software AG

webMethods EntireX

EntireX Java ACI

Version 10.9

April 2023

WEBMETHODS

This document applies to webMethods EntireX Version 10.9 and all subsequent releases.

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

Document ID: EXX-ACI-109-20230403JAVA

Table of Contents

EntireX Java ACI v
1 About this Documentation 1
Document Conventions 2
Online Information and Support 2
Data Protection 3
2 Using EntireX Java ACI 5
3 Writing Applications - EntireX Java ACI 7
Introduction 8
Required Steps 8
4 Writing Advanced Applications - EntireX Java ACI 9
Using Compression 10
Using EntireX Security with Java-based EntireX Applications 11
Setting Transport Methods 12
Tracing 17
Using Internationalization with Java ACI 18

EntireX Java ACI

EntireX Java ACI is a Java class library that provides access to the EntireX Broker ACI for Java programmers. It covers the whole EntireX Broker ACI which enables you to write both client and server applications in Java. Any of these can then interact with each other and with other applications written in other languages on the same network using EntireX Broker. The EntireX Java ACI also contains the framework necessary for Java RPC requests.

Related Literature

- For a description of classes, see EntireX Java ACI (Javadoc).
- *Java Wrapper*
- For a description of error messages see *Message Class 0013 - EntireX Java*.
- Broker HTTP(S) Agent in the platform-specific Administration documentation.

1 About this Documentation

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

Document Conventions

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

Online Information and Support

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.softwareag.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 Using EntireX Java ACI

EntireX Java ACI is a Java class library (Java package) that provides access to the EntireX Broker ACI for Java programmers. You can visualize it as the Java "language binding" for the EntireX Broker ACI. In this way, it is related to, for example, the C and Natural language interfaces.

EntireX Java ACI comes as a single Java package: *com.softwareag.entirex.aci*. All of the EntireX Broker ACI functionality is wrapped in a system of collaborating classes. Thus, it should appeal equally well to both the experienced Java programmer and to those familiar with the EntireX Broker ACI. The package also contains the framework for RPCs.

The class library is packaged in the *entirex.jar* file which can be found in the *classes* subdirectory of the EntireX installation.

Since EntireX Java ACI is only a thin interface layer to communicate with EntireX Broker, little "local" knowledge is implemented in it. All essential information will be passed to and received from EntireX Broker itself.

Documentation of the classes consists of the documentation generated by Javadoc and the documentation of the core API shipped with the JDK. This provides quick recognition for those familiar with this format.

3 Writing Applications - EntireX Java ACI

- Introduction 8
- Required Steps 8

Introduction

Interaction with the API occurs through instantiating objects of different classes, invoking their methods and manipulating their inner state. Not all features are necessary for all applications, depending on whether you are writing a client or a server application. The following is a general list of basic steps you have to perform.

Required Steps

- Instantiate a `Broker` object. This is the central object you will work with. One object instance represents one session to an EntireX Broker on your network. If you want to work with multiple EntireX Brokers or with multiple sessions, create one object for each session to an EntireX Broker. Using the same object instance in multiple sessions may result in slow response times and performance issues.
- Use the `Broker` object to log on the application to EntireX Broker.
- Instantiate a `BrokerService` object. If you are writing a server application, use the `BrokerService` object to register your service with the EntireX Broker.
- Declare a `BrokerMessage` variable. If you want to send a message, instantiate a new `BrokerMessage` object, complete it with your message and send it using one of the send methods. Messages received from the Broker are received in a newly created `BrokerMessage` object.
- Non-conversational communication is handled by the `BrokerService` and `BrokerMessage` objects. Use the `send`, `sendReceive` and `receive` methods of `BrokerService` for synchronous and asynchronous non-conversational communication. When writing a server, you can use the `reply` method of `BrokerMessage`.
- Conversational communication is handled by the `Conversation` and `BrokerMessage` objects.
- Unit-of-work communication is handled by the `UnitofWork` and `BrokerMessage` objects.
- Perform all your business logic processing on the message contents.
- When finished, end your conversations, deregister your service (if you are writing a server) and log off from EntireX Broker.

