webMethods EntireX

Java ACI JavaDoc

Version 8.2 SP2

Month 2011

webMethods EntireX

This document applies to webMethods EntireX Version 8.2 SP2 and to all subsequent releases.

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

Table of Contents

How This API Document Is Organized

This API (Application Programming Interface) document has pages corresponding to the items in the navigation bar, described as follows.

Overview

The Overview page is the front page of this API document and provides a list of all packages with a summary for each. This page can also contain an overall description of the set of packages.

Package

Each package has a page that contains a list of its classes and interfaces, with a summary for each. This page can contain four categories:

- Interfaces (italic)
- Classes
- Enums
- Exceptions
- Errors
- Annotation Types

Class/Interface

Each class, interface, nested class and nested interface has its own separate page. Each of these pages has three sections consisting of a class/interface description, summary tables, and detailed member descriptions:

- Class inheritance diagram
- Direct Subclasses
- All Known Subinterfaces
- All Known Implementing Classes
- Class/interface declaration
- Class/interface description

- Nested Class Summary
- Field Summary
- Constructor Summary
- Method Summary

- Field Detail
- Constructor Detail
- Method Detail

Each summary entry contains the first sentence from the detailed description for that item. The

summary entries are alphabetical, while the detailed descriptions are in the order they appear in the source code. This preserves the logical groupings established by the programmer.

Annotation Type

Each annotation type has its own separate page with the following sections:

- Annotation Type declaration
- Annotation Type description
- Required Element Summary
- Optional Element Summary
- Element Detail

Enum

Each enum has its own separate page with the following sections:

- Enum declaration
- Enum description
- Enum Constant Summary
- Enum Constant Detail

Tree (Class Hierarchy)

There is a Class Hierarchy page for all packages, plus a hierarchy for each package. Each hierarchy page contains a list of classes and a list of interfaces. The classes are organized by inheritance structure starting with `java.lang.Object`. The interfaces do not inherit from `java.lang.Object`.

- When viewing the Overview page, clicking on "Tree" displays the hierarchy for all packages.
- When viewing a particular package, class or interface page, clicking "Tree" displays the hierarchy for only that package.

Deprecated API

The Deprecated API page lists all of the API that have been deprecated. A deprecated API is not recommended for use, generally due to improvements, and a replacement API is usually given. Deprecated APIs may be removed in future implementations.

Index

The Index contains an alphabetic list of all classes, interfaces, constructors, methods, and fields.

Prev/Next

These links take you to the next or previous class, interface, package, or related page.

Frames/No Frames

These links show and hide the HTML frames. All pages are available with or without frames.

Serialized Form

Each serializable or externalizable class has a description of its serialization fields and methods. This information is of interest to re-implementors, not to developers using the API. While there is no link in the navigation bar, you can get to this information by going to any serialized class and clicking "Serialized Form" in the "See also" section of the class description.

Constant Field Values

The Constant Field Values page lists the static final fields and their values.

This help file applies to API documentation generated using the standard doclet.

Overview Package Class **Tree** **Deprecated** **Index** **Help**

PREV NEXT **FRAMES** **NO FRAMES** **All Classes**

EntireX Java ACI

Packages	
com.softwareag.entirex.aci	This package is the Java ACI for the EntireX Broker.
com.softwareag.entirex.jms	This package is the Java Message Service implementation with the EntireX Broker.
com.softwareag.entirex.xml.rt	

Overview Package Class **Tree** **Deprecated** **Index** **Help**
PREV NEXT **FRAMES** **NO FRAMES** **All Classes**

A

B

C

D

E

F

G

I

J

K

L

M

O

P

Q

R

S

T

U

X

A B C D E F G I J K L M O P Q R S T U X

Overview Package Class **Tree** **Deprecated** **Index** **Help**

PREV NEXT **FRAMES** **NO FRAMES** **All Classes**

com.softwareag.entirex.aci

Class Broker

```
java.lang.Object
  └ com.softwareag.entirex.aci.Broker
```

```
public final class Brokerextends java.lang.Object
```

Represents a session instance of an EntireX Broker and handles the connection to an EntireX Broker. This version supports TCP/IP and SSL connections to EntireX Broker. HTTP and HTTPS connections to EntireX Broker via the EntireX TunnelServlet are also supported.
Connection-related information include:

- the Broker address ("BrokerID") (for a definition see `getBrokerID()`);
- the user ID;
- the token (optional for conversational communication, mandatory for UnitofWork communication).

User ID and token can be changed after instantiating an object of this class with the reconnect() method. This class also handles security for user authentication and encryption of messages.
Instances of this class can be used safely in multi-threaded applications. However, all Broker calls going through the same instance of a Broker object will be serialized. Parallel Broker calls are supported when using multiple Broker objects.

Field Summary

static int	**ENCRYPTION_LEVEL_BROKER** Specifies that the message data for send and receive calls will be encrypted.
static int	**ENCRYPTION_LEVEL_NONE** Specifies that the message data will not be encrypted.
static int	**ENCRYPTION_LEVEL_TARGET** Specifies that the message data for send and receive calls will be encrypted.

Constructor Summary

Broker(java.lang.String brokerID, java.lang.String userID) Creates a Broker object for the specified Broker address and user ID.
Broker(java.lang.String brokerID, java.lang.String userID, java.lang.String token) Creates a Broker object for the specified Broker address, user ID and token.

Method Summary

void	**autoLogon**(java.lang.String password) Provides a password to this instance of the Broker object to be used in subsequent calls.
void	**disconnect**() Disconnects this Broker instance from the EntireX Broker.
java.lang.String	**getApplicationName**() Gets the name of the application.
java.lang.String	**getBrokerID**() Returns the Broker ID which was specified in the constructor of this class.
int	**getCompressionLevel**() Returns the compression level.
java.lang.String	**getConnInfo**() Returns connection information.
byte[]	**getIAFToken**() Get the IAF Security Token.
java.lang.String	**getPartnerBrokerId**() Gets the partner Broker id.
byte[]	**getSecurityToken**() Returns the current value of the security token.
static ThreadRunner	**getThreadRunner**() Gets the ThreadRunner object.
java.lang.String	**getToken**() Returns the token specified in the constructor of this class.
static int	**getTrace**() Returns the current trace level.
static int	**getTransportTimeout**() Gets the socket timeout value in seconds.
java.lang.String	**getUniqueID**() Returns a String object, which is unique for this instance of the Broker object.

java.lang.String	**getUserID**() Returns the current value of the user ID.
static java.lang.String	**getVersion**() Returns version information of the EntireX Java ACI package.
boolean	**isCommandLogging**() Gets the state of the command logging.
java.lang.String	**kernelversion**() Retrieves the version of the EntireX Broker.
void	**logoff**() Logs off the application from EntireX Broker.
void	**logon**() Logs the application on to EntireX Broker, either as client or server.
void	**logon**(java.lang.String password) Logs the application on to EntireX Broker with a password, either as client or server.
void	**logon**(java.lang.String password, java.lang.String newpassword) Logs the application on to EntireX Broker with a password and new password, either as client or server.
void	**reconnect**(java.lang.String userID) Reconnects this Broker object to the Broker address specified in the constructor and set the user ID.
void	**reconnect**(java.lang.String userID, java.lang.String token) Reconnects this Broker object to the Broker address specified in the constructor and set user ID and token.
void	**setApplicationName**(java.lang.String name) Sets the name of the application.
void	**setCommandLogging**(boolean value) Switch command logging on and off.
void	**setCompressionLevel**(int level) Sets the compression level.
void	**setCompressionLevel**(java.lang.String level) Sets the compression level.
void	**setIAFToken**(byte[] iafToken) Sets the IAF Security Token.
void	**setSecurity**(BrokerSecurity securityObject, boolean doEncrypt) Specifies a security object, enables BrokerSecurity and permits encryption/decryption of message data.

void	**setSecurity**(BrokerSecurity securityObject, int encryptionLevel) Specifies a security object, enables BrokerSecurity and permits encryption/decryption of message data.
void	**setSecurity**(BrokerSecurity securityObject, int encryptionLevel, boolean autoMode) Specifies a security object, enables BrokerSecurity and permits encryption/decryption of message data.
void	**setSecurityToken**(byte[] s) Sets the security token to the argument.
static void	**setThreadRunner**(ThreadRunner t) Sets the ThreadRunner object, which wraps the method to start a new thread.
static void	**setTrace**(int level) Sets the trace level.
static void	**setTrace**(int level, java.io.PrintWriter pw) Sets the trace level.
static void	**setTransportTimeout**(int timeout) Sets the socket timeout value in seconds.
java.lang.String	**toString**() Overrides the toString() method of the class Object.
void	**useEntireXSecurity**() Enables EntireX Security for this Broker instance.
void	**useEntireXSecurity**(boolean autoMode) Enables EntireX Security for this Broker instance.
void	**useEntireXSecurity**(int encryptionLevel) Enables EntireX Security for this Broker instance.
void	**useEntireXSecurity**(int encryptionLevel, boolean autoMode) Enables EntireX Security for this Broker instance.

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Field Detail

ENCRYPTION_LEVEL_NONE

```
public static final int ENCRYPTION_LEVEL_NONE
```

Specifies that the message data will not be encrypted.

Since:
EntireX 6.1.1
See Also:
Constant Field Values

ENCRYPTION_LEVEL_BROKER

```
public static final int ENCRYPTION_LEVEL_BROKER
```

Specifies that the message data for send and receive calls will be encrypted. Message data will be encrypted between the current application and the Broker. Encryption from the Broker to the partner application depends on the settings of the Broker and the partner application.

Since:
EntireX 6.1.1
See Also:
Constant Field Values

ENCRYPTION_LEVEL_TARGET

```
public static final int ENCRYPTION_LEVEL_TARGET
```

Specifies that the message data for send and receive calls will be encrypted. Data will be encrypted between the current application and the Broker, as well as between the Broker and the partner application of this communication. It is guaranteed that the message is encrypted from the client to the server and vice versa.

Since:
EntireX 6.1.1
See Also:
Constant Field Values

Constructor Detail

Broker

```
public Broker(java.lang.String brokerID,
              java.lang.String userID)
```

Creates a Broker object for the specified Broker address and user ID.

Parameters:
`brokerID` - The BrokerID, for a definition see `getBrokerID()`
`userID` - Authentication for the Broker instance.

Throws:

`java.lang.IllegalArgumentException` - if the broker ID or user ID is invalid.

Broker

```
public Broker(java.lang.String brokerID,
              java.lang.String userID,
              java.lang.String token)
```

Creates a Broker object for the specified Broker address, user ID and token.

Parameters:

`brokerID` - The BrokerID, for a definition see `getBrokerID()`
`userID` - Authentication for the Broker instance.
`token` - Token for the Broker instance. Mandatory when using units of work.

Throws:

`java.lang.IllegalArgumentException` - if the broker ID or user ID is invalid.

See Also:

`getToken()`

Method Detail

getVersion

```
public static java.lang.String getVersion()
```

Returns version information of the EntireX Java ACI package.

Returns:

Version information of Java ACI package as string.

---

getConnInfo

```
public java.lang.String getConnInfo()
```

Returns connection information. For TCP/IP or SSL connections the information contains host and port of the socket in use and details on the pool of sockets. This includes the pool size (as configured with the parameter 'poolsize=' in the Broker ID), 'Waits', the number of events waiting for a free socket, and 'MaxSockets', the accumulated number of sockets used during the lifetime of the application. For HTTP and HTTPS connections, this information is not available. This is for information purposes only.

Returns:

Connection information for this Broker object as string.

---

getBrokerID

```
public java.lang.String getBrokerID()
```

Returns the Broker ID which was specified in the constructor of this class.
The Broker ID may have the following syntax depending on the used transport method:
If the Broker ID starts with *http://* or *https://* HTTP/HTTPS tunneling via the EntireX TunnelServlet will be used. The Broker ID has the format *http://urlOfTunnelServlet?tunnelParams* or *https://urlOfTunnelServlet?tunnelParams*. Note that the *?tunnelParams* part is optional. The TunnelServlet accepts the parameters *broker* and *log* which overwrite the corresponding values in the servlet configuration. If the parameter *checkheaders=no* is specified, the HTTP headers returned by the web server are not checked. This might be necessary when using the Netscape Navigator with Sun's Java Plug-in and HTTPS.
If the Broker ID starts with *tcpip://* or if no protocol is specified TCP/IP will be used as the transport method. The Broker ID has the format *HostnamePortnumber* or *tcpip://HostnamePortnumber*.
HostnamePortnumber can be either *hostname*, *hostname:portnumber* or *:portnumber*. The hostname is either an IP hostname or a numeric IP address, the default is *localhost*. The port number must be numeric, default for TCP/IP is 1971, default for SSL is 1958.
If the Broker ID starts with *ssl://*, SSL will be used. The Broker ID has the format *ssl://HostnamePortnumber?sslParams*. *sslParams* need to specify *trust_store=file*, where *file* is the path name of a Java Keystore file which contains the list of trusted certificate authorities. Specify *verify_server=no*, if you do not want to check that the certificate of the SSL server is issued for the specified hostname.
If the SSL server requests a client certificate, *key_store=file* and *key_passwd=pwd* have to be specified. Again *file* is the path name of a Java Keystore file which contains the private key and *pwd* is the password which is needed to access the private key entry in the Keystore. Note that the & character must not appear in the password.

For TCP/IP and SSL there are two parameters to control the socket pooling. They are specified as part of the Broker ID. The parameter *poolsize* specifies the maximum number of socket connections which are kept in the socket pool. The default value for the *poolsize* parameter is 32. Set the *poolsize* parameter to 0 to disable the socket pooling. Automatic closing of socket connections is controlled by the parameter *pooltimeout*. If a socket connection has not been used for the specified the number of seconds, it will be closed automatically. The default for this parameter is 300 seconds.

The compression level can be set within the broker id. Add the parameter *compresslevel=<value>* to the broker id. Allowed values are

- 0 or NO_COMPRESSION or N
- 1 or BEST_SPEED
- 2
- 3
- 4
- 5
- 6 or DEFAULT_COMPRESSION or Y
- 7
- 8 or DEFLATED
- 9 or BEST_COMPRESSION

Returns:
Broker ID as string.

---

toString

```
public java.lang.String toString()
```

Overrides the `toString()` method of the class Object.

Overrides:
`toString` in class `java.lang.Object`
Returns:
BrokerID as a String.

---

getUserID

```
public java.lang.String getUserID()
```

Returns the current value of the user ID.

Returns:
user ID as String.

---

getSecurityToken

```
public byte[] getSecurityToken()
```

Returns the current value of the security token.
Note that the logoff() method resets the security token to `null` after the call has been made.

Returns:
Security Token as byte array or *null*.

---

setSecurityToken

```
public void setSecurityToken(byte[] s)
```

Sets the security token to the argument.
Note that the logon() methods reset the security token to `null` before the call is done and the logoff() method resets the security token after the call has been made.

Parameters:
`s` - new Security Token as byte array.

---

getToken

```
public java.lang.String getToken()
```

Returns the token specified in the constructor of this class.
Working with UnitofWorks requires a token. Using a token an application can reconnect to a Conversation or UnitofWork which was created by a different process.
When working without a token, the EntireX Broker identifies an (client or server) application by the user ID and an internal unique identifier. This internal identifier is unique to each object instance of this Broker class.
When a token is specified, the EntireX Broker identifies an application by the user ID and token. When working with tokens, ensure that all applications which communicate with the same EntireX Broker in parallel either use different user IDs or different tokens.

Returns:

Token as String or *null*.

---

setSecurity

```
public void setSecurity(BrokerSecurity securityObject,
                        boolean doEncrypt)
```

Specifies a security object, enables BrokerSecurity and permits encryption/decryption of message data.
The security object is used by the autoLogon() and logon() methods and when sending/receiving message data.
To use the security implementation of EntireX, call the method useEntireXSecurity.

Parameters:

`securityObject` - object which implements the BrokerSecurity interface.
`doEncrypt` - if *true* then message data will be encrypted/decrypted.

Throws:

`java.lang.IllegalStateException` - if security object is already set.

See Also:

`logon(java.lang.String)`, `autoLogon(java.lang.String)`, `BrokerSecurity`, `useEntireXSecurity()`, `useEntireXSecurity(int)`

---

setSecurity

```
public void setSecurity(BrokerSecurity securityObject,
                        int encryptionLevel)
```

Specifies a security object, enables BrokerSecurity and permits encryption/decryption of message data.
The security object is used by the autoLogon() and logon() methods and when sending/receiving message data.
To use the security implementation of EntireX, call the method useEntireXSecurity.

Parameters:

`securityObject` - object which implements the BrokerSecurity interface.
`encryptionLevel` - one of the values ENCRYPTION_LEVEL_NONE, ENCRYPTION_LEVEL_BROKER, or ENCRYPTION_LEVEL_TARGET.

Throws:

`java.lang.IllegalArgumentException` - if encryptionLevel is invalid.
`java.lang.IllegalStateException` - if security object is already set.

Since:

6.1.1

See Also:

`logon(java.lang.String)`, `autoLogon(java.lang.String)`, `BrokerSecurity`, `useEntireXSecurity()`, `useEntireXSecurity(int)`, `ENCRYPTION_LEVEL_NONE`, `ENCRYPTION_LEVEL_BROKER`, `ENCRYPTION_LEVEL_TARGET`

---

setSecurity

```
public void setSecurity(BrokerSecurity securityObject,
                        int encryptionLevel,
                        boolean autoMode)
```

Specifies a security object, enables BrokerSecurity and permits encryption/decryption of message data.
The security object is used by the autoLogon() and logon() methods and when sending/receiving message data.
To use the security implementation of EntireX, call the method useEntireXSecurity.

