This means that all relevant files of the RPC Server for .NET are installed in a single folder. The only prerequisite is that the *EntireX Mini Runtime Considerations* is installed. The following files are typically required:

- **.NET Wrapper Runtime** *SoftwareAG.EntireX.NETWrapper.Runtime.dll*
- **RPC Server for .NET** *rpcserver.exe* and *dotNetServer.dll*
- **Configuration File** *dotNetServer.cfg*
- .NET Server assemblies (containing customer-written code), see *Writing a .NET Server Assembly* in the .NET Wrapper documentation



### Notes:

1. The XCOPY deployment method has the drawback that copies of the **.NET Wrapper Runtime** and the **RPC Server for .NET** have to be deployed with customer-written .NET Server assemblies. It is possible to avoid this by deploying the .NET Server assemblies in different folders. See also *Locating and Calling the Target Server*.

## 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 .NET as a Windows Service

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



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

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

See also *Starting the RPC Server*.

- 2 Test your RPC server to see whether it will start if you run your script file.
- 3 Use the *EntireX RPC Service Tool* and install the `RPCService` with some meaningful extension, for example `MyServer`. If your *Start Script* is `dotNetServer.bat`, the command will be

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

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

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

## Activating Tracing for the RPC Server

---

### ➤ To switch on tracing for the RPC Server for .NET

- 1 Set the parameters `tracelevel`, `traceoption` and `tracedestination`. See *Configuring the RPC Server*.
- 2 Start the RPC Server for .NET. See *Starting the RPC Server*.
- 3 To evaluate the return codes, see *Component Return Codes in EntireX*.

### ➤ To switch off tracing

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

# 6 Scenarios

---

- Writing a New .NET Server Assembly ..... 74

## Writing a New .NET Server Assembly

---

➤ **To write a new .NET server assembly**

- 1 Use the .NET Wrapper to generate a C# server skeleton. See *Writing a .NET Server Assembly*.
- 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