4 Writing Advanced Applications - EntireX Java ACI

- Using Compression 10
- Using EntireX Security with Java-based EntireX Applications 11
- Setting Transport Methods 12
- Tracing 17
- Using Internationalization with Java ACI 18

Using Compression

Java-based EntireX applications (including applications using classes generated by the Java Wrapper) may compress the messages sent to and received from the Broker. There are two ways to enable compression:

- Use the method `setCompressionLevel()` of the Broker object.
- Use a Broker ID with the parameter `compresslevel=<value>`.

Using setCompressionLevel()

Add the compression level to the method `setCompressionLevel()` as an integer or a string argument.

You can use the constants defined in class `java.util.zip.Deflater`.

If the string

- starts with Y, compression is turned on with level 6,
- starts with N, compression is turned off (level 0).

Permitted values are the integers 0 - 9 and the corresponding strings:

`BEST_COMPRESSION`	level 9
`BEST_SPEED`	level 1
`DEFAULT_COMPRESSION`	level 6
`DEFLATED`	level 8
`NO_COMPRESSION`	level 0

Using Broker ID

You may append the keyword `COMPRESSLEVEL` with one of the values above to the Broker ID.

Examples

- `localhost:1971?compresslevel=BEST_COMPRESSION`
- `localhost?poolsize=4&compresslevel=9`

Both examples set the compression level to 9.

Using EntireX Security with Java-based EntireX Applications

Java-based EntireX applications that require security can use EntireX Security. Use one of the supported methods of `useEntireXSecurity()` within class `Broker`. See also *Introduction to EntireX Security* for additional prerequisites. The auto mode specifies that the broker object uses EntireX Security as needed by the broker kernel. If the broker kernel is set up with security, the Broker object uses EntireX Security. If the broker kernel is not set up with EntireX Security, it is not used.

Notes:

1. For encrypted transport we strongly recommend using the Secure Sockets Layer/Transport Layer Security protocol. See *SSL/TLS, HTTP(S), and Certificates with EntireX* in the platform-independent Administration documentation.
2. Existing user-written security exits created for earlier versions of EntireX will continue to be supported.

Setting Transport Methods

- Socket Pooling Parameters for TCP and SSL/TLS Communication
- Using the Broker ACI with SSL/TLS
- Using HTTP(S) Tunneling
- Setting the Transport Timeout

Socket Pooling Parameters for TCP and SSL/TLS Communication

Socket connections for *EntireX Java ACI* applications and applications using classes generated by the Java Wrapper are assigned dynamically to instances of Broker objects. They are closed automatically when they are not used for a certain period of time. The behavior of the socket pooling can be controlled by two parameters (`poolsize` and `pooltimeout`) specified as part of the Broker ID. They are used for both TCP and SSL communications.

You can

- specify the maximum number of socket connections which are kept in the socket pool
- disable socket pooling
- control the automatic closing of socket connections

To specify the maximum number of socket connections

- Specify the parameter `poolsize` as part of the Broker ID.

  If the number entered is reached, further Broker calls going through a Broker instance will be delayed until a socket becomes available. If a multithreaded application uses blocking sendReceive or Receive calls with a longer waiting time, the `poolsize` parameter must be at least equal to the number of threads. The value of `entirex.timeout` (in seconds) is used to terminate the wait time for free sockets. If all sockets in the pool are in use, the calls will be delayed at the most by the period of time specified by this timeout. Afterwards, the call returns with error code 0013 0333. This is to prevent applications from hanging up if all sockets are in use and never become available due to network problems.

  The default for `poolsize` is 32. The default can be changed with a Java system property. Set the property `entirex.socket.poolsize` to specify a different value. Values that are not numeric or less than 1 are ignored.

To disable socket pooling

- Set the parameter `poolsize` (as part of the Broker ID) to "0".

To control the automatic closing of socket connections

- Specify the parameter `pooltimeout` (as part of the Broker ID).

  If a socket connection has not been used for the specified number of seconds, it will be closed automatically.

  The default for `pooltimeout` is 300 seconds.

  Example of a maximum number of 10 socket connections and a timeout of 60 seconds:

```
Broker broker = new Broker("yourbroker?poolsize=10&pooltimeout=60","userID");
```

Using the Broker ACI with SSL/TLS

ACI applications 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. ACI-based clients or servers are always SSL clients. The SSL server can be either the EntireX Broker or the Broker SSL Agent. For an introduction see *SSL/TLS, HTTP(S), 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 sample 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 Broker ID and SSL parameters.