Parameters:

`securityObject` - object which implements the BrokerSecurity interface.
`encryptionLevel` - one of the values ENCRYPTION_LEVEL_NONE, ENCRYPTION_LEVEL_BROKER, or ENCRYPTION_LEVEL_TARGET.
`autoMode` - if `true`, let the Broker object figure out whether the Broker uses security. If the Broker uses security, set the specified `securityObject`.

Throws:

`java.lang.IllegalArgumentException` - if `encryptionLevel` is invalid.
`java.lang.IllegalStateException` - if security object is already set.

Since:

7.2.1

See Also:

`logon(java.lang.String)`, `autoLogon(java.lang.String)`, `BrokerSecurity`, `useEntireXSecurity()`, `useEntireXSecurity(int)`, `ENCRYPTION_LEVEL_NONE`, `ENCRYPTION_LEVEL_BROKER`, `ENCRYPTION_LEVEL_TARGET`

---

useEntireXSecurity

```
public void useEntireXSecurity()
```

Enables EntireX Security for this Broker instance.
Message data will not be encrypted.

Throws:

`java.lang.IllegalStateException` - if EntireX Security is already set.

Since:

6.1.1

See Also:

`logon(java.lang.String)`, `autoLogon(java.lang.String)`, `useEntireXSecurity(int)`

useEntireXSecurity

```
public void useEntireXSecurity(int encryptionLevel)
```

Enables EntireX Security for this Broker instance.
Message data will be encrypted according to the encryptionLevel parameter.

Parameters:

`encryptionLevel` - one of the values ENCRYPTION_LEVEL_NONE, ENCRYPTION_LEVEL_BROKER, or ENCRYPTION_LEVEL_TARGET.

Throws:

`java.lang.IllegalArgumentException` - if encryptionLevel is invalid.
`java.lang.IllegalStateException` - if EntireX Security is already set.

Since:

6.1.1

See Also:

`logon(java.lang.String)`, `autoLogon(java.lang.String)`, `ENCRYPTION_LEVEL_NONE`, `ENCRYPTION_LEVEL_BROKER`, `ENCRYPTION_LEVEL_TARGET`

---

useEntireXSecurity

```
public void useEntireXSecurity(boolean autoMode)
```

Enables EntireX Security for this Broker instance.

Parameters:

`autoMode` - if `true`, let the Broker object figure out whether the Broker uses security. If the Broker uses security, set the `EntireXSecurity` object.

Throws:

`java.lang.IllegalStateException` - if EntireX Security is already set.

Since:

7.2.1

See Also:

`logon(java.lang.String)`, `autoLogon(java.lang.String)`

---

useEntireXSecurity

```
public void useEntireXSecurity(int encryptionLevel,
                               boolean autoMode)
```

Enables EntireX Security for this Broker instance.
Message data will be encrypted according to the encryptionLevel parameter.

Parameters:

`encryptionLevel` - one of the values ENCRYPTION_LEVEL_NONE, ENCRYPTION_LEVEL_BROKER, or ENCRYPTION_LEVEL_TARGET.
`autoMode` - if `true`, let the Broker object figure out whether the Broker uses security. If the Broker uses security, set the `EntireXSecurity` object.

Throws:

`java.lang.IllegalArgumentException` - if encryptionLevel is invalid.
`java.lang.IllegalStateException` - if EntireX Security is already set.

Since:

7.2.1

See Also:

`logon(java.lang.String)`, `autoLogon(java.lang.String)`, `ENCRYPTION_LEVEL_NONE`, `ENCRYPTION_LEVEL_BROKER`, `ENCRYPTION_LEVEL_TARGET`

---

kernelversion

```
public java.lang.String kernelversion()
                                throws BrokerException
```

Retrieves the version of the EntireX Broker.

Returns:

the Broker version and platform (if supported by Broker)

Throws:

`BrokerException` - if the Broker call fails.

---

logon

```
public void logon()
           throws BrokerException
```

Logs the application on to EntireX Broker, either as client or server.

Throws:

`BrokerException` - if the Broker call fails.

---

logon

```
public void logon(java.lang.String password)
           throws BrokerException
```

Logs the application on to EntireX Broker with a password, either as client or server.
If EntireX Broker is running in a secured environment and a security object has been set, the `logon()` method performs the authentication process.

Parameters:

`password` - Password used for authentication together with the user ID.

Throws:

`BrokerException` - if the Broker call fails.

---

logon

```
public void logon(java.lang.String password,
                  java.lang.String newpassword)
           throws BrokerException
```

Logs the application on to EntireX Broker with a password and new password, either as client or server.
If EntireX Broker is running in a secured environment and a security object has been set, the logon() method performs the authentication process.

Parameters:

`password` - Password used for authentication together with the user ID.
`newpassword` - The new password is transmitted to the Broker security exit to allow a password change.

Throws:

`BrokerException` - if the Broker call fails.

---

autoLogon

```
public void autoLogon(java.lang.String password)
```

Provides a password to this instance of the Broker object to be used in subsequent calls. Use this method only when EntireXSecurity is enabled. If `autoLogon` is used, no logon to the Broker will be done and the password will be sent with every subsequent Broker call. If `logon` is used, a logon to the Broker will be done and the password will be sent only with the logon call.
You should use `autoLogon` whenever there is a high probability that a sequence of Broker calls is interrupted by a client non-activity timeout. In the case of a client timeout the 0008 0146 or 0002 0134 is avoided when autoLogon has been called.

Parameters:

`password` - Password used for authentication together with the user ID.

Since:

5.3.1.2

---

logoff

```
public void logoff()
           throws BrokerException
```

Logs off the application from EntireX Broker. The logoff() method should be called after the last interaction with the Broker. EntireX Broker will release all resources used by the application.

Throws:

`BrokerException` - a Broker Exception

---

setTrace

```
public static void setTrace(int level)
```

Sets the trace level. If set to 0, then no trace output will be produced. Trace level 1 will trace all Broker calls and other major actions. Trace level 2 will dump the send and receive buffer. Trace level 3 will dump the buffers sent to the Broker and received from the Broker. Can be initialized using the entirex.trace system property. Trace output will be written to standard output (java.lang.System.out) or the specified PrintWriter object.

Parameters:

`level` - Trace level.

See Also:

`setTrace(int, java.io.PrintWriter)`

---

setTrace

```
public static void setTrace(int level,
                            java.io.PrintWriter pw)
```

Sets the trace level. If set to 0, then no trace output will be produced. Trace level 1 will trace all Broker calls and other major actions. Trace level 2 will dump the send and receive buffer. Trace level 3 will dump the buffers sent to the Broker and received from the Broker. Trace output will be written to the specified PrintWriter object.

Parameters:

`level` - Trace level.

`pw` - instance of PrintWriter object which will receive the trace output.

Since:

EntireX 5.2.1

---

getTrace

```
public static int getTrace()
```

Returns the current trace level.

Returns:

Trace level as Int.

---

reconnect

```
public void reconnect(java.lang.String userID)
```

Reconnects this Broker object to the Broker address specified in the constructor and set the user ID. If this changes the user ID, call `logon` afterwards.

Parameters:

`userID` - Authentication for the Broker instance.

Throws:

`java.lang.IllegalArgumentException` - if the user ID is invalid.

Since:

EntireX 5.2.1

---

reconnect

```
public void reconnect(java.lang.String userID,
                      java.lang.String token)
```

Reconnects this Broker object to the Broker address specified in the constructor and set user ID and token. If this changes the user ID, call `logon` afterwards.

Parameters:

`userID` - Authentication for the Broker instance.

`token` - Token for the Broker instance.

Throws:

`java.lang.IllegalArgumentException` - if the user ID is invalid.

Since:

EntireX 5.2.1

---

disconnect

```
public void disconnect()
```

Disconnects this Broker instance from the EntireX Broker. This method allows the EntireX Java runtime to close or reuse the network resources allocated for this Broker instance. You can use this method after `logoff`. If `disconnect` is not used, the network resources are freed if the `Broker` object is destroyed or the socket pooling frees the unused sockets.
You should call this method whenever your Java application continues to run after communication with this Broker instance has been terminated (i.e. in a multi-threading application).
Reusing this Broker object will establish a new connection to the EntireX Broker.

Since:

EntireX 5.2.1

---

getUniqueID

```
public java.lang.String getUniqueID()
                              throws java.lang.IllegalStateException
```

Returns a String object, which is unique for this instance of the Broker object. A value can be returned only when at least one Broker call has been made using this Broker instance. Otherwise an IllegalStateException is thrown. Calling this method for the same Broker instance always returns the same value.

Returns:

uniqueID the unique string

Throws:

`java.lang.IllegalStateException` - if uniqueID cannot be obtained.

Since:
EntireX 7.1.1.0

---

getCompressionLevel

```
public int getCompressionLevel()
```

Returns the compression level.

Returns:
the compression level.
Since:
EntireX 7.1.1.0

---

setCompressionLevel

```
public void setCompressionLevel(int level)
```

Sets the compression level. Valid compression levels are -1 (default compression), 0 (no compression), 1 (best speed) up to 9 (best compression).

Parameters:
`level` - the compression level.
Since:
EntireX 7.1.1.0

---

setCompressionLevel

```
public void setCompressionLevel(java.lang.String level)
```

Sets the compression level. If the string starts with 'Y', compression is turned on with level 6. If the string starts with 'N', compression is turned off (Level 0). If the string denotes an integer between 0 and 9, compression is set to this level. In addition, you may use the strings 'BEST_COMPRESSION' (level 9), 'BEST_SPEED' (1), 'DEFAULT_COMPRESSION' (6), 'DEFLATED' (8), 'NO_COMPRESSION' (0). You may use the constants defined in class `java.util.zip.Deflater`.

Parameters:
`level` - the compression level.
Since:
EntireX 7.1.1.0

---

setThreadRunner

```
public static void setThreadRunner(ThreadRunner t)
```

Sets the `ThreadRunner` object, which wraps the method to start a new thread.

Parameters:
`t` - the `ThreadRunner` object.

getThreadRunner

```
public static ThreadRunner getThreadRunner()
```

Gets the `ThreadRunner` object. The `ThreadRunner` object wraps the start method of the thread. This is used in application server to use the method of the application server to start a new thread.

Returns:
the `ThreadRunner` object

setTransportTimeout

```
public static void setTransportTimeout(int timeout)
                            throws BrokerException
```

Sets the socket timeout value in seconds. Used for the transport of messages over TCP/IP. Not used with HTTP. This timeout value is used for new sockets. It does not change the timeout for sockets in use. The system property `entirex.timeout` may be used to configure the timeout value.

Parameters:
`timeout` - the socket timeout value in seconds.
Throws:
`BrokerException` - if the timeout value is negative.
Since:
7.2.1.1
See Also:
`getTransportTimeout()`

getTransportTimeout

```
public static int getTransportTimeout()
                           throws BrokerException
```

Gets the socket timeout value in seconds. Reads the system property `entirex.timeout`. The system property is read once. The system property wins over use of `setTransportTimeout`.

Returns:
the socket timeout value in seconds.
Throws:
`BrokerException` - if the property `entirex.timeout` is not a number or negative.
Since:
7.2.1.1
See Also:
`setTransportTimeout(int)`

getApplicationName

```
public java.lang.String getApplicationName()
```

Gets the name of the application.

Returns:
the name of the application.

---

setApplicationName

```
public void setApplicationName(java.lang.String name)
```

Sets the name of the application. If the name is longer than 64 characters, the name is truncated to 64 characters.

Parameters:
`name` - the name of the application.
Throws:
`java.lang.IllegalArgumentException` - if the name is null or empty.

---

getPartnerBrokerId

```
public java.lang.String getPartnerBrokerId()
```

Gets the partner Broker id.

Returns:
the partner Broker id.

---

isCommandLogging

```
public boolean isCommandLogging()
```

Gets the state of the command logging.

Returns:
`true` if command logging is on. `false` otherwise.

---

setCommandLogging

```
public void setCommandLogging(boolean value)
```

Switch command logging on and off. The value is used for all following calls of this `Broker` object.

Parameters:
`value` - `true` to turn command logging on, `false` to turn it off.

getIAFToken

```
public byte[] getIAFToken()
```

Get the IAF Security Token. The IAF token is generated by the Broker if the Broker is set-up with IAF.

Returns:
the IAF Security Token.

setIAFToken

```
public void setIAFToken(byte[] iafToken)
```

Sets the IAF Security Token. If an IAF token is set, this is used to identify the user. The next `Broker.logon` call uses the IAF token, not the user id / password if the token is not `null`.

Parameters:
`iafToken` - the IAF Security Token or `null`.

Overview Package Class Tree Deprecated Index Help

PREV CLASS **NEXT CLASS** **FRAMES NO FRAMES All Classes**

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

com.softwareag.entirex.aci

Class BrokerAttachInfo

```
java.lang.Object
  └ com.softwareag.entirex.aci.BrokerAttachInfo
```

```
public class BrokerAttachInfoextends java.lang.Object
```

Contains information for attach servers returned by the `receiveAttachInfo()` method for an attach server.
Based on this information new server replicas can be started.

Since:
EntireX 5.2.1
See Also:
`BrokerService.receiveAttachInfo()`

Field Summary

int	**activeConversations** Number of active conversations.
int	**missingServers** Number of unsuccessful server lookups.
int	**pendingConversations** Number of pending conversations.
int	**replicates** Number of registered server replicas.
java.lang.String	**serverAddress** The server address.

Method Summary

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

missingServers

```
public int missingServers
```

Number of unsuccessful server lookups.

---

replicates

```
public int replicates
```

Number of registered server replicas.

---

pendingConversations

```
public int pendingConversations
```

Number of pending conversations.

---

activeConversations

```
public int activeConversations
```

Number of active conversations.

---

serverAddress

```
public java.lang.String serverAddress
```

The server address.

---

com.softwareag.entirex.aci

Class BrokerCommunication

```
java.lang.Object
  └ com.softwareag.entirex.aci.BrokerCommunication
```

Direct Known Subclasses:
Conversation, UnitofWork

```
public abstract class BrokerCommunicationextends java.lang.Object
```

Abstract superclass for conversational communication and UnitofWork communication.
Defines methods which are used by both Conversation and UnitofWork objects.

See Also:
`Conversation`, `UnitofWork`

Field Summary

protected BrokerService	**brokerService** The BrokerService object to which the Communication belongs.

Method Summary

void	**dispose**() **Deprecated.** *This method does nothing, since no reference to the Conversation or UnitofWork object is held anymore.*
BrokerService	**getBrokerService**() Returns the BrokerService object to which the communication belongs.
byte[]	**getUserData**() Returns the user data associated with this communication.
ConversationState	**saveState**() Returns a ConversationState object for the current Conversation object or the current UnitofWork object.
void	**setUserData**(byte[] u) Sets the user data associated with this communication.

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

brokerService

```
protected BrokerService brokerService
```

The BrokerService object to which the Communication belongs. Should only be used in the direct subclasses.

Method Detail

getBrokerService

```
public BrokerService getBrokerService()
```

Returns the BrokerService object to which the communication belongs.

Returns:
reference to BrokerService object.

---

getUserData

```
public byte[] getUserData()
```

Returns the user data associated with this communication. User data can be up to 16 bytes and can contain any type of binary data. The user data is totally untouched by EntireX Broker. It is never transmitted from one application to another (i.e., from a client to a server). Both sides of a conversation can store different user data, and both sides always receive their own data.
The user data is transmitted on a send() method and returned with the message to the receiving application. User data can be overwritten with each send() call.

Returns:
user data as byte array or *null*.

---

setUserData

```
public void setUserData(byte[] u)
```

Sets the user data associated with this communication. User data can be up to 16 bytes.
Note: This method updates only the UserData property of this object.

Parameters:
u - user data as byte array
Throws:
`java.lang.IllegalArgumentException` - Thrown if the length of the user data exceeds 16 bytes.
See Also:
`getUserData()`

---

saveState

```
public ConversationState saveState()
                            throws java.lang.IllegalArgumentException
```

Returns a ConversationState object for the current Conversation object or the current UnitofWork object. This object can be used to restore the Conversation object or UnitofWork object in a separate Java process.

Returns:
the ConversationState object.
Throws:
`java.lang.IllegalArgumentException` - if conversation ID is invalid.
Since:
7.1.1.10
See Also:
`ConversationState`

---

dispose

```
public void dispose()
             throws BrokerException
```

Deprecated. *This method does nothing, since no reference to the Conversation or UnitofWork object is held anymore.*

Call this method to release the Conversation or UnitofWork object in case you do not call end() or cancel(). `dispose()` does not call the Broker.

Throws:
`BrokerException` - A Broker exception.
Since:
5.2.1.2

---

com.softwareag.entirex.aci

Class BrokerException

```
java.lang.Object
  └ java.lang.Throwable
      └ java.lang.Exception
          └ com.softwareag.entirex.aci.BrokerException
```

All Implemented Interfaces:

java.io.Serializable

```
public class BrokerExceptionextends java.lang.Exception
```

Exception class thrown by EntireX Java ACI/RPC classes.
Use `toString()` to retrieve the error message including the error class and error code.
Use `getMessage()` to retrieve the error message only.

See Also:

Serialized Form

Method Summary

int	**getErrorClass**() Returns the error class part of the Broker error.
int	**getErrorCode**() Returns the error code part of the Broker error.
java.lang.String	**getErrorInfo**() Returns additional information from the Broker call which caused the error.
java.lang.String	**toString**() Overrides the `toString` method.

Methods inherited from class java.lang.Throwable

`fillInStackTrace, getCause, getLocalizedMessage, getMessage, getStackTrace, initCause, printStackTrace, printStackTrace, printStackTrace, setStackTrace`

Methods inherited from class java.lang.Object
`clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait`

Method Detail

getErrorClass

`public int getErrorClass()`

Returns the error class part of the Broker error.

Returns:
Broker error class as int.

---

getErrorCode

`public int getErrorCode()`

Returns the error code part of the Broker error.

Returns:
Broker error code as int.

---

getErrorInfo

`public java.lang.String getErrorInfo()`

Returns additional information from the Broker call which caused the error.
If this is an error returned from EntireX Broker, the trace information for the Broker call and its result values are returned. This information is available only when tracing is enabled. Errors which are detected by Java ACI return `null`.

Returns:
Trace information (if tracing is enabled) for the Broker call which caused the error.

---

toString

`public java.lang.String toString()`

Overrides the `toString` method.

Overrides:
`toString` in class `java.lang.Throwable`

Returns:

A complete error message including error class and error code.

Overview Package Class Tree Deprecated Index Help

PREV CLASS NEXT CLASS **FRAMES NO FRAMES All Classes**

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

---

com.softwareag.entirex.aci

Class BrokerMessage

```
java.lang.Object
  └ com.softwareag.entirex.aci.BrokerMessage
```

```
public class BrokerMessageextends java.lang.Object
```

This class encapsulates a single message, which will be sent to or received from another participant via the EntireX Broker.
The contents of a message is always a byte array. Constructors and methods, which provide a transformation between byte arrays and String objects, are available.
When the message has been created as the result of a successful receive or sendReceive call, the `getService()`, `getConversation()` and `getUnitofWork()` methods can be used to navigate to the context of the received message.

Constructor Summary

`BrokerMessage()` Creates a Message object with an empty message.	
`BrokerMessage(byte[] msg)` Creates a Message object initialized with the passed byte array.	
`BrokerMessage(java.lang.String msg)` Creates a BrokerMessage object initialized with the passed String object.	

Method Summary

byte[]	**getClientIAFToken**() Gets the IAF Security Token for the client.
java.lang.String	**getClientUID**() Returns the client's user ID.
Conversation	**getConversation**() Returns the corresponding Conversation object.
byte[]	**getMessage**() Returns the current message as a byte array.
BrokerService	**getService**() Returns the BrokerService object to which the message belongs.
UnitofWork	**getUnitofWork**() Returns the corresponding UnitofWork object.
void	**reply**(BrokerMessage msg) Sends a reply to a previously received message.
void	**setMessage**(byte[] s) Sets the current message to the passed byte array.
void	**setMessage**(java.lang.String s) Sets the current message to the passed string.
java.lang.String	**toString**() Returns the current message as a string.

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Constructor Detail

BrokerMessage

public **BrokerMessage**()

Creates a Message object with an empty message.

BrokerMessage

```
public BrokerMessage(byte[] msg)
```

Creates a Message object initialized with the passed byte array.

Parameters:
`msg` - the message as byte array.

BrokerMessage

```
public BrokerMessage(java.lang.String msg)
```

Creates a BrokerMessage object initialized with the passed String object. The String object is converted to a byte array.

Parameters:
`msg` - the message as a String object.

Method Detail

toString

```
public java.lang.String toString()
```

Returns the current message as a string.

Overrides:
`toString` in class `java.lang.Object`
Returns:
The message data as a String or `null`.

---

getClientUID

```
public java.lang.String getClientUID()
```

Returns the client's user ID. Only available after a receive issued by a server application.

Returns:
The client's user ID as a string or `null`.

---

getMessage

```
public byte[] getMessage()
```

Returns the current message as a byte array.

Returns:
the message as a byte array or `null`.

setMessage

```
public void setMessage(byte[] s)
```

Sets the current message to the passed byte array.

Parameters:

s - the message as byte array.

setMessage

```
public void setMessage(java.lang.String s)
```

Sets the current message to the passed string.

Parameters:

s - the message as string.

getService

```
public BrokerService getService()
```

Returns the BrokerService object to which the message belongs. Only available when the Message object has been created by a receive or sendReceive call.

Returns:

a reference to the BrokerService object.

getConversation

```
public Conversation getConversation()
```

Returns the corresponding Conversation object.
If the message has been created by a receive or sendReceive call and if the message is part of a conversational communication, this method returns the corresponding Conversation object. Otherwise, `null` is returned.

Returns:

a reference to the Conversation object or `null`.

getUnitofWork

```
public UnitofWork getUnitofWork()
```

Returns the corresponding UnitofWork object.
If the message has been created by a receive call and if the message is part of a UnitofWork communication, this method returns the corresponding UnitofWork object. Otherwise, `null` is returned.

Returns:

a reference to the Conversation object or `null`.

---

reply

```
public void reply(BrokerMessage msg)
           throws BrokerException
```

Sends a reply to a previously received message.
If the message has been created by a receive or sendReceive call and if the message is either the only message of a non-conversational communication or if the message is part of a conversational communication, this method sends a reply back to the originator of this message.

Parameters:

`msg` - BrokerMessage to send.

Throws:

`BrokerException` - A Broker exception.
`java.lang.IllegalArgumentException` - Thrown if the parameter is invalid.

---

getClientIAFToken

```
public byte[] getClientIAFToken()
```

Gets the IAF Security Token for the client. Only available after a receive issued by a server application.

Returns:

the IAF Security Token for the client.

Overview Package Class Tree Deprecated Index Help

PREV CLASS NEXT CLASS **FRAMES NO FRAMES All Classes**

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

---

com.softwareag.entirex.aci
Interface BrokerSecurity

All Known Implementing Classes:
EntireXSecurity

```
public interface BrokerSecurity
```

Interface which defines the interface to EntireX Security.
A security object used by Java ACI must implement this interface.
The class **EntireXSecurity** implements this interface and supports the standard EntireX Security.

See Also:
`EntireXSecurity`, `getSecurityToken()`

Method Summary

`void`	**`decryptData`**`(byte[] data)` Decrypts the received data in place.
`void`	**`encryptData`**`(byte[] data)` Encrypts the sent data in place.
`byte[]`	**`getNewpassword`**`()` Returns the encrypted Newpassword if it was supplied in the `prepareLogon` call.
`byte[]`	**`getPassword`**`()` Returns the encrypted password if it was supplied in the `prepareLogon` call.
`byte[]`	**`getSecurityToken`**`()` Returns a Security Token.
`void`	**`prepareLogon`**`(java.lang.String userID, byte[] password, byte[] newpassword, byte[] securityToken)` Encrypts the password(s) and generates a security token.

Method Detail

prepareLogon

```
void prepareLogon(java.lang.String userID,
                  byte[] password,
                  byte[] newpassword,
                  byte[] securityToken)
```

Encrypts the password(s) and generates a security token.
This method is always called inside the Broker `logon` methods. It is called before performing a logon call to the Broker. This method prepares the encryption of the password and the new password and the generation of the security token. These values will be retrieved by the `getPassword`, `getNewpassword` and `getSecurityToken` methods.

Parameters:

`userID` - user ID from Broker constructor.
`password` - Password from `logon` method. May be `null`.
`newpassword - New password from logon method. May be null.`
`securityToken - Security token from Broker object. May be null.`

See Also:

`Broker.logon(), Broker.logon(java.lang.String), Broker.logon(java.lang.String, java.lang.String), getPassword(), getNewpassword(), getSecurityToken()`

---

getPassword

```
byte[] getPassword()
```

`Returns the encrypted password if it was supplied in the prepareLogon call.`

Returns:

`The encrypted password or null.`

See Also:

`prepareLogon(java.lang.String, byte[], byte[], byte[])`

---

getNewpassword

```
byte[] getNewpassword()
```

`Returns the encrypted Newpassword if it was supplied in the prepareLogon call.`

Returns:

`The encrypted new password or null.`

See Also:

`prepareLogon(java.lang.String, byte[], byte[], byte[])`

---

getSecurityToken

byte[] **getSecurityToken**()

Returns a Security Token.

Returns:
The Security Token.
See Also:
prepareLogon(java.lang.String, byte[], byte[], byte[])

---

encryptData

void **encryptData**(byte[] data)

Encrypts the sent data in place.

Parameters:
data - The send data.

---

decryptData

void **decryptData**(byte[] data)

Decrypts the received data in place.

Parameters:
data - The receive data.

---

com.softwareag.entirex.aci

Class BrokerService

```
java.lang.Object
  └ com.softwareag.entirex.aci.BrokerService
```

Direct Known Subclasses:
RPCService

```
public class BrokerServiceextends java.lang.Object
```

Represents a service that is available through the EntireX Broker and is used by both clients which want to access a service, and by servers which register the services they provide.
An instance of this class is always linked to an instance of a Broker object.

Since:
EntireX 5.2.1

Field Summary

static java.lang.String	**DEFAULT_WAITTIME** The initial default wait time used in the sendReceive() method call and all receive() method calls.

Constructor Summary

BrokerService(Broker broker, java.lang.String serverAddr) Creates a new BrokerService object.
BrokerService(Broker broker, java.lang.String serverClass, java.lang.String serverName, java.lang.String serviceName) Creates a new BrokerService object.

Method Summary

void	**cancelallConversations**() Cancels all conversations for this service.

void	**deregister**() Deregisters a registered BrokerService object from the EntireX Broker.
void	**deregisterImmediate**() Deregisters a registered BrokerService object from the EntireX Broker.
void	**endallConversations**() Ends all conversations for this service.
Broker	**getBroker**() Returns the Broker object to which the service belongs.
java.lang.String	**getCharacterEncoding**() Gets the character encoding name or `null`.
java.lang.String	**getDefaultWaittime**() Returns the default wait time.
java.lang.String	**getEnvironment**() Returns the environment.
int	**getMaxReceiveLen**() Returns the current maximum receive length.
java.lang.String	**getServerClass**() Returns the server class.
java.lang.String	**getServerName**() Returns the server name.
java.lang.String	**getServiceName**() Returns the service name.
boolean	**isGeneric**() Returns an indication whether this is a generic service.
BrokerMessage	**receive**() Receives an incoming request or message.
BrokerMessage	**receive**(java.lang.String wait) Receives an incoming request or message.
BrokerMessage	**receiveAny**() Receives an incoming request or message.
BrokerAttachInfo	**receiveAttachInfo**() Receives for attach servers a notification about waiting clients.
BrokerMessage	**receiveOld**() Receives an incoming request or message.
void	**register**() Registers a BrokerService with the EntireX Broker.
void	**registerAttach**() Registers a Broker Attach Server with the EntireX Broker.
boolean	**replyError**(java.lang.String errorCode, java.lang.String errorText) Sends a reply with an error code to the Broker.

void	**send**(BrokerMessage msg) Sends an asynchronous non-conversational message.
BrokerMessage	**sendReceive**(BrokerMessage msg) Sends a synchronous non-conversational message.
BrokerMessage	**sendReceive**(BrokerMessage msg, java.lang.String wait) Sends a synchronous non-conversational message.
void	**setAdjustReceiveLen**(boolean adj) Enables or disables the automatic adjustment of the maximum receive length when an application issues `receive` calls.
void	**setCharacterEncoding**(java.lang.String enc) Sets the character encoding for the payload encoding.
void	**setDefaultWaittime**(java.lang.String wait) Sets the value of the default wait time field to the argument.
void	**setEnvironment**(java.lang.String env) Sets the value of the environment field to the argument.
void	**setLogicalBroker**(java.lang.String logicalBroker) Resolves a logical Broker and changes the Broker object accordingly.
void	**setLogicalBroker**(java.lang.String logicalBroker, java.lang.String logicalSetName) Resolves a logical Broker and changes the Broker object accordingly.
protected void	**setLogicalBroker**(java.lang.String logicalBroker, java.lang.String serverAddress, java.lang.String logicalSetName) Resolves a logical Broker and changes the Broker object and server address accordingly.
void	**setLogicalService**(java.lang.String logicalService) Resolves a logical service and changes the Broker object and server address accordingly.
void	**setLogicalService**(java.lang.String logicalService, java.lang.String logicalSetName) Resolves a logical service and changes the Broker object and server address accordingly.
void	**setMaxReceiveLen**(int len) Sets the current maximum receive length.
java.lang.String	**toString**() Overrides the toString() method of Object.
boolean	**useCodePage**() If communicating with broker version 7.1.x and below check if a locale string is sent to the broker or not.
void	**useCodePage**(boolean b) Force to send a locale string if communicating with Broker version 7.1.x and below.

Methods inherited from class java.lang.Object
`clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait`

Field Detail

DEFAULT_WAITTIME

```
public static final java.lang.String DEFAULT_WAITTIME
```

The initial default wait time used in the sendReceive() method call and all receive() method calls. The value is 60 seconds.
The default wait time can be changed using the setDefaultWaittime() method call.
The value 'YES', specified in the general documentation of the broker ACI fields is not allowed for Java components.

Since:
EntireX 5.2.1

See Also:
`setDefaultWaittime(java.lang.String)`, Description of WAIT values, Constant Field Values

Constructor Detail

BrokerService

```
public BrokerService(Broker broker,
                     java.lang.String serverAddr)
```

Creates a new BrokerService object.

Parameters:
`broker` - An instance of a Broker object.
`serverAddr` - The address of the server in the form "serverClass/serverName/serviceName".

Throws:
`java.lang.IllegalArgumentException` - Thrown if a parameter is invalid.

Since:
EntireX 5.2.1

See Also:
`DEFAULT_WAITTIME`

BrokerService

```
public BrokerService(Broker broker,
                     java.lang.String serverClass,
                     java.lang.String serverName,
                     java.lang.String serviceName)
```

Creates a new BrokerService object.

Parameters:

`broker` - An instance of a Broker object.
`serverClass` - The name of the server class.
`serverName` - The name of the server.
`serviceName` - The name of the service.

Throws:

`java.lang.IllegalArgumentException` - Thrown if a parameter is invalid.

See Also:

`DEFAULT_WAITTIME`

Method Detail

getBroker

```
public Broker getBroker()
```

Returns the Broker object to which the service belongs.

Returns:

An instance of the Broker class.

---

toString

```
public final java.lang.String toString()
```

Overrides the toString() method of Object.

Overrides:

`toString` in class `java.lang.Object`

Returns:

The complete service address as a string in the form ServerClass/ServerName/ServiceName.

---

getServerName

```
public final java.lang.String getServerName()
```

Returns the server name.

Returns:

The server name as a string.

getServerClass

```
public final java.lang.String getServerClass()
```

Returns the server class.

Returns:
The server class as a string.

getServiceName

```
public final java.lang.String getServiceName()
```

Returns the service name.

Returns:
The service name as a string.

getEnvironment

```
public final java.lang.String getEnvironment()
```

Returns the environment.

Returns:
The environment as a String.
See Also:
`setEnvironment(java.lang.String)`

setEnvironment

```
public final void setEnvironment(java.lang.String env)
```

Sets the value of the environment field to the argument.
The environment field provides information for an EntireX Broker translation routine. The environment field is transmitted on all send and receive calls to the Broker.
This value can be specified only at the BrokerService level.

Parameters:
`env` - The value of the Environment field. The maximum length is 32 characters.

getDefaultWaittime

```
public final java.lang.String getDefaultWaittime()
```

Returns the default wait time.

Returns:
The default wait time as a String.

Since:
EntireX 5.2.1
See Also:
`DEFAULT_WAITTIME`

---

setDefaultWaittime

```
public final void setDefaultWaittime(java.lang.String wait)
```

Sets the value of the default wait time field to the argument.

Parameters:
`wait` - The new value of the default wait time. Allowed values are
- `NO` No wait time. Control is returned to the caller immediately.
- `nS` The number of seconds the caller will wait for a reply.
- `nM` The number of minutes the caller will wait for a reply.
- `nH` The number of hours the caller will wait for a reply.

In all cases the number `n` has to be smaller than 100000 and not negative. Setting an illegal value results in an `IllegalArgumentException` on `send`, `sendReceive`, or `receive`. The value 'YES', specified in the general documentation of the broker ACI fields is illegal for Java components.

Since:
EntireX 5.2.1
See Also:
`DEFAULT_WAITTIME`, Detailed description of WAIT values

---

getMaxReceiveLen

```
public final int getMaxReceiveLen()
```

Returns the current maximum receive length.

Returns:
The current maximum receive length in number of bytes.

---

setMaxReceiveLen

```
public final void setMaxReceiveLen(int len)
```

Sets the current maximum receive length.
The receive length specifies the maximum number of bytes a `receive` or `sendReceive` method can receive. If the returned data is longer than the current maximum receive length, the data is truncated and a BrokerException with errorclass=20 and errorcode=94 is thrown.
This value can only be specified on the BrokerService level.
The default value for the receive length is 7168 bytes.

Parameters:
`len` - The new receive length in bytes.

setAdjustReceiveLen

```
public final void setAdjustReceiveLen(boolean adj)
```

Enables or disables the automatic adjustment of the maximum receive length when an application issues `receive` calls.

Parameters:

`adj` - `true` to enable receive length adjustment, `false` to disable receive length adjustment.

Since:

EntireX 5.2.1

isGeneric

```
public final boolean isGeneric()
```

Returns an indication whether this is a generic service.
A generic service has one of its identifying fields (ServerClass/ServerName/ServiceName) specified as "*".
`register`, `registerAttach`, and `send` calls are not allowed with generic services.

Returns:

`true` if this is a generic Service, `false` otherwise.

useCodePage

```
public final void useCodePage(boolean b)
```

Force to send a locale string if communicating with Broker version 7.1.x and below. You cannot use a codepage other than the default encoding of the JVM.

If set conversion is enabled. The default code page configured for your Java Virtual Machine will be sent to the Broker as the LOCALE_STRING. To change the default, see the Java Virtual Machine documentation. On some implementations, it can be changed with the `file.encoding` property. See the documentation of Java Virtual Machine on Internationalization for the supported code pages.

If not set the used approach translation or conversion is determined by the configuration of your partner (client or server). See Broker's handling of conversion/translation within the EntireX Internationalization overview for more information. Determines the translation processing of the EntireX Broker. It is assumed that you have read the introductory document Internationalization with EntireX and are familiar with the various internationalization approaches.

Parameters:

`b` - `true` to enable conversion; `false` to use configuration of partner.