   SSL transport will be chosen if the Broker ID starts with the string `ssl://`. Example of a typical *URL-style Broker ID*:

```
Broker broker = new ↩
Broker("ssl://yourbroker:10000?trust_store=castore&trust_passwd=trustpwd","userID");
```

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

   If the SSL client checks the validity of the SSL server only, this is known as *one-way SSL*. Two SSL parameters must be specified on the SSL client side: `trust_store` and `trust_passwd`. The mandatory `trust_store` parameter specifies the file name of a PKCS#12 certificate store that must contain the certificate chain of the trusted certificate authority (CA) that issued the SSL server's certificate.
   To unlock this certificate store, the password has to be set with SSL parameter `trust_passwd`.

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*.

Example of one-way SSL:

```
Broker broker = new ↩
Broker("ssl://yourbroker:10000?trust_store=castore&trust_passwd=trustpwd&verify_server=no","userID");
```

Example of two-way SSL:

```
Broker broker = new ↩
Broker("ssl://yourbroker:10000?trust_store=castore&trust_passwd=trustpwd&key_store=keystore&key_passwd=pwd","userID");
```

3 Make sure the SSL server to which the ACI application (client or server) 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 platform-specific Administration documentation

Using HTTP(S) Tunneling

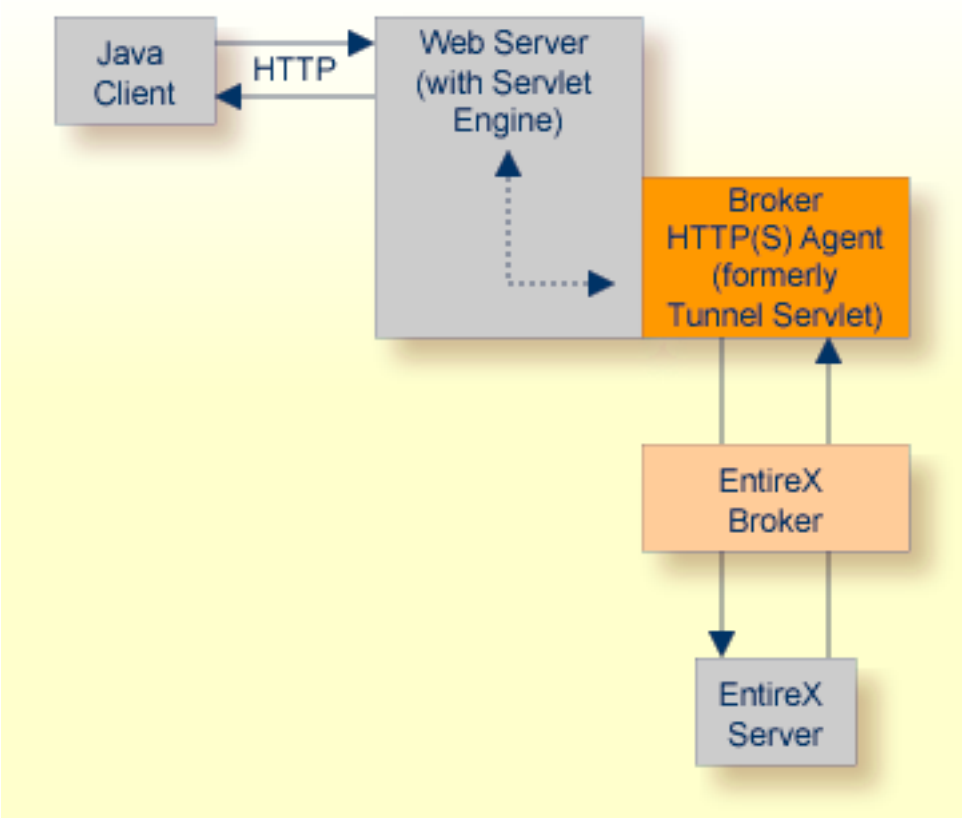
When communicating with EntireX Broker over the internet, direct access to the EntireX Broker's TCP/IP port is necessary. This access is often restricted by proxy servers or firewalls. Java-based EntireX applications can pass communication data via HTTP or HTTPS. This means that a running EntireX Broker in the intranet is made accessible by a Web server without having to open additional TCP/IP ports on your firewall (HTTP tunneling). This section covers the following topics:

- How the Communication Works
- Enabling HTTP Support

- Enabling HTTPS Support

How the Communication Works

The Broker HTTP(S) Agent builds the bridge between Web server and EntireX Broker in the intranet.