Since:

EntireX 5.2.1

useCodePage

```
public final boolean useCodePage()
```

If communicating with broker version 7.1.x and below check if a locale string is sent to the broker or not. It is assumed that you have read the introductory document Internationalization with EntireX and are familiar with the various internationalization approaches.

Returns:

`true` if conversion is enabled; `false` if conversion or translation is determined by the partner.

register

```
public final void register()
                  throws BrokerException
```

Registers a BrokerService with the EntireX Broker.
This method is used by a server application to register the current BrokerService object with the EntireX Broker.
To register a service, the service must be defined in the EntireX Broker attribute file.

Throws:

`BrokerException` - A Broker exception.

registerAttach

```
public final void registerAttach()
                         throws BrokerException
```

Registers a Broker Attach Server with the EntireX Broker.
This method is used to register an Attach Server with EntireX Broker. Use the receiveAttachInfo() method to process attch requests.
To register a service, the service must be defined in the EntireX Broker attribute file.

Throws:

`BrokerException` - A Broker exception.

See Also:

`receiveAttachInfo()`

deregister

```
public final void deregister()
                     throws BrokerException
```

Deregisters a registered BrokerService object from the EntireX Broker.
This method is used by a server application to deregister the current BrokerService object from the EntireX Broker. No new conversations are accepted for the affected service, but existing ones can continue until they are normally ended.
The application that issues this deregister call must remain active until all conversations are ended.

Throws:

`BrokerException` - A Broker exception.

---

deregisterImmediate

```
public final void deregisterImmediate()
                            throws BrokerException
```

Deregisters a registered BrokerService object from the EntireX Broker.
This method is used by a server application to deregister the current BrokerService object from the EntireX Broker. The service is removed immediately, all affected conversations are ended and the partners are informed. Any active units of work are backed out.

Throws:

`BrokerException` - A Broker exception.

---

send

```
public final void send(BrokerMessage msg)
               throws BrokerException
```

Sends an asynchronous non-conversational message.
Sends the message to the Broker without waiting for an answer. Suitable for immediate, one-way messages.

Parameters:

`msg` - BrokerMessage to send.

Throws:

`BrokerException` - A Broker exception.
`java.lang.IllegalArgumentException` - Thrown if no message is specified.

See Also:

`sendReceive(BrokerMessage, java.lang.String)`

---

sendReceive

```
public final BrokerMessage sendReceive(BrokerMessage msg)
                            throws BrokerException
```

Sends a synchronous non-conversational message.
Sends the message to the Broker and waits to receive an answer. The default wait time is used. The answer is returned in the BrokerMessage object. Uses the maximum receive length.

Parameters:

`msg` - BrokerMessage to send.

Returns:

BrokerMessage the received message.

Throws:

`BrokerException` - A Broker exception.

Since:

EntireX 5.2.1

See Also:

`send(com.softwareag.entirex.aci.BrokerMessage)`

---

sendReceive

```
public final BrokerMessage sendReceive(BrokerMessage msg,
                                       java.lang.String wait)
                                throws BrokerException
```

Sends a synchronous non-conversational message.
Sends the message to the Broker and waits the specified time to receive an answer. The answer is returned in the BrokerMessage object. Uses the maximum receive length.

Parameters:

`msg` - BrokerMessage to send.
`wait` - A timeout period, how long to wait for an immediate reply, measured in seconds/minutes/hours (depending on the trailing S/M/H character). The value 'YES', specified in the general documentation of the broker ACI fields is not allowed for Java components.

Returns:

BrokerMessage the received message.

Throws:

`BrokerException` - A Broker exception.
`java.lang.IllegalArgumentException` - Thrown if no message is specified or parameter is invalid.

See Also:

`send(com.softwareag.entirex.aci.BrokerMessage)`, Description of WAIT values

---

receive

```
public final BrokerMessage receive()
                          throws BrokerException
```

Receives an incoming request or message.
This receive method will accept only non-conversational messages and new conversations.
The answer is returned in the BrokerMessage object. Uses the maximum receive length.
If this receive call opens a new conversation, an appropriate Conversation object is created automatically. The default wait time determines how long to wait for messages.

Returns:

A BrokerMessage object.

Throws:

`BrokerException` - A Broker exception.

Since:

EntireX 5.2.1

---

receive

```
public final BrokerMessage receive(java.lang.String wait)
                            throws BrokerException
```

Receives an incoming request or message.
This receive method will accept only non-conversational messages and new conversations.
The answer is returned in the BrokerMessage object. Uses the maximum receive length.
If this receive call opens a new conversation, an appropriate Conversation object is created automatically.

Parameters:

`wait` - A timeout period, how long to wait for a receive, measured in seconds/minutes/hours (depending on the trailing S/M/H character). When null or 'NO' is specified, the operation is non-blocked. If no message is available, a Broker Exception with errorclass=74 and errorcode=74 is thrown. The value 'YES' is illegal.

Returns:

A BrokerMessage object.

Throws:

`BrokerException` - A Broker exception.
`java.lang.IllegalArgumentException` - Thrown if parameter is invalid.

---

receiveOld

```
public final BrokerMessage receiveOld()
                               throws BrokerException
```

Receives an incoming request or message.
This receive method will accept only messages from existing conversations.
The answer is returned in the BrokerMessage object. Uses the maximum receive length and the default wait time.

Returns:

A BrokerMessage object.

Throws:

`BrokerException` - A Broker exception.

Since:

EntireX 5.2.1

---

receiveAny

```
public final BrokerMessage receiveAny()
                               throws BrokerException
```

Receives an incoming request or message.
This receive method will accept only non-conversational messages, messages from existing conversations, and new conversations.
The answer is returned in the BrokerMessage object. Uses the maximum receive length and the default wait time.
If this receive call opens a new conversation, an appropriate Conversation object is created automatically.

Returns:
A BrokerMessage object.
Throws:
`BrokerException` - A Broker exception.
Since:
EntireX 5.2.1

---

replyError

```
public final boolean replyError(java.lang.String errorCode,
                                java.lang.String errorText)
                         throws BrokerException
```

Sends a reply with an error code to the Broker. A server uses this method to indicate an error.

Parameters:
`errorCode` - the error code (must be 8 characters long).
`errorText` - the error text.
Returns:
`true` if the reply is send and the Broker supports this call. `false` if the current conversation is canceled. This is the way to signal an error for Brokers which do not support the function REPLYERROR. In this case `errorCode` and `errorText` are ignored.
Throws:
`BrokerException` - if the Broker replies with an error code or this service is an attach service or a generic service.
`java.lang.IllegalArgumentException` - if no message is specified.
Since:
7.2.1

---

endallConversations

```
public final void endallConversations()
                              throws BrokerException
```

Ends all conversations for this service. Receives for outstanding messages are possible.

Throws:
`BrokerException` - if the Broker replies with an error code.

---

cancelallConversations

```
public final void cancelallConversations()
                                  throws BrokerException
```

Cancels all conversations for this service.

Throws:
`BrokerException` - A Broker exception.

receiveAttachInfo

```
public final BrokerAttachInfo receiveAttachInfo()
                                        throws BrokerException
```

Receives for attach servers a notification about waiting clients. A server which has been registered as an attach server is able to obtain information when client requests cannot be handled either because no server is registered for this service or all registered servers are busy.
Calling this method puts the thread into a wait state until either the default wait time expires (BrokerException 0074 0074) or a client request cannot be handled by an active server. In the latter case a new instance of a BrokerAttachInfo object is returned. The attach server can evaluate this information to decide to start a new server instance.
If an attach server has been registered for more than one service, you have to check the serverAddress field in the BrokerAttachInfo object to find the server.

Returns:
A new instance of a BrokerAttachInfo object.
Throws:
`BrokerException` - A Broker exception.
Since:
EntireX 5.2.1
See Also:
`registerAttach()`

---

setLogicalService

```
public final void setLogicalService(java.lang.String logicalService)
                            throws BrokerException
```

Resolves a logical service and changes the Broker object and server address accordingly. For a logical service the real service and the real Broker ID are retrieved from an LDAP server. ’DefaultSet’ is used as the name of the set.

Parameters:
`logicalService` - the logical service.
Throws:
`BrokerException` - the BrokerException thrown by `LocationTransparencyService`
See Also:
`setLogicalService(String, String)`

---

setLogicalService

```
public final void setLogicalService(java.lang.String logicalService,
                                    java.lang.String logicalSetName)
                            throws BrokerException
```

Resolves a logical service and changes the Broker object and server address accordingly. For a logical service the real service and the real Broker ID using the name of the set are retrieved from an LDAP server.

Parameters:

`logicalService` - the logical service.

`logicalSetName` - the logical set name.

Throws:

`BrokerException` - the BrokerException thrown by `LocationTransparencyService`

See Also:

`LocationTransparencyService`

---

setLogicalBroker

```
public final void setLogicalBroker(java.lang.String logicalBroker)
                            throws BrokerException
```

Resolves a logical Broker and changes the Broker object accordingly. For a logical Broker the real Broker ID is retrieved from an LDAP server. 'DefaultSet' is used as the name of the set.

Parameters:

`logicalBroker` - the logical broker.

Throws:

`BrokerException` - the BrokerException thrown by `LocationTransparencyService`

See Also:

`setLogicalBroker(String)`

---

setLogicalBroker

```
public final void setLogicalBroker(java.lang.String logicalBroker,
                                   java.lang.String logicalSetName)
                            throws BrokerException
```

Resolves a logical Broker and changes the Broker object accordingly. For a logical Broker the real Broker ID using the name of the set is retrieved from an LDAP server.

Parameters:

`logicalBroker` - the logical Broker.

`logicalSetName` - the logical set name.

Throws:

`BrokerException` - the BrokerException thrown by `LocationTransparencyService`

See Also:

`LocationTransparencyService`

---

setLogicalBroker

```
protected final void setLogicalBroker(java.lang.String logicalBroker,
                                      java.lang.String serverAddress,
                                      java.lang.String logicalSetName)
                               throws BrokerException
```

Resolves a logical Broker and changes the Broker object and server address accordingly. For a logical Broker the real Broker ID using the name of the set is retrieved from an LDAP server.

Parameters:

`logicalBroker` - the logical Broker.
`serverAddress` - the server address.
`logicalSetName` - the logical set name.

Throws:

`BrokerException` - the BrokerException thrown by `LocationTransparencyService`

See Also:

`LocationTransparencyService`

---

setCharacterEncoding

```
public void setCharacterEncoding(java.lang.String enc)
                          throws BrokerException
```

Sets the character encoding for the payload encoding. The default is the default encoding of your JVM.

The encoding is always sent as the locale string to the Broker if you are communicating with a Broker version 7.2.x or above.

If you are communicating with a Broker version 7.1.x or below the encoding is sent as the locale string to the Broker after setting the locale string with this method or after calling `useCodePage(true)`. Setting `enc` to `null` disables this and sends no character encoding name to the Broker. In this case, the default encoding of the JVM is used and the character encoding is not send to the Broker. Use `setCharacterEncoding("LOCAL")` to send the default encoding to the Broker. It is assumed that you have read the introductory document Internationalization with EntireX and are familiar with the various internationalization approaches.

Parameters:

`enc` - the name of the character encoding or `LOCAL` or `null`.

Throws:

`BrokerException` - if setting the character encoding support fails.

Since:

7.1.1.24

---

getCharacterEncoding

```
public java.lang.String getCharacterEncoding()
```

Gets the character encoding name or `null`.

Returns:

the current character encoding name.

Since:

7.1.1.24

com.softwareag.entirex.aci

Class Conversation

```
java.lang.Object
  └ com.softwareag.entirex.aci.BrokerCommunication
      └ com.softwareag.entirex.aci.Conversation
```

```
public class Conversationextends BrokerCommunication
```

Represents a conversational communication with a participant.
When a new instance of a Conversation object is created, the first send or sendReceive call will establish a new conversation. The caller plays the role of a client in this conversation.
When a new instance of a Conversation object is created, the first receive call will wait for a new conversation established by a client. The caller plays the role of a server in this conversation.
An instance of a Conversation object is automatically created when a new conversation is received from the receive call of a BrokerService object.

See Also:

`BrokerService.receive(java.lang.String)`

Field Summary

Fields inherited from class com.softwareag.entirex.aci.BrokerCommunication
brokerService

Constructor Summary

Conversation(BrokerService brokerService) Creates a new Conversation object and attaches it to the specified BrokerService.
Conversation(BrokerService brokerService, ConversationState cstate) Creates a new Conversation object and attaches it to the specified BrokerService.

Method Summary

void	**cancel**() Cancels the current conversation.
void	**end**() Ends the current conversation.
void	**ignoreEOC**(boolean sendEOC) Per default the call to receive and receivePreview methods will return *null* for the 0003 0005 error (Partner finished the conversation) only.
BrokerMessage	**receive**() Receives an incoming request and waits for an answer.
BrokerMessage	**receive**(java.lang.String wait) Receives an incoming request and waits the specified time for an answer.
BrokerMessage	**receiveLast**() Re-reads the last message that was received for this conversation.
BrokerMessage	**receivePreview**() Receives an incoming request in preview mode and waits the default time for an answer.
BrokerMessage	**receivePreview**(java.lang.String wait) Receives an incoming request in preview mode and waits the specified time for an answer.
void	**send**(BrokerMessage msg) Sends an asynchronous conversational message.
BrokerMessage	**sendReceive**(BrokerMessage msg) Sends a synchronous conversational message.
BrokerMessage	**sendReceive**(BrokerMessage msg, java.lang.String wait) Sends a synchronous conversational message.

Methods inherited from class com.softwareag.entirex.aci.BrokerCommunication
dispose, getBrokerService, getUserData, saveState, setUserData

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

Conversation

```
public Conversation(BrokerService brokerService)
```

Creates a new Conversation object and attaches it to the specified BrokerService.
If the next method call on this object is a send or a sendReceive, this call will establish a new conversation.
If the next method call on this object is a receive, this call will wait until a request for a new conversation is issued by a client.

Parameters:

`brokerService` - an instance of a BrokerService object.

Conversation

```
public Conversation(BrokerService brokerService,
                    ConversationState cstate)
             throws java.lang.IllegalArgumentException
```

Creates a new Conversation object and attaches it to the specified BrokerService.
The conversation is restored from the ConversationState object.
If the next method call on this object is a send or a sendReceive, this call will establish a new conversation.
If the next method call on this object is a receive, this call will wait until a request for a new conversation is issued by a client.

Parameters:

`brokerService` - an instance of a BrokerService object.
`cstate` - the ConversationState object to be restored.

Throws:

`java.lang.IllegalArgumentException` - if ConversationState object is null.

Since:

7.1.1.10

See Also:

`ConversationState`

Method Detail

ignoreEOC

```
public void ignoreEOC(boolean sendEOC)
```

Per default the call to receive and receivePreview methods will return *null* for the 0003 0005 error (Partner finished the conversation) only.
Once the ignoreEOC method has been called all errors from the class 0003 will force the receive methods to return *null* instead of throwing a BrokerException. Optionally an EOC can be send to Broker (only if conversation still exists).

Parameters:

`sendEOC` - if true send automatically an EOC to Broker for this conversation

Since:
7.1.1.20

---

send

```
public void send(BrokerMessage msg)
          throws BrokerException
```

Sends an asynchronous conversational message.
Send the message to the Broker without waiting for an answer. Suitable for immediate, uni-directional messages.

Parameters:
`msg` - BrokerMessage to send.
Throws:
`BrokerException` - A Broker exception.

---

sendReceive

```
public BrokerMessage sendReceive(BrokerMessage msg,
                                 java.lang.String wait)
                          throws BrokerException
```

Sends a synchronous conversational message.
Sends the message to the Broker and waits the specified time to receive an answer. The answer is returned in the BrokerMessage object. Uses the maximum receive length.

Parameters:
`msg` - BrokerMessage to send.
`wait` - A timeout period, how long to wait for an immediate reply, measured in seconds/minutes/hours (depending on the trailing S/M/H character).
Returns:
the Broker message received.
Throws:
`BrokerException` - A Broker exception.
`java.lang.IllegalArgumentException` - if parameter is invalid.

---

sendReceive

```
public BrokerMessage sendReceive(BrokerMessage msg)
                          throws BrokerException
```

Sends a synchronous conversational message.
Sends the message to the Broker and waits to receive an answer. The answer is returned in the BrokerMessage object. Uses the maximum receive length and the default wait time of the service.

Parameters:
`msg` - the message to send.
Returns:
the Broker message received.

Throws:

`BrokerException` - A Broker exception.
`java.lang.IllegalArgumentException` - if parameter is invalid.

Since:

EntireX 5.2.1

---

end

```
public void end()
         throws BrokerException
```

Ends the current conversation. The partner can receive messages sent before the end of conversation was issued.
Create a new `Conversation` object for the next conversation after calling this method.

Throws:

`BrokerException` - A Broker exception.

---

cancel

```
public void cancel()
            throws BrokerException
```

Cancels the current conversation.
Create a new `Conversation` object for the next conversation after calling this method.

Throws:

`BrokerException` - A Broker exception.

---

receive

```
public BrokerMessage receive(java.lang.String wait)
                      throws BrokerException
```

Receives an incoming request and waits the specified time for an answer.
Depending on the state of the Conversation object this receive method will accept either a new conversation and receive the first message of this conversation or it will receive the next unprocessed message belonging to the current conversation.
The answer is returned in the BrokerMessage object. If the conversation has been terminated normally by the partner (via the end() method call), *null* is returned. Uses the maximum receive length.

Parameters:

`wait` - A timeout period, how long to wait for a receive, measured in seconds/minutes/hours (depending on the trailing S/M/H character). When null or "NO" is specified, the operation is non-blocked. If no message is available, a Broker Exception with errorclass=74 and errorcode=74 is thrown.

Returns:

A BrokerMessage object, or *null* if the conversation has been ended normally.

Throws:

`BrokerException` - A Broker exception.
`java.lang.IllegalArgumentException` - Thrown if parameter is invalid.

---

receive

```
public BrokerMessage receive()
                     throws BrokerException
```

Receives an incoming request and waits for an answer.
Depending on the state of the Conversation object this receive method will accept either a new conversation and receive the first message of this conversation or it will receive the next unprocessed message belonging to the current conversation.
The answer is returned in the BrokerMessage object. If the conversation has been terminated normally by the partner (via the end() method call), *null* is returned. Uses the maximum receive length and the default wait time from the service. If no message is available a Broker Exception with errorclass=74 and errorcode=74 is thrown.

Returns:

A BrokerMessage object, or *null* if the conversation has been ended normally.

Throws:

`BrokerException` - A Broker exception.

Since:

EntireX 5.2.1

---

receiveLast

```
public BrokerMessage receiveLast()
                         throws BrokerException
```

Re-reads the last message that was received for this conversation.
This method can be called as often as necessary.
The answer is returned in the BrokerMessage object. Uses the maximum receive length.

Returns:

A BrokerMessage object.

Throws:

`BrokerException` - A Broker exception.

---

receivePreview

```
public BrokerMessage receivePreview(java.lang.String wait)
                             throws BrokerException
```

Receives an incoming request in preview mode and waits the specified time for an answer.
Depending on the state of the Conversation object this receive method will accept either a new conversation and receive the first message of this conversation or it will receive the next unprocessed message belonging to the current conversation.
A subsequent receive method call will return this previewed message again.
The answer is returned in the BrokerMessage object. If the conversation has been terminated normally by the partner (via the end() method call) *null* is returned. Uses the maximum receive

length.

Parameters:

`wait` - A timeout period, how long to wait for a receive, measured in seconds/minutes/hours (depending on the trailing S/M/H character). When null or "NO" is specified, the operation is non-blocked. If no message is available, a Broker Exception with errorclass=74 and errorcode=74 is thrown.

Returns:

A BrokerMessage object, or `null` if the conversation has been ended normally.

Throws:

`BrokerException` - A Broker exception.
`java.lang.IllegalArgumentException` - Thrown if parameter is invalid.

---

receivePreview

```
public BrokerMessage receivePreview()
                            throws BrokerException
```

Receives an incoming request in preview mode and waits the default time for an answer. Depending on the state of the Conversation object this receive method will accept either a new conversation and receive the first message of this conversation or it will receive the next unprocessed message belonging to the current conversation.
A subsequent receive method call will return this previewed message again.
The answer is returned in the BrokerMessage object. If the conversation has been terminated normally by the partner (via the end() method call), `null` is returned. Uses the maximum receive length.

Returns:

A BrokerMessage object, or *null* if the conversation has been ended normally.

Throws:

`BrokerException` - A Broker exception.

---

com.softwareag.entirex.aci

Class ConversationState

```
java.lang.Object
  └ com.softwareag.entirex.aci.ConversationState
```

All Implemented Interfaces:

java.io.Serializable

```
public final class ConversationStateextends java.lang.Objectimplements java.io.Serializable
```

Class used to save the state of conversations. Can be used for both Conversation and UnitofWork objects. Objects of this class can be serialized and deserialized using standard Java serialization.
Objects can also be re-instantiated using a ticket string. The ticket is obtained with the getTicket() method. The object can be instantiated using the ticket with the restoreFromTicket() method.

Since:

7.1.1.10

See Also:

`BrokerCommunication.saveState()`, `Conversation.Conversation(BrokerService, ConversationState)`, `UnitofWork.UnitofWork(BrokerService, ConversationState)`, Serialized Form

Method Summary

`java.lang.String`	**`getTicket`**`()` Returns the ticket of this ConversationState object.
`static ConversationState`	**`restoreFromTicket`**`(java.lang.String ticket)` Create a ConversationState object from the specifed ticket.
`java.lang.String`	**`toString`**`()` Return the ticket of this ConversationState object.

Methods inherited from class java.lang.Object

`clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait`

Method Detail

restoreFromTicket

```
public static ConversationState restoreFromTicket(java.lang.String ticket)
                                           throws java.lang.IllegalArgumentException
```

Create a ConversationState object from the specifed ticket. The ticket has to be obtained by a call to getTicket().

Parameters:
`ticket` - the ticket
Returns:
the ConversationState object to the given ticket.
Throws:
`java.lang.IllegalArgumentException` - if ticket is not valid.
See Also:
`getTicket()`

---

toString

```
public java.lang.String toString()
```

Return the ticket of this ConversationState object.

Overrides:
`toString` in class `java.lang.Object`
Returns:
the ticket

---

getTicket

```
public java.lang.String getTicket()
```

Returns the ticket of this ConversationState object.

Returns:
the ticket

---

com.softwareag.entirex.aci

Class EntireXSecurity

```
java.lang.Object
  └ com.softwareag.entirex.aci.EntireXSecurity
```

All Implemented Interfaces:

BrokerSecurity

```
public class EntireXSecurityextends java.lang.Objectimplements BrokerSecurity
```

EntireX implementation of the Broker Security interface.
Never call the methods of this class directly, they are called automatically during logon processing.

See Also:

`BrokerSecurity`, `Broker.logon(java.lang.String)`

Constructor Summary

`EntireXSecurity``()`	

Method Summary

void	**decryptData**(byte[] data) Decrypts the received data in place.
void	**encryptData**(byte[] data) Encrypts the sent data in place.
byte[]	**getNewpassword**() Returns the encrypted Newpassword if it was supplied in the prepareLogon call.
byte[]	**getPassword**() Returns the encrypted password if it was supplied in the prepareLogon call.
byte[]	**getSecurityToken**() Returns a Security Token.
byte[]	**prepareAutoLogon**(byte[] securityToken)
void	**prepareLogon**(java.lang.String USERID, byte[] password, byte[] newpassword, byte[] securityToken) Encrypts the password(s) and generates a security token.

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

EntireXSecurity

```
public EntireXSecurity()
```

Method Detail

prepareLogon

```
public void prepareLogon(java.lang.String USERID,
                         byte[] password,
                         byte[] newpassword,
                         byte[] securityToken)
```

Description copied from interface: BrokerSecurity

Encrypts the password(s) and generates a security token.

This method is always called inside the Broker logon methods. It is called before performing a logon call to the Broker. This method prepares the encryption of the password and the new password and the generation of the security token. These values will be retrieved by the

getPassword, getNewpassword and getSecurityToken methods.

Specified by:
prepareLogon in interface BrokerSecurity
Parameters:
USERID - user ID from Broker constructor.
password - Password from logon method. May be null.
newpassword - New password from logon method. May be null.
securityToken - Security token from Broker object. May be null.
See Also:
Broker.logon(), Broker.logon(java.lang.String), Broker.logon(java.lang.String, java.lang.String), BrokerSecurity.getPassword(), BrokerSecurity.getNewpassword(), BrokerSecurity.getSecurityToken()

---

prepareAutoLogon

```
public byte[] prepareAutoLogon(byte[] securityToken)
```

---

getPassword

```
public byte[] getPassword()
```

Description copied from interface: BrokerSecurity
Returns the encrypted password if it was supplied in the prepareLogon call.

Specified by:
getPassword in interface BrokerSecurity
Returns:
The encrypted password or null.
See Also:
BrokerSecurity.prepareLogon(java.lang.String, byte[], byte[], byte[])

---

getNewpassword

```
public byte[] getNewpassword()
```

Description copied from interface: BrokerSecurity
Returns the encrypted Newpassword if it was supplied in the prepareLogon call.

Specified by:
getNewpassword in interface BrokerSecurity
Returns:
The encrypted new password or null.

See Also:
BrokerSecurity.prepareLogon(java.lang.String, byte[], byte[], byte[])

---

getSecurityToken

public byte[] **getSecurityToken**()

Description copied from interface: BrokerSecurity
Returns a Security Token.

Specified by:
getSecurityToken in interface BrokerSecurity
Returns:
The Security Token.
See Also:
BrokerSecurity.prepareLogon(java.lang.String, byte[], byte[], byte[])

---

encryptData

public void **encryptData**(byte[] data)

Description copied from interface: BrokerSecurity
Encrypts the sent data in place.

Specified by:
encryptData in interface BrokerSecurity
Parameters:
data - The send data.

---

decryptData

public void **decryptData**(byte[] data)

Description copied from interface: BrokerSecurity
Decrypts the received data in place.

Specified by:
decryptData in interface BrokerSecurity
Parameters:
data - The receive data.

---

com.softwareag.entirex.aci

Class LocationTransparencyService

```
java.lang.Object
  └ com.softwareag.entirex.aci.LocationTransparencyService
```

```
public final class LocationTransparencyServiceextends java.lang.Object
```

Connects to the LDAP directory via JNDI to retrieve Broker and BrokerService objects. Broker objects and BrokerService objects are configured with attributes in a directory. This locator service retrieves the objects with the help of com.softwareag.entirex.aci.ExxObjectFactory. Broker objects are named by logical Broker IDs. A broker object is returned with a real Broker ID and a dummy user ID, if the KERNELVERScall to this Broker was successful. BrokerService objects have, in addition to Broker objects, a server address in the form class/server/service.

This locator service is initialized implicitly with the file XDS.ini. Specify the location of this file in the system property ’entirex.location.transparency.ini’. Example: entirex.location.transparency.ini =C:\SoftwareAG\webMethods8\EntireX\config\XDS.ini.

An alternative is to specify the connection to the LDAP directory in a string. This string is read from the system property ’entirex.location.transparency.config’. It has the form ’ldap://<host>:<port>/<base DN> ?<key>=<value>&<key>=<value>&...’. The pairs key=value may contain any property for JNDI, i.e. ’java.naming.security.authentication=simple’. Two special keys ’AuthDN’ and ’AuthPass’ are mapped to ’java.naming.security.principal’ and ’java.naming.security.credentials’, respectively.

Only one of the system properties ’entirex.location.transparency.ini’ and ’entirex.location.transparency.config’ may be used. In both cases, "java.naming.factory.initial=com.sun.jndi.ldap.LdapCtxFactory" and "java.naming.factory.object=com.softwareag.entirex.aci.ExxObjectFactory" are included automatically.

This class is a Service Locator in the sense of the locator pattern of J2EE. For direct use of the JNDI lookup the following names for logical services or logical brokers have to be used. For a logical service use "sag-key=LogService=<logical service name>, sag-key=<set name>, sag-key=200, sag-key=LocTrans, sag-key=EntireX, sag-key=Software AG" below your base name. Here <logical service name> is the name of the logical service and <set name> is the name of the set of rules for location transparency. For a logical broker use "sag-key=LogBroker=<logical broker name>, sag-key=<set name>, sag-key=200, sag-key=LocTrans, sag-key=EntireX, sag-key=Software AG" below your base name. Here <logical broker name> is the name of the logical Broker and <set name> is the name of the set of rules for location transparency. Initialize the context for JNDI with properties including "java.naming.factory.initial=com.sun.jndi.ldap.LdapCtxFactory" and "java.naming.factory.object=com.softwareag.entirex.aci.ExxObjectFactory".

Since:

6.2.1

Method Summary	
static void	**init**(java.util.Properties jndiProperties) Initializes the JNDI directory context from the properties.
static void	**init**(java.lang.String jndiPropsFilename) Initializes the JNDI directory context.
static Broker	**lookupBroker**(java.lang.String logicalBrokerName) Retrieves the Broker bound to this logical Broker ID in the set 'DefaultSet'.
static Broker	**lookupBroker**(java.lang.String logicalBrokerName, java.lang.String locTransSetName) Retrieves the Broker bound to this logical Broker ID in the given set.
static BrokerService	**lookupBrokerService**(java.lang.String logicalServiceName) Retrieves the BrokerService bound to this logical service in the set 'DefaultSet'.
static BrokerService	**lookupBrokerService**(java.lang.String logicalServiceName, java.lang.String locTransSetName) Retrieves the BrokerService bound to this logical service in the given set.

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Method Detail

lookupBrokerService

```
public static BrokerService lookupBrokerService(java.lang.String logicalServiceName)
                                         throws BrokerException
```

Retrieves the BrokerService bound to this logical service in the set 'DefaultSet'.

Parameters:

`logicalServiceName` - the name of the logical service.

Returns:

`BrokerService` bound to this logical service in the set 'DefaultSet'.

Throws:

`BrokerException` - All Exceptions are wrapped into BrokerExceptions.

lookupBrokerService

```
public static BrokerService lookupBrokerService(java.lang.String logicalServiceName,
                                                 java.lang.String locTransSetName)
                                          throws BrokerException
```

Retrieves the BrokerService bound to this logical service in the given set.

Parameters:

`logicalServiceName` - the name of the logical service.
`locTransSetName` - the name of the set of rules for Location Transparency. If the name is `null` or has length `0`, the name is set to 'DefaultSet'.

Returns:

`BrokerService` bound to this logical service in the given set.

Throws:

`BrokerException` - All Exceptions are wrapped into BrokerExceptions.

lookupBroker

```
public static Broker lookupBroker(java.lang.String logicalBrokerName)
                           throws BrokerException
```

Retrieves the Broker bound to this logical Broker ID in the set 'DefaultSet'.

Parameters:

`logicalBrokerName` - the name of the logical Broker.

Returns:

`Broker` bound to this logical Broker in the set 'DefaultSet'.

Throws:

`BrokerException` - All Exceptions are wrapped into BrokerExceptions.

lookupBroker

```
public static Broker lookupBroker(java.lang.String logicalBrokerName,
                                  java.lang.String locTransSetName)
                           throws BrokerException
```

Retrieves the Broker bound to this logical Broker ID in the given set.

Parameters:

`logicalBrokerName` - the name of the logical Broker.
`locTransSetName` - the name of the set of rules for Location Transparency. If the name is `null` or has length `0`, the name is set to 'DefaultSet'.

Returns:

`Broker` bound to this logical Broker in the given set.

Throws:

`BrokerException` - All Exceptions are wrapped into BrokerExceptions.

init

```
public static void init(java.lang.String jndiPropsFilename)
                throws BrokerException
```

Initializes the JNDI directory context. Use this only if the configuration with ’XDS.ini’ is not possible.

Parameters:

`jndiPropsFilename` - the name of the file with the JNDI properties. If `null` is used, ’JNDI.properties’ is read.

Throws:

`BrokerException` - on errors in configuration.

---

init

```
public static void init(java.util.Properties jndiProperties)
                throws BrokerException
```

Initializes the JNDI directory context from the properties. Use this only if the configuration with ’XDS.ini’ is not possible, i.e. from an applet.

Parameters:

`jndiProperties` - the properties object with the JNDI properties.

Throws:

`BrokerException` - on errors in configuration.

---

com.softwareag.entirex.aci
Interface MessageListener

```
public interface MessageListener
```

Interface used by the `PublicationListener`. Enables the application to handle the received messages.

Since:
7.2.1

Method Summary

void	**onMessage**(BrokerMessage msg) Process the received message.

Method Detail

onMessage

```
void onMessage(BrokerMessage msg)
```

Process the received message. The application has to implement this method and set this `MessageListener` in the constructor of the `PublicationListener`.

Parameters:
`msg` - the received `BrokerMessage`.

com.softwareag.entirex.aci

Class Publication

```
java.lang.Object
  └ com.softwareag.entirex.aci.Publication
```

```
public class Publicationextends java.lang.Object
```

This class is the Java ACI for the Broker Publish & Subscribe ACI. The class holds a publication of the Broker, published or received. The publication can hold one or more messages. The messages are `BrokerMessage` objects. The publication is closed with `commit` and discarded with `backout`.

Since:

7.2.1

Field Summary

static java.lang.String	**MSG_FIRST** A publication status.
static java.lang.String	**MSG_LAST** A publication status.
static java.lang.String	**MSG_MIDDLE** A publication status.
static java.lang.String	**MSG_ONLY** A publication status.

Constructor Summary

Publication(Broker broker, java.lang.String topicName) Create a publication with a given `Broker` and a topic name.	

Method Summary	
void	**backout**() Backs out this publication.
void	**commit**() Commits this publication.
java.lang.String	**getPublicationId**() Gets the publication ID.
java.lang.String	**getPublicationStatus**() Gets the status of the publication.
int	**getReceiveLength**() Gets the recieve length.
java.lang.String	**getUserStatus**() Gets the user status field of this publication.
java.lang.String	**last**() Gets the status of the last publication of this user.
void	**publish**(BrokerMessage message) Publishes a message within the topic.
java.lang.String	**query**() Gets the status of the publication given by the publication id.
BrokerMessage	**receive**() Receive a message within the topic with no wait time.
BrokerMessage	**receive**(java.lang.String wait) Receive a message within the topic with a wait time.
void	**setPublicationId**(java.lang.String publicationId) Sets the publication ID.
void	**setReceiveLength**(int receiveLength) Sets the recieve length.
void	**setUserStatus**(java.lang.String userStatus) Sets the user status field of this publication.
void	**subscribe**(boolean durable) Subscribes to the topic of this publication.
void	**unsubscribe**() Unsubscribes this subscriber from the topic.

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

MSG_ONLY

```
public static final java.lang.String MSG_ONLY
```

A publication status. This status is set if the message is the only message in a publication. No further calls of `receive` are allowed.

See Also:
Constant Field Values

MSG_LAST

```
public static final java.lang.String MSG_LAST
```

A publication status. This status is set if the message is the last message in a publication. No further calls of `receive` are allowed.

See Also:
Constant Field Values

MSG_FIRST

```
public static final java.lang.String MSG_FIRST
```

A publication status. This status is set if the message is the first message in a publication.

See Also:
Constant Field Values

MSG_MIDDLE

```
public static final java.lang.String MSG_MIDDLE
```

A publication status. This status is set if the message is not the first and not the last message in a publication, but 'in the middle' of a publication.

See Also:
Constant Field Values

Constructor Detail

Publication

```
public Publication(Broker broker,
                   java.lang.String topicName)
```

Create a publication with a given `Broker` and a topic name.

Parameters:

`broker` - the `Broker`. Must not equal `null`.
`topicName` - the name of the topic.

Throws:

`java.lang.IllegalArgumentException` - if the `Broker` is `null`.

Method Detail

publish

```
public void publish(BrokerMessage message)
             throws BrokerException
```

Publishes a message within the topic.

Parameters:

`message` - the `BrokerMessage` to publish.

Throws:

`BrokerException` - if the `Broker` returns an error or the publication ID is corrupt.
`java.lang.IllegalArgumentException` - if the message is `null`.

---

receive

```
public BrokerMessage receive()
                     throws BrokerException
```

Receive a message within the topic with no wait time. After receive check the status of the publication with `getPublicationStatus`. If the status is `MSG_ONLY` or `MSG_LAST`, `commit` or `backout` the publication and create a new `Publication` object before receiving the next message. The `PublicationListen` does this checking and always commits the publication.

Returns:

the received message.

Throws:

`BrokerException` - if the `Broker` returns an error or the publication ID is corrupt.

See Also:

`getPublicationStatus()`, `PublicationListener`, `receive(java.lang.String)`

---

receive

```
public BrokerMessage receive(java.lang.String wait)
                     throws BrokerException
```

Receive a message within the topic with a wait time. After receive check the status of the publication with `getPublicationStatus`. If the status is `MSG_ONLY` or `MSG_LAST`, `commit` or `backout` the publication and create a new `Publication` object before receiving the next message. The `PublicationListen` does this checking and always commits the publication.

Parameters:

`wait` - the timeout period, how long to wait for a message, measured in seconds/minutes/hours (depending on the trailing S/M/H character). When `null` or "NO" is specified, the operation is non-blocked. If no message is available, a `BrokerException` with errorclass=74 and errorcode=74 is thrown.

Returns:

the received message.

Throws:

`BrokerException` - if the Broker returns an error or the publication ID is corrupt or a wait time out occurs.

See Also:

`getPublicationStatus()`, `PublicationListener`, `receive()`

---

subscribe

```
public void subscribe(boolean durable)
               throws BrokerException
```

Subscribes to the topic of this publication. This method uses the user ID and the token of the `Broker` object as a subscriber name. Only the subscriber uses this method.

Parameters:

`durable` - if `true`, the subscription is stored in the Broker and it is valid until `unsubscribe`, even if this subscriber is not active.

Throws:

`BrokerException` - if the `Broker` returns an error.

---

unsubscribe

```
public void unsubscribe()
                throws BrokerException
```

Unsubscribes this subscriber from the topic. The subscriber is identified by user ID and token in the `logon` call. Only the subscriber uses this method.

Throws:

`BrokerException` - if the `Broker` returns an error.

---

commit

```
public void commit()
            throws BrokerException
```

Commits this publication.

If a publisher commits a publication, this is visible for the subscribers and they can receive the publication. Once a publisher commits a publication, no more messages can be added.

If the subscriber commits a publication, it finished receiving messages of this publication. Afterwards, this subscriber cannot receive any unread message from this publication.

Throws:

`BrokerException` - if the `Broker` returns an error.

---

backout

```
public void backout()
             throws BrokerException
```

Backs out this publication.

If a publisher backs out a publication, all messages in this publication are deleted. Create a new `Publication` object before publishing new messages.

If a subscriber backs out a publication it can start to recieve the first and all following messages again.

Throws:

`BrokerException` - if the `Broker` returns an error.

---

last

```
public java.lang.String last()
                     throws BrokerException
```

Gets the status of the last publication of this user. Only used by the publisher. The search for the last publication considers only publications to topics where durable subscriptions are allowed.

Returns:

the status of the last publication.

Throws:

`BrokerException` - if the `Broker` returns an error.

---

getPublicationStatus

```
public java.lang.String getPublicationStatus()
```

Gets the status of the publication.

Returns:
the status of the publication.
See Also:
`MSG_ONLY`, `MSG_FIRST`, `MSG_MIDDLE`, `MSG_LAST`

---

query

```
public java.lang.String query()
                      throws BrokerException
```

Gets the status of the publication given by the publication id. Only used by the publisher.

Returns:
the status of the publication.
Throws:
`BrokerException` - if the `Broker` returns an error.

---

setUserStatus

```
public void setUserStatus(java.lang.String userStatus)
                   throws BrokerException
```

Sets the user status field of this publication. Publisher and subscriber can use this method.

Parameters:
`userStatus` - the user status.
Throws:
`BrokerException` - if the `Broker` returns an error.

---

getUserStatus

```
public java.lang.String getUserStatus()
```

Gets the user status field of this publication. Publisher and subscriber can use this method.

Returns:
the user status.
Since:
7.2.1.35

---

getReceiveLength

```
public int getReceiveLength()
```

Gets the recieve length. Only used by the subscriber.

Returns:
the receive length in bytes.

setReceiveLength

```
public void setReceiveLength(int receiveLength)
```

Sets the recieve length. The recieve length is the maximal length, a received message can have. Only used by the subscriber.

Parameters:

`receiveLength` - the receive length given in bytes.

getPublicationId

```
public java.lang.String getPublicationId()
```

Gets the publication ID. The publication ID can be stored by the application. With this ID the application can publish or receive further messages for this publication.

Returns:

the publication ID.

setPublicationId

```
public void setPublicationId(java.lang.String publicationId)
```

Sets the publication ID. Use this modifier to publish or receive messages for an existing publication. Set the ID before `publish` or `receive`.

Parameters:

`publicationId` - the publication ID.

Overview Package Class Tree Deprecated Index Help

PREV CLASS NEXT CLASS **FRAMES NO FRAMES All Classes**

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

com.softwareag.entirex.aci

Class PublicationListener

```
java.lang.Object
  └ java.lang.Thread
      └ com.softwareag.entirex.aci.PublicationListener
```

All Implemented Interfaces:

java.lang.Runnable

```
public class PublicationListenerextends java.lang.Thread
```

A simple listener for publications. This listener starts receiving after calling `start`. To stop the listener, use `stopListener`. The listener stops after a wait time of 10 seconds. If the listener receives the last message of a publication, it commits this publication and starts receiving messages with a new publication.

Since:

7.2.1

Nested Class Summary

Nested classes/interfaces inherited from class java.lang.Thread
java.lang.Thread.State, java.lang.Thread.UncaughtExceptionHandler

Field Summary

Fields inherited from class java.lang.Thread
MAX_PRIORITY, MIN_PRIORITY, NORM_PRIORITY

Constructor Summary

PublicationListener(Broker broker, java.lang.String topicName, java.lang.String wait, MessageListener listener) Creates a PublicationListener with a MessageListener.

Method Summary

BrokerException	**getLastException**() Gets the last BrokerException which occurred in the run method of this listener.
void	**run**() Receives messages in publications.
void	**stopListener**() Stop the listener.

Methods inherited from class java.lang.Thread

activeCount, checkAccess, countStackFrames, currentThread, destroy, dumpStack, enumerate, getAllStackTraces, getContextClassLoader, getDefaultUncaughtExceptionHandler, getId, getName, getPriority, getStackTrace, getState, getThreadGroup, getUncaughtExceptionHandler, holdsLock, interrupt, interrupted, isAlive, isDaemon, isInterrupted, join, join, join, resume, setContextClassLoader, setDaemon, setDefaultUncaughtExceptionHandler, setName, setPriority, setUncaughtExceptionHandler, sleep, sleep, start, stop, stop, suspend, toString, yield

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Constructor Detail

PublicationListener

```
public PublicationListener(Broker broker,
                           java.lang.String topicName,
                           java.lang.String wait,
                           MessageListener listener)
```

Creates a `PublicationListener` with a `MessageListener`.

Parameters:

`broker` - the `Broker` object.
`topicName` - the name of the topic.
`wait` - the wait time for the receive calls.
`listener` - the object implementing the `onMessage` method.

Method Detail

run

```
public void run()
```

Receives messages in publications. If the last message in a publication is received, the current publication is committed and a new publication is received. The method returns if it is stopped with `stopListener` or if a `BrokerException` occurs.

Specified by:

`run` in interface `java.lang.Runnable`

Overrides:

`run` in class `java.lang.Thread`

---

stopListener

```
public void stopListener()
```

Stop the listener. The listener waits for the current receive call to return and then it stops.

---

getLastException

```
public BrokerException getLastException()
```

Gets the last BrokerException which occurred in the `run` method of this listener.

Returns:

the latest `BrokerException`.

---

com.softwareag.entirex.aci

Class RPCService

```
java.lang.Object
  └ com.softwareag.entirex.aci.BrokerService
      └ com.softwareag.entirex.aci.RPCService
```

Direct Known Subclasses:

XMLRPCService

```
public abstract class RPCServiceextends BrokerService
```

This abstract subclass of BrokerService represents a Broker service used by EntireX RPC. The client stub and server stub generated by the Java Wrapper are subclasses of RPCService.
You should not call the methods of the superclass BrokerService directly.

Since:

EntireX 5.2.1

Field Summary

static int	**RELIABLE_AUTO_COMMIT** RELIABLE_AUTO_COMMIT = 1
static int	**RELIABLE_CLIENT_COMMIT** RELIABLE_CLIENT_COMMIT = 2
static int	**RELIABLE_OFF** RELIABLE_OFF = 0

Fields inherited from class com.softwareag.entirex.aci.BrokerService
DEFAULT_WAITTIME

Constructor Summary

protected	**RPCService**() Creates an RPCService object without parameters.
protected	**RPCService**(BrokerService brokerService, java.lang.String libName, boolean compress) Creates an RPCService object.
protected	**RPCService**(Broker broker, java.lang.String serverAddr, java.lang.String libName) Creates an RPCService object.
protected	**RPCService**(Broker broker, java.lang.String serverAddr, java.lang.String libName, boolean compress) Creates an RPCService object.

Method Summary

void	**closeConversation**() Closes the running RPC conversation.
void	**closeConversationCommit**() Closes the running RPC conversation.
boolean	**getCompression**() Returns the current setting for compression.
protected Conversation	**getConversation**() Returns the Conversation object.
java.lang.String	**getLibraryName**() Returns the current value of the library name used by the RPC.
java.lang.String	**getMessageID**() Gets the message id (valid for reliable RPC).
boolean	**getNaturalLogon**() Returns the current setting for logon to Natural Security for Natural RPC servers.
java.lang.String	**getProgramName**() Returns the current value of the RPC subprogram name.
int	**getReliable**() Gets the mode for reliable RPC.
java.lang.String	**getRPCPassword**() Gets the RPC password (used with NATURAL logon).
java.lang.String	**getRPCUserId**() Returns the user ID which is used by the RPCs.

java.lang.String	**getStatusOfMessage**(java.lang.String messageID) Gets the status of the message identified by the message id (valid for reliable RPC).
protected void	**onEnter**(java.lang.String progname) User exit method called at the beginning of a generated method.
protected void	**onException**(java.lang.String progname, BrokerException exception) User exit method called when an exception which is an instance of BrokerException is thrown in the generated method.
protected void	**onLeave**(java.lang.String progname, int sendLength, int receiveLength) User exit method called at the end of a generated method.
protected boolean	**onRetry**(java.lang.String progname, BrokerException exception) User exit method called when an exception which is an instance of BrokerException is thrown in the generated method.
java.lang.String	**ping**() Sends an RPC PING command to the service and returns the response string.
void	**reliableCommit**() Commit a transaction (unit of work) for reliable RPC.
void	**reliableRollback**() Roll back a transaction (unit of work) for reliable RPC.
void	**setBroker**(Broker broker) Dynamically assigns the instance of a Broker object.
void	**setCompression**(boolean rpcCompression) Switches RPC compression ON or OFF.
void	**setConversation**(Conversation conversation) Enables conversational RPC.
void	**setLibraryName**(java.lang.String libName) Changes the library name used by the RPC.
void	**setNaturalLogon**(boolean logon) Enables or disables logon to Natural Security for Natural RPC servers.
void	**setReliable**(int mode) Sets reliable RPC mode.
void	**setRpcLibrary**(java.lang.String libraryName) Sets the RPC library name which is send to the broker with the RPCLIB keyword.
void	**setRPCPassword**(java.lang.String password) Changes the password used for an RPC.
void	**setRpcProgram**(java.lang.String programName) Sets the RPC program name which is send to the broker with the RPCPGM keyword.

void	**setRPCUserId**(java.lang.String userId) Changes the user ID used for an RPC call.
void	**setServerAddress**(java.lang.String serverAddr) Dynamically assigns the server address.

Methods inherited from class com.softwareag.entirex.aci.BrokerService

cancelallConversations, deregister, deregisterImmediate, endallConversations, getBroker, getCharacterEncoding, getDefaultWaittime, getEnvironment, getMaxReceiveLen, getServerClass, getServerName, getServiceName, isGeneric, receive, receive, receiveAny, receiveAttachInfo, receiveOld, register, registerAttach, replyError, send, sendReceive, sendReceive, setAdjustReceiveLen, setDefaultWaittime, setEnvironment, setLogicalBroker, setLogicalBroker, setLogicalBroker, setLogicalService, setLogicalService, setMaxReceiveLen, toString, useCodePage, useCodePage

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Field Detail

RELIABLE_OFF

```
public static final int RELIABLE_OFF
```

RELIABLE_OFF = 0

See Also:
Constant Field Values

RELIABLE_AUTO_COMMIT

```
public static final int RELIABLE_AUTO_COMMIT
```

RELIABLE_AUTO_COMMIT = 1

See Also:
Constant Field Values

RELIABLE_CLIENT_COMMIT

```
public static final int RELIABLE_CLIENT_COMMIT
```

RELIABLE_CLIENT_COMMIT = 2

See Also:

Constant Field Values

Constructor Detail

RPCService

```
protected RPCService(Broker broker,
                     java.lang.String serverAddr,
                     java.lang.String libName,
                     boolean compress)
```

Creates an RPCService object.

Parameters:

`broker` - A Broker instance.
`serverAddr` - The server address (class/name/service).
`libName` - The default library name.
`compress` - Compression ON/OFF.

RPCService

```
protected RPCService(BrokerService brokerService,
                     java.lang.String libName,
                     boolean compress)
```

Creates an RPCService object.

Parameters:

`brokerService` - A BrokerService instance.
`libName` - The default library name.
`compress` - Compression ON/OFF.

RPCService

```
protected RPCService(Broker broker,
                     java.lang.String serverAddr,
                     java.lang.String libName)
```

Creates an RPCService object.

Parameters:

`broker` - A Broker instance.
`serverAddr` - The server address (class/name/service).
`libName` - The default library name.

RPCService

```
protected RPCService()
```

Creates an RPCService object without parameters.

Method Detail

setRPCUserId

```
public final void setRPCUserId(java.lang.String userId)
```

Changes the user ID used for an RPC call. Default is the user ID specified in the Broker constructor. At the moment this is only used by Natural RPC servers running with Natural Security.

Parameters:

`userId` - The new user ID.

Since:

5.2.1.7

---

getRPCUserId

```
public final java.lang.String getRPCUserId()
```

Returns the user ID which is used by the RPCs.

Returns:

user ID as string.

Since:

5.2.1.7

---

setRPCPassword

```
public final void setRPCPassword(java.lang.String password)
```

Changes the password used for an RPC. Default is the password specified in the Broker `logon` method. At the moment this is only used by Natural RPC servers running with Natural Security.

Parameters:

`password` - The new Password.

Since:

5.2.1.7

---

getRPCPassword

```
public final java.lang.String getRPCPassword()
```

Gets the RPC password (used with NATURAL logon).

Returns:
the RPC password.

---

setBroker

```
public final void setBroker(Broker broker)
                     throws BrokerException
```

Dynamically assigns the instance of a Broker object. The method cannot be called during a conversational RPC sequence. Can be used by Java Wrapper Customization classes.

Parameters:
`broker` - A Broker instance.
Throws:
`BrokerException` - A BrokerException.
Since:
5.3.1.2

---

setServerAddress

```
public final void setServerAddress(java.lang.String serverAddr)
                            throws BrokerException
```

Dynamically assigns the server address. The method cannot be called during a conversational RPC sequence. Can be used by Java Wrapper Customization classes.

Parameters:
`serverAddr` - The server address (class/name/service).
Throws:
`BrokerException` - A BrokerException.
Since:
5.3.1.2

---

setLibraryName

```
public final void setLibraryName(java.lang.String libName)
```

Changes the library name used by the RPC.

Parameters:
`libName` - The new library name (maximum 8 characters).

---

getLibraryName

```
public java.lang.String getLibraryName()
```

Returns the current value of the library name used by the RPC.

Returns:
The current library name as a string.

---

getProgramName

```
public java.lang.String getProgramName()
```

Returns the current value of the RPC subprogram name.

Returns:
The current subprogram name as a String.

---

setNaturalLogon

```
public final void setNaturalLogon(boolean logon)
```

Enables or disables logon to Natural Security for Natural RPC servers.

Parameters:
`logon` - `true` to enable, `false` to disable.

---

getNaturalLogon

```
public final boolean getNaturalLogon()
```

Returns the current setting for logon to Natural Security for Natural RPC servers.

Returns:
`true` if enabled, `false` if disabled.

---

setCompression

```
public final void setCompression(boolean rpcCompression)
```

Switches RPC compression ON or OFF.

Parameters:
`rpcCompression` - `true` if ON, `false` if OFF.

---

getCompression

```
public final boolean getCompression()
```

Returns the current setting for compression.

Returns:
`true` if ON, `false` if OFF.

---

setRpcProgram

```
public final void setRpcProgram(java.lang.String programName)
```

Sets the RPC program name which is send to the broker with the RPCPGM keyword.

Parameters:
`programName` - the program name.

---

setRpcLibrary

```
public final void setRpcLibrary(java.lang.String libraryName)
```

Sets the RPC library name which is send to the broker with the RPCLIB keyword.

Parameters:
`libraryName` - the library name.

---

setConversation

```
public final void setConversation(Conversation conversation)
```

Enables conversational RPC. All RPCs going through this instance of the RPCService object will use the Conversation object passed as parameter. The same instance of a Conversation object can be passed to different instances of an RPCService object. They will all run in the same conversation.

Parameters:
`conversation` - A non-null Conversation object.
Throws:
`java.lang.IllegalArgumentException` - Thrown if no conversation is specified.
`java.lang.IllegalStateException` - if this service is already used for reliable RPC.

---

getConversation

```
protected final Conversation getConversation()
```

Returns the Conversation object.

Returns:
Conversation object or null;

closeConversation

```
public final void closeConversation()
                           throws BrokerException
```

Closes the running RPC conversation. The RPC server receives a "backout" notification.

Throws:

`BrokerException` - A BrokerException.

closeConversationCommit

```
public final void closeConversationCommit()
                                 throws BrokerException
```

Closes the running RPC conversation. The RPC server receives a "commit" notification.

Throws:

`BrokerException` - A BrokerException.

reliableCommit

```
public final void reliableCommit()
                         throws BrokerException
```

Commit a transaction (unit of work) for reliable RPC.

Throws:

`BrokerException` - if the broker call to commit the messages fails.

Since:

8.0

reliableRollback

```
public final void reliableRollback()
                           throws BrokerException
```

Roll back a transaction (unit of work) for reliable RPC.

Throws:

`BrokerException` - if the broker call to commit the messages fails.

Since:

8.0

getMessageID

```
public final java.lang.String getMessageID()
```

Gets the message id (valid for reliable RPC). The message id changes after `reliableCommit()` or `reliableRollback()` in `RELIABLE_CLIENT_COMMIT` mode and after sending a message in `RELIABLE_AUTO_COMMIT` mode.

Returns:
the message id.
Since:
8.0

---

getStatusOfMessage

```
public final java.lang.String getStatusOfMessage(java.lang.String messageID)
                                          throws BrokerException
```

Gets the status of the message identified by the message id (valid for reliable RPC). Due to Broker settings the status may be not available after processing the message.

Parameters:
`messageID` - the message id obtained by a previous call of `getMessageID()`.
Returns:
the status of the message.
Throws:
`BrokerException` - if the Broker call fails.
Since:
8.0

---

onEnter

```
protected void onEnter(java.lang.String progname)
                throws BrokerException
```

User exit method called at the beginning of a generated method. This method has a default implementation and can be overwritten in the Java Wrapper Customization class.

Parameters:
`progname` - The RPC program name.
Throws:
`BrokerException` - A BrokerException.
Since:
5.3.1.2

---

onLeave

```
protected void onLeave(java.lang.String progname,
                       int sendLength,
                       int receiveLength)
                throws BrokerException
```

User exit method called at the end of a generated method. This method is only called when no exception is thrown. This method has a default implementation and can be overwritten in the Java Wrapper Customization class.

Parameters:

`progname` - The RPC program name.
`sendLength` - length of send buffer
`receiveLength` - length of receive buffer

Throws:

`BrokerException` - A BrokerException.

Since:

5.3.1.2

---

onException

```
protected void onException(java.lang.String progname,
                           BrokerException exception)
                    throws BrokerException
```

User exit method called when an exception which is an instance of `BrokerException` is thrown in the generated method. After calling this method the exception is thrown again. There is no need to throw the exception in the implementation of this method. This method has a default implementation and can be overriden in the Java Wrapper Customization class.

Parameters:

`progname` - The RPC program name.
`exception` - reference to the exception, which is thrown in the generated method.

Throws:

`BrokerException` - A BrokerException.

Since:

5.3.1.2

---

onRetry

```
protected boolean onRetry(java.lang.String progname,
                          BrokerException exception)
                   throws BrokerException
```

User exit method called when an exception which is an instance of `BrokerException` is thrown in the generated method. This method is only called when the exception is thrown during the processing of the RPC. If this method returns `false`, the exception will be thrown again and the `onException` method is called. If this method returns `true`, the RPC will be executed once again. If the second RPC fails, `onException` will be called immediately. This method has a default implementation and can be overridden in the Java Wrapper Customization class.

Parameters:

`progname` - The RPC program name.
`exception` - reference to the exception which is thrown in the generated method.

Returns:

false as a default implementation.

Throws:

`BrokerException` - A BrokerException.

Since:

5.3.1.2

ping

```
public java.lang.String ping()
                     throws BrokerException
```

Sends an RPC PING command to the service and returns the response string.

Returns:
the response string to the PING command.
Throws:
`BrokerException` - if a Broker error occurs.
Since:
7.1.1.24

getReliable

```
public final int getReliable()
```

Gets the mode for reliable RPC. Allowed values are `RELIABLE_OFF`, `RELIABLE_AUTO_COMMIT`, `RELIABLE_CLIENT_COMMIT`.

Returns:
the mode for reliable RPC
Since:
8.0

setReliable

```
public final void setReliable(int mode)
```

Sets reliable RPC mode. Allowed values are `RELIABLE_OFF`, `RELIABLE_AUTO_COMMIT`, `RELIABLE_CLIENT_COMMIT`.

Parameters:
`mode` - the reliable RPC mode to set
Throws:
`java.lang.IllegalStateException` - if this service is already used for reliable RPC.
Since:
8.0

com.softwareag.entirex.aci
Interface ServerImplementation

```
public interface ServerImplementation
```

This interface may be implemented by server classes which implement the programs of a library. Second, this interface can be used to implement custom methods for initialization and shutdown of the server. These methods can deal with connections to other components like databases.

Since:

7.1.1.50

Method Summary

void	**closeConversation**(boolean commit) Method called by the RPC Server when a conversation RPC ends.
void	**finish**() Called on termination of a worker thread.
void	**init**() Called on start by the Java RPC server.
void	**shutdown**() Called on shutdown by the Java RPC server.

Method Detail

closeConversation

```
void closeConversation(boolean commit)
```

Method called by the RPC Server when a conversation RPC ends.
The paramter indicates that the RPC client has closed the conversation with the option commit. Otherwise the commit option has not been specified by the RPC client or the conversation has been terminated abnormally.
Implement this method in your server class.

Parameters:

`commit` - `true` if closeConversationCommit has been called by RPC client, `false` otherwise

Since:

EntireX 7.1.1.50

---

init

```
void init()
          throws java.lang.Exception
```

Called on start by the Java RPC server. Create a class implementing this interface. The server reads the name of the class from the configuration, instantiates the class and calls this method on start-up. If this method throws an exception, the stack trace is written to the log file or to `System.out`. In this case the server does not start.
The invocation of this method is not synchronized.

Throws:

`java.lang.Exception` - if the method fails.

---

shutdown

```
void shutdown()
               throws java.lang.Exception
```

Called on shutdown by the Java RPC server. Create a class implementing this interface. The server uses the same instance as for the `init()` and calls this method on shutdown. If this method throws an exception, the stack trace is written to the log file or to `System.out`. In this case the shutdown proceeds.
The invocation of this method is not synchronized. If the `init` starts any threads, `shutdown` must stop these threads. Otherwise, the Java RPC server will not terminate.

Throws:

`java.lang.Exception` - if the method fails.

---

finish

```
void finish()
             throws java.lang.Exception
```

Called on termination of a worker thread. If a server class implements this interface, this method is called when a worker thread of the Java RPC server terminates. If this method throws an exception, the stack trace is written to the log file or to `System.out`. This method is called once for each worker thread and each server class of this worker thread.

Throws:

`java.lang.Exception` - if the method fails.

---

com.softwareag.entirex.aci
Interface ThreadRunner

```
public interface ThreadRunner
```

The `ThreadRunner` interface wraps starting a thread. This is used in an application server to use the method of the application server to start a new thread.

Method Summary

void	**startThread**(java.lang.Thread t) Start a thread.

Method Detail

startThread

```
void startThread(java.lang.Thread t)
                 throws java.lang.Exception
```

Start a thread.

Parameters:

`t` - the thread to start.

Throws:

`java.lang.Exception` - if starting the thread fails.

com.softwareag.entirex.aci

Class UnitofWork

```
java.lang.Object
  └ com.softwareag.entirex.aci.BrokerCommunication
      └ com.softwareag.entirex.aci.UnitofWork
```

```
public class UnitofWorkextends BrokerCommunication
```

Represents a UnitofWork communication.
A unit of work is a set of one or more messages that are processed as a single unit. The sender of a unit of work adds messages to the unit of work and then indicates that it is complete by calling the commit or sendCommit method. The unit of work and its messages are not visible to the receiver until the sender has committed the unit of work. Once it is committed, the receiver can receive messages and can indicate when it is complete by calling the `commit` or `commitBoth` method.

When a unit of work has been committed, the application can either create a new unit of work (by calling the send method) or receive a new unit of work (by calling the receive method), using the same UnitofWork object.

Each unit of work runs implicitly in a conversation. This conversation is handled automatically by the UnitofWork object without using the Conversation object. The conversation spans the lifetime of a UnitofWork object and can be explicitly terminated by the `commitCancelConversation`, `commitEndConversation` or `endConversation` method. Example for a sender: send `numberConversations` conversations, each with `numberUows` units of work, containing `numberMessages` messages each. Thus, `numberConversations * numberUows * numberMessages` messages are send in total.

```
Broker b = new Broker("my broker id", "my user id");
BrokerService service = new BrokerService(b, "ACLASS/ASERVER/ASERVICE");
BrokerMessage msg = new BrokerMessage();
for (int i = 0; i < numberConversations; i++) {
       ConversationState state = null;
       for (int j = 0; j < numberUows; j++) {
              UnitofWork uow = (j == 0) ? new UnitofWork(service) : new UnitofWork(service, state);
              for (int k = 0; k < numberMessages; k++) {
                     msg.setMessage("Message " + k + " in UOW " + uow.getUnitofWorkID() + " (" + j + ") in " + ((j>0) ? state.getTicket() : "(no ticket)") + " (" + i + ").");
                     uow.send(msg);
              }
              if (j + 1 == numberUows)
                     uow.commitEndConversation();
              else
                     uow.commit();
              state = uow.saveState();
       }
}
```

Example for a receiver: receive all available messages in all units of work over multiple conversations.

```
    Broker b = new Broker("my broker id", "my user id");
    BrokerService service = new BrokerService(b, "ACLASS/ASERVER/ASERVICE");
    service.register();
    String wait = "10S";
    boolean nextMessage = true;
    BrokerMessage msg = new BrokerMessage();
    UnitofWork uow = null;
    try {
           // loop over the conversations
           while (true) {
                  try {
                         ConversationState state = null;
                         // loop over the units of work in one conversation.
                         while (true) {
```

```
                                uow = (state == null) ? new UnitofWork(service) : new UnitofWork(service, state);
                                // loop over all messages in one unit of work.
                                while (nextMessage) {
                                        msg = uow.receive(wait);
                                        if (msg != null) {
                                                // do something with the message.
                                        }
                                        nextMessage = !(uow.getStatus().equals("RECV_ONLY") || uow.getStatus().equals("RECV_LAST"));
                                }
                                nextMessage = true;
                                uow.commit();
                                state = uow.saveState();
                        }
                }
                catch (BrokerException bEx) {
                        if (bEx.getErrorClass() == 3 && bEx.getErrorCode() == 5) {
                                // conversation has ended, try next conversation.
                        } else {
                                throw bEx;
                        }
                }
        }
}
catch (BrokerException bEx1) {
        if (bEx1.getErrorClass() == 74 && bEx1.getErrorCode() == 74) {
                // timeout means: no more messages, thus deregister.
        } else {
                throw bEx1;
        }
}
service.deregisterImmediate();
```

Since:

6.2.1.0

Field Summary

Fields inherited from class com.softwareag.entirex.aci.BrokerCommunication
`brokerService`

Constructor Summary

`UnitofWork``(BrokerService brokerService)` Creates a new UnitofWork object and attaches it to the given BrokerService.	
`UnitofWork``(BrokerService brokerService, ConversationState cstate)` Creates a new UnitofWork object and attaches it to the given BrokerService.	

Method Summary

`void`	**`backout`**`()` Backs out the current unit of work.
`void`	**`cancel`**`()` Cancels the current unit of work.
`void`	**`commit`**`()` Commits the current unit of work.

void	**commitBoth**() Commits the two units of work, one being currently received and one being currently sent in a single atomic operation.
void	**commitCancelConversation**() Commits the current unit of work and cancels the associated conversation.
void	**commitEndConversation**() Commits the current unit of work and ends the associated conversation.
void	**delete**() Deletes the persistent status of the current unit of work.
static void	**delete**(java.lang.String unitofWorkID, Broker broker) Deletes the persistent status of the specified unit of work.
void	**endConversation**() Ends the current conversation.
int	**getAttemptedDeliveryCount**() Returns how often it was attempted to deliver the unit of work.
java.util.Date	**getCommitTimestamp**() Returns the sender's commit timestamp for this unit of work as a Date object.
java.lang.String	**getCommitTimestampString**() Returns the sender's commit timestamp for this unit of work as a String object.
java.lang.String	**getLifetime**() Returns the lifetime value of a unit of work.
java.lang.String	**getStatus**() Returns the current status of the current unit of work.
java.lang.String	**getUnitofWorkID**() Returns the unique identifier for the current unit of work.
java.lang.String	**getUserStatus**() Returns the user-defined status associated with the current unit of work.
UnitofWork	**query**() Queries the status of the current unit of work.
static UnitofWork	**query**(java.lang.String unitofWorkID, Broker broker) Queries the status of the specified unit of work.
static UnitofWork	**query**(java.lang.String unitofWorkID, BrokerService service) **Deprecated.** *If more than one service is used by one user, the returned UnitofWork object might belong to some other service.*
static UnitofWork	**queryLast**(Broker broker) Queries the status of the last unit of work created by the caller.

static UnitofWork	**queryLast**(BrokerService service) **Deprecated.** *If more than one service is used by one user, the returned UnitofWork object might belong to some other service.*
BrokerMessage	**receive**() Receives the first or subsequent message of a unit of work.
BrokerMessage	**receive**(java.lang.String wait) Receives the first or subsequent message of a unit of work.
static BrokerMessage	**receiveAny**(BrokerService bs) Receives the first message or subsequent message of a unit of work.
static BrokerMessage	**receiveAny**(BrokerService bs, java.lang.String wait) Receives the first or subsequent message of a unit of work.
static BrokerMessage	**receiveOld**(BrokerService bs) Receives the first message or subsequent message of a unit of work.
static BrokerMessage	**receiveOld**(BrokerService bs, java.lang.String wait) Receives the first or subsequent message of a unit of work.
void	**send**(BrokerMessage msg) Sends an asynchronous message as part of a unit of work.
void	**sendCommit**(BrokerMessage msg) Sends an asynchronous message which commits the unit of work.
void	**setDataPersistence**(boolean persist) Enables or disables data persistence when creating a new unit of work.
void	**setLifetime**(java.lang.String t) Sets the lifetime value of a unit of work.
void	**setStatusPersistence**(boolean persist) Enables or disables status persistence when creating a new unit of work.
void	**setStatusPersistence**(int lifetimeMultiplier) Enables status persistence when creating a new unit of work and sets the lifetime of the persistent status.
void	**setStatusPersistence**(java.lang.String lifetime) Enables status persistence when creating a new unit of work and sets the lifetime of the persistent status.
void	**setUserStatus**(java.lang.String u) Sets the user-defined status associated with the current unit of work.
void	**updateUserStatus**() Updates the user status field of the current unit of work.

Methods inherited from class com.softwareag.entirex.aci.BrokerCommunication
dispose, getBrokerService, getUserData, saveState, setUserData

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

UnitofWork

```
public UnitofWork(BrokerService brokerService)
```

Creates a new UnitofWork object and attaches it to the given BrokerService.

Parameters:

`brokerService` - the BrokerService object to which the unit of work communication belongs.

UnitofWork

```
public UnitofWork(BrokerService brokerService,
                  ConversationState cstate)
           throws java.lang.IllegalArgumentException
```

Creates a new UnitofWork object and attaches it to the given BrokerService.
The conversation of the UnitofWork object is restored from the ConversationState object.

Parameters:

`brokerService` - the BrokerService object to which the unit of work communication belongs.

`cstate` - the ConversationState object to be restored.

Throws:

`java.lang.IllegalArgumentException` - if ConversationState object is null

Since:

7.1.1.10

See Also:

`ConversationState`

Method Detail

setDataPersistence

```
public void setDataPersistence(boolean persist)
```

Enables or disables data persistence when creating a new unit of work.
When a new unit of work is created and this method has not been previously called, the persistence option specified in the Broker attribute file (keyword STORE) is used. If the sender wants to

control data persistence, this method must be called before the first send method call which creates the unit of work.
If a unit of work is persistent, its messages are saved in the persistent store when the sender commits the unit of work. They are retained until the receiver commits or cancels the unit of work or until its lifetime expires. If the EntireX Broker or the system fails after the unit of work is committed, the unit of work (and its associated conversation) will be restored to their last stable status when the EntireX Broker restarts.

Parameters:

`persist` - if *true* sender wants data persistence, if *false* sender does not want data persistence.

---

setStatusPersistence

```
public void setStatusPersistence(boolean persist)
```

Enables or disables status persistence when creating a new unit of work.
When a new unit of work is created and this method has not been previously called, the persistence option specified in the Broker attribute file (keyword UWSTATP) is used. If the sender wants to control status persistence, this method must be called before the first send method call which creates the unit of work. The lifetime of the persistent status is the same as the lifetime of the unit of work.

Parameters:

`persist` - if *true* sender wants persistence of status, if *false* sender does not want status persistence.

See Also:

`setStatusPersistence(int)`, `getStatus()`

---

setStatusPersistence

```
public void setStatusPersistence(int lifetimeMultiplier)
```

Enables status persistence when creating a new unit of work and sets the lifetime of the persistent status.
The lifetime for persistent status is a multiplier of the unit of work lifetime. The default is 1. This method must be called before the first send method call to set a higher lifetime value.
If a unit of work has persistent status, this status is maintained in the persistent store and is updated whenever the status changes. The persistent status remains in the persistent store after the unit of work is completed until the status lifetime has expired.

Parameters:

`lifetimeMultiplier` - a value between 1 and 254 (inclusive).

Throws:

`java.lang.IllegalArgumentException` - if argument is not in the range of 1 to 254.

See Also:

`setStatusPersistence(boolean)`, `setLifetime(java.lang.String)`, `getStatus()`

setStatusPersistence

```
public void setStatusPersistence(java.lang.String lifetime)
```

Enables status persistence when creating a new unit of work and sets the lifetime of the persistent status.
The lifetime for persistent status is set as a time value. This lifetime is added to the lifetime of the unit of work. This method must be called before the first send method call to set a higher lifetime value.
If a unit of work has persistent status, this status is maintained in the persistent store and is updated whenever the status changes. The persistent status remains in the persistent store after the unit of work is completed until the status lifetime has expired.
If you call `setStatusPersistence(int)` with a multiplier and `setStatusPersistence(java.lang.String)` with a lifetime value, the second method wins.

Parameters:

`lifetime` - a period, how the status of th unit of work should live. This time is measured in seconds/minutes/hours (depending on the trailing S/M/H character). When null or "NO" is specified, the operation is non-blocked.

See Also:

`setStatusPersistence(boolean)`, `setStatusPersistence(int)`, `setLifetime(java.lang.String)`, `getStatus()`

---

getStatus

```
public java.lang.String getStatus()
```

Returns the current status of the current unit of work.
The status is set by all methods which manipulate or query a unit of work. Applicable values are:

- `RECEIVED` - one or more messages have been sent as part of a unit of work which is not yet committed.
- `ACCEPTED` - the unit of work has been committed by the sender.
- `DELIVERED` - the unit of work is currently being received by the sender.
- `BACKEDOUT` - the unit of work was backed out prior to being committed by the sender.
- `PROCESSED` - the receiver of the unit of work has committed it.
- `CANCELLED` - the unit of work was not processed within the specified lifetime.
- `DISCARDED` - the unit of work was not persistent and its data was discarded over a restart.

In addition the following status values are returned by the receive method, they all reflect an current status of `DELIVERED`:

- `RECV_FIRST` - this message is the first message in the unit of work.
- `RECV_MIDDLE` - this message is not the first or last message in the unit of work.
- `RECV_LAST` - this message is the last message in the unit of work.
- `RECV_ONLY` - this message is the only message in the unit of work.

Returns:

the status as a String.

setLifetime

```
public void setLifetime(java.lang.String t)
```

Sets the lifetime value of a unit of work.
Each unit of work has a lifetime value. This is the period of time the unit of work is allowed to exist without being completed. It starts when the unit of work is created and ends when it is completed. If the unit of work has status ACCEPTED when this lifetime expires, it is placed into a TIMEOUT status. Lifetime timeouts will not occur in the RECEIVED or DELIVERED status. The lifetime clock is running only when the EntireX Broker is up and active.

Parameters:
t - the lifetime value measured in seconds/minutes/hours/days (depending on the trailing S/M/H/D character).
Throws:
java.lang.IllegalArgumentException - if argument is invalid.

getLifetime

```
public java.lang.String getLifetime()
```

Returns the lifetime value of a unit of work.

Returns:
the lifetime as a String.

getUnitofWorkID

```
public java.lang.String getUnitofWorkID()
```

Returns the unique identifier for the current unit of work.
The value generated by the EntireX Broker can be used by the query method call.

Returns:
the UnitofWorkID as a String.
See Also:
query(java.lang.String, BrokerService)

setUserStatus

```
public void setUserStatus(java.lang.String u)
```

Sets the user-defined status associated with the current unit of work.
It will be transmitted with a send or receive call. It will be returned with a query or queryLast call.

Parameters:
u - the user status, maximum length is 32 characters.
Throws:
java.lang.IllegalArgumentException - Thrown if the length of the user status exceeds 32 bytes.

getUserStatus

```
public java.lang.String getUserStatus()
```

Returns the user-defined status associated with the current unit of work.
It will be transmitted with a send or receive call. It will be returned with a query or queryLast call.

Returns:
the current value of the user status of this unit of work.

getAttemptedDeliveryCount

```
public int getAttemptedDeliveryCount()
```

Returns how often it was attempted to deliver the unit of work.
This count is incremented whenever a unit of work is backed out, either explicitly or by a timeout or restart.

Returns:
the attempted delivery count as `int`.

send

```
public void send(BrokerMessage msg)
          throws BrokerException
```

Sends an asynchronous message as part of a unit of work. Sends the message to the Broker without waiting for an answer.
The status of the unit of work is `RECEIVED`.

Parameters:
`msg` - BrokerMessage to send.
Throws:
`BrokerException` - A Broker exception.
`java.lang.IllegalArgumentException` - if parameter is invalid.

sendCommit

```
public void sendCommit(BrokerMessage msg)
               throws BrokerException
```

Sends an asynchronous message which commits the unit of work. Sends the message to the Broker without waiting for an answer.
The status of the unit of work changes from `RECEIVED` to `ACCEPTED`. It is now available to be received by the server.

Parameters:
`msg` - BrokerMessage to send.
Throws:
`BrokerException` - A Broker exception.
`java.lang.IllegalArgumentException` - Thrown if parameter is invalid.

backout

```
public void backout()
             throws BrokerException
```

Backs out the current unit of work.
If the sender of the unit of work calls this method and the status is RECEIVED, the status changes to BACKEDOUT. If persistent status is disabled, no trace of this unit of work remains.
If the receiver of the unit of work calls this method and the status is DELIVERED, the status changes to ACCEPTED and the number of attempted deliveries is incremented.

Throws:

BrokerException - A Broker exception.

cancel

```
public void cancel()
            throws BrokerException
```

Cancels the current unit of work.
If the sender of the unit of work calls this method and the status is ACCEPTED, the status changes to CANCELLED.
If the receiver of the unit of work calls this method and the status is DELIVERED, the status changes to CANCELLED.
In both cases, if persistent status is disabled, no trace of this unit of work remains.

Throws:

BrokerException - A Broker exception.

commit

```
public void commit()
            throws BrokerException
```

Commits the current unit of work.
If the sender of the unit of work calls this method, the status of the unit of work changes from RECEIVED to ACCEPTED. It is now available to be received by the server.
If the receiver of the unit of work calls this method and the status is DELIVERED, the status changes to PROCESSED. If persistent status is disabled, no trace of this unit of work remains.

Throws:

BrokerException - A Broker exception

commitBoth

```
public void commitBoth()
                throws BrokerException
```

Commits the two units of work, one being currently received and one being currently sent in a single atomic operation.

Throws:
`BrokerException` - A Broker exception
See Also:
`commit()`

---

delete

```
public void delete()
            throws BrokerException
```

Deletes the persistent status of the current unit of work.
The unit of work must be complete and must have been created by the caller. If persistent status is disabled, no trace of this unit of work remains.

Throws:
`BrokerException` - A Broker exception

---

delete

```
public static void delete(java.lang.String unitofWorkID,
                          Broker broker)
                   throws BrokerException
```

Deletes the persistent status of the specified unit of work.
The unit of work must be complete and must have been created by the caller. The caller is identified by the user ID and token specified in the constructor for the Broker object (which is the second parameter). If persistent status is disabled, no trace of this unit of work remains.

Parameters:
`unitofWorkID` - unique identifier retrieved by the getUnitofWorkID method
`broker` - reference to a Broker object
Throws:
`BrokerException` - A Broker exception
See Also:
`getUnitofWorkID()`

---

commitEndConversation

```
public void commitEndConversation()
                           throws BrokerException
```

Commits the current unit of work and ends the associated conversation.

Throws:
`BrokerException` - A Broker exception
See Also:
`commit()`

---

commitCancelConversation

```
public void commitCancelConversation()
                            throws BrokerException
```

Commits the current unit of work and cancels the associated conversation.

Throws:
`BrokerException` - A Broker exception
See Also:
`commit()`

---

queryLast

```
public static UnitofWork queryLast(BrokerService service)
                           throws BrokerException
```

Deprecated. *If more than one service is used by one user, the returned UnitofWork object might belong to some other service.*

Queries the status of the last unit of work created by the caller.
The caller is identified by the user ID and token specified in the constructor for the Broker object the BrokerService object belongs to.
A corresponding UnitofWork object is returned which is created automatically if necessary. If there is no UnitofWork object for the caller, `null` is returned.

Parameters:
`service` - a reference to the BrokerService the object belongs to.
Returns:
a UnitofWork object or `null`.
Throws:
`BrokerException` - A Broker exception
`java.lang.IllegalArgumentException` - if applied to a generic service.

---

queryLast

```
public static UnitofWork queryLast(Broker broker)
                           throws BrokerException
```

Queries the status of the last unit of work created by the caller.
The caller is identified by the user ID and token specified in the constructor for the Broker object.
A corresponding `UnitofWork` object is returned which is created automatically if necessary. If there is no `UnitofWork` object for the caller, `null` is returned.

Parameters:
`broker` - the Broker object.
Returns:
a UnitofWork object or `null`
Throws:
`BrokerException` - A Broker exception

query

```
public UnitofWork query()
                throws BrokerException
```

Queries the status of the current unit of work.
The status can be read with the `getStatus` method.

Returns:
the current unit of work.
Throws:
`BrokerException` - A Broker exception

query

```
public static UnitofWork query(java.lang.String unitofWorkID,
                               BrokerService service)
                        throws BrokerException
```

Deprecated. *If more than one service is used by one user, the returned UnitofWork object might belong to some other service.*

Queries the status of the specified unit of work.
The unit of work must have been created by the caller. A corresponding unit of work object is returned which is created automatically if necessary.

Parameters:
`unitofWorkID` - unique identifier retrieved by the getUnitofWorkID method
`service` - reference to a BrokerService object
Returns:
the unit of work for this ID and this BrokerService
Throws:
`BrokerException` - A Broker exception.
`java.lang.IllegalArgumentException` - Thrown if parameter is invalid.

query

```
public static UnitofWork query(java.lang.String unitofWorkID,
                               Broker broker)
                        throws BrokerException
```

Queries the status of the specified unit of work.
The unit of work must have been created by the caller. A corresponding unit of work object is returned which is created automatically if necessary.

Parameters:
`unitofWorkID` - unique identifier retrieved by the getUnitofWorkID method
`broker` - the Broker object.
Returns:
the unit of work for this ID and this Broker.

Throws:

`BrokerException` - A Broker exception.

---

updateUserStatus

```
public void updateUserStatus()
                    throws BrokerException
```

Updates the user status field of the current unit of work.

Throws:

`BrokerException` - A Broker exception

See Also:

`setUserStatus(java.lang.String)`

---

receive

```
public BrokerMessage receive(java.lang.String wait)
                    throws BrokerException
```

Receives the first or subsequent message of a unit of work. Uses the maximum receive length from the service.

Parameters:

`wait` - A timeout period, how long to wait for a receive, measured in seconds/minutes/hours (depending on the trailing S/M/H character). When null or "NO" is specified, the operation is non-blocked.

Returns:

the BrokerMessage received.

Throws:

`BrokerException` - A Broker exception

`java.lang.IllegalArgumentException` - Thrown if parameter is invalid.

---

receive

```
public BrokerMessage receive()
                    throws BrokerException
```

Receives the first or subsequent message of a unit of work. Uses the maximum receive length and the default wait time from the service.

Returns:

the BrokerMessage received.

Throws:

`BrokerException` - A Broker exception

Since:

EntireX 5.2.1

---

receiveOld

```
public static BrokerMessage receiveOld(BrokerService bs,
                                       java.lang.String wait)
                                throws BrokerException
```

Receives the first or subsequent message of a unit of work.
This receive method will accept only messages from existing conversations. Uses the maximum receive length from the service.

Parameters:

`bs` - The BrokerService used to receive the message.
`wait` - A timeout period, how long to wait for a receive, measured in seconds/minutes/hours (depending on the trailing S/M/H character). When null or "NO" is specified, the operation is non-blocked.

Returns:

the BrokerMessage received.

Throws:

`BrokerException` - A Broker exception
`java.lang.IllegalArgumentException` - Thrown if parameter is invalid.

Since:

EntireX 6.2.1.16

---

receiveOld

```
public static BrokerMessage receiveOld(BrokerService bs)
                                throws BrokerException
```

Receives the first message or subsequent message of a unit of work.
This receive method will accept only messages from existing conversations. Uses the maximum receive length and the default wait time from the service.

Parameters:

`bs` - The BrokerService used to receive the message.

Returns:

the BrokerMessage received.

Throws:

`BrokerException` - A Broker exception
`java.lang.IllegalArgumentException` - Thrown if a parameter is invalid.

Since:

EntireX 6.2.1.16

---

receiveAny

```
public static BrokerMessage receiveAny(BrokerService bs,
                                       java.lang.String wait)
                                throws BrokerException
```

Receives the first or subsequent message of a unit of work.
This receive method will accept messages from existing conversations and new conversations. Uses the maximum receive length from the service.

Parameters:

`bs` - The BrokerService used to receive the message.
`wait` - A timeout period, how long to wait for a receive, measured in seconds/minutes/hours (depending on the trailing S/M/H character). When null or "NO" is specified, the operation is non-blocked.

Returns:

the BrokerMessage received.

Throws:

`BrokerException` - A Broker exception
`java.lang.IllegalArgumentException` - Thrown if parameter is invalid.

Since:

EntireX 6.2.1.16

---

receiveAny

```
public static BrokerMessage receiveAny(BrokerService bs)
                                throws BrokerException
```

Receives the first message or subsequent message of a unit of work.
This receive method will accept messages from existing conversations and new conversations.
Uses the maximum receive length and the default wait time from the service.

Parameters:

`bs` - The BrokerService used to receive the message.

Returns:

the BrokerMessage received.

Throws:

`BrokerException` - A Broker exception
`java.lang.IllegalArgumentException` - Thrown if a parameter is invalid.

Since:

EntireX 6.2.1.16

---

endConversation

```
public void endConversation()
                    throws BrokerException
```

Ends the current conversation.
If the sender of the unit of work calls this method, the current conversation ends normally. It is now available to be received by the server.
If the receiver of the unit of work calls this method, the current conversation ends. Receiving further messages is possible.

Throws:

`BrokerException` - if the broker call fails.

---

getCommitTimestamp

```
public java.util.Date getCommitTimestamp()
```

Returns the sender's commit timestamp for this unit of work as a `Date` object. If no commit timestamp is available or not in the format `yyyyMMddHHmmssSSSZ`, `null` is returned.

Returns:

the date object for the commit timestamp or null.

---

getCommitTimestampString

```
public java.lang.String getCommitTimestampString()
```

Returns the sender's commit timestamp for this unit of work as a `String` object. The format is "yyyyMMddHHmmssSSSz"; "yyyy" for the year, "MM" for the month, "dd" for the day, "HH" for the hour, "mm" for the minute, "ss" for the second, "SSS" for the millisecond, and "z" for the timezone (see also `SimpleDateFormat` for detailed information on the format). The timezone is given as an offset and is always "-0000" for UTC. The timezone is provided to avoid unintended changes if supplied to the `Date` class or other classes. If no commit timestamp is available, `null` is returned.

Returns:

the string for the commit timestamp or null.

---

com.softwareag.entirex.jms

Interface JMSFormatter

All Known Implementing Classes:
TextFormatter, TextFormatterReplyQueue

```
public interface JMSFormatter
```

Interface to allow customer specific formatting of the JMS messages to connect to non-JMS clients. This interface has two methods. One converts byte arrays into JMS messages and the other converts JMS messages into byte arrays. The JMS messages are used by the JMS application and the byte arrays are send to or received from the EntireX Broker. Applications using the Broker ACI with units of work can send and receive these messages.
The methods have to obey the encoding of the byte array. Use the default encoding in conjunction with translation and conversion of the EntireX Broker. The byte array may contain binary parts.

Since:
7.1.1.54

Method Summary

byte[]	**fromJMSMessage**(javax.jms.Session session, javax.jms.Message message) Format a JMS message to send to a non-JMS application.
javax.jms.Message	**toJMSMessage**(javax.jms.Session session, byte[] buffer) Create a Message from the bytes received from the Broker.

Method Detail

fromJMSMessage

```
byte[] fromJMSMessage(javax.jms.Session session,
                      javax.jms.Message message)
                      throws javax.jms.JMSException
```

Format a JMS message to send to a non-JMS application. Use the methods of Message and its subclasses to access the content of the message.

Parameters:

`session` - the session used to send the message.
`message` - the `Message` to send to a non-JMS client.

Returns:

the message as a byte array in a customer specific format. It is not allowed to return `null` or an empty byte array.

Throws:

`javax.jms.JMSException` - if the message is not created properly. Wrap all `Throwable` objects into JMSExceptions and re-throw them.

---

toJMSMessage

```
javax.jms.Message toJMSMessage(javax.jms.Session session,
                               byte[] buffer)
                               throws javax.jms.JMSException
```

Create a `Message` from the bytes received from the Broker. Use the `createMessageXxx` methods of the `session` and the methods of `Message` and its subclasses to create the message.

Parameters:

`session` - the session which received the byte array.
`buffer` - the bytes received from the broker.

Returns:

a `Message`, created in the `session`.

Throws:

`javax.jms.JMSException` - if the message is not created properly. Wrap all `Throwable` objects into JMSExceptions and re-throw them.

---

com.softwareag.entirex.jms

Class TextFormatter

```
java.lang.Object
  └ com.softwareag