The figure above shows how the communication works. In this scenario, a Java client program communicates via HTTP and EntireX Broker with an EntireX server. By using a Broker ID starting with "http://" (passing the URL of the installed Broker HTTP(S) Agent) each Broker request is sent to a Web server, which immediately processes the Broker HTTP(S) Agent, passes the contents to EntireX Broker, receives the answer and sends it back via HTTP. For the two partners (client and server) it is transparent that they are communicating through the Web. Java server programs can also communicate via HTTP if necessary.

For the configuration, see Broker HTTP(S) Agent in the platform-specific Administration documentation.

Enabling HTTP Support

To enable HTTP support

- Pass the URL of your Broker HTTP(S) Agent installation as Broker ID to your Broker objects.

  For Example:

```
import com.softwareag.entirex.aci.Broker;

...
//  "http://www.yourhost.com/servlets/tunnel" is the URL to reach your broker ↩
over HTTP

Broker broker = new Broker("http://www.yourhost.com/servlets/tunnel","userID");
...

// other code not affected
...
```

The Broker HTTP(S) Agent optionally accepts parameters as part of the URL. It is possible to define values for Broker and log that override the corresponding values in the configuration of the Broker HTTP(S) Agent.

To enforce logging of the Broker HTTP(S) Agent

- Type, e.g. the following:

```
Broker broker = new ↩
Broker("http://www.yourhost.com/servlets/tunnel?log=yes","userID");
```

Enabling HTTPS Support

To use HTTPS instead of HTTP

- Replace "http://" by "https://" at the beginning of the Broker ID.

  Using HTTPS requires a Web server with SSL support enabled. Check your Web server's documentation for information on how to configure SSL support.

  Many Java implementations do not support HTTPS. If this is the case, your application will receive a BrokerException with error code 00130325.

Setting the Transport Timeout

Java-based EntireX applications (including applications using classes generated by the Java Wrapper) can set a transport timeout to abort socket connections when not receiving any reply.

To specify a TCP or SSL transport timeout

1 Use the system property `entirex.timeout`.

The default is 20 seconds.

A numeric value of 1 or greater indicates the transport timeout in seconds.

Setting the value to 0 results in a potentially infinite wait (that is, until the Broker returns a reply or the socket connection is closed).

If the Broker call is a send call with wait or a receive call, the transport timeout is added to the Broker wait time specified as part of the Broker call.

The value of `entirex.timeout` is used as a timeout for waiting for free sockets in the socket pools. If the application does not get a free socket during this timeout period, an exception will be thrown.

2 Use the static method `Broker.setTransportTimeout(int timeout)` in your application.

This method sets the socket timeout value in seconds. It is used for TCP/IP, but not with HTTP. The timeout value is used for new sockets, it does not change the timeout for sockets in use.

To query the current setting, use the method `Broker.getTransportTimeout()`.

Tracing

Java-based EntireX applications (including applications using classes generated by the Java Wrapper) can use tracing to log program flow and locate problems.

To specify the trace level

- Use the `setTrace()` method of class `Broker`.

  Or:

  Use the Java system property `entirex.trace`. The system property uses the same values as the `setTrace` method call.

  0 No tracing, default.

1 Trace all broker calls and other major actions.

2 Dump the send and receive buffer.

3 Dump the buffers sent to the broker and received from the broker.

Using Internationalization with Java ACI

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 Linux implementations, it can be changed with the `LANG` environment variable.

With the `setCharacterEncoding(enc)` method of the `BrokerService` (EntireX Java ACI) you can override the encoding used for the payload sent to / received from the broker without affecting the default encoding of your JVM.

Methods of the Java ACI are inherited by the Java Wrapper when programming RPC clients. Thus, regarding character conversion, most of what is valid for the Java ACI is also valid for Java-based RPC clients, but EntireX Broker configuration is different: use the service-specific attribute `CONVERSION` to enable correct character conversion in the broker.

- For Java-based RPC clients communicating with RPC-based components and Reliable RPC, set `CONVERSION=SAGTRPC`.
- For Java ACI clients and servers using ACI-based programming, set `CONVERSION=SAGTCHA`.

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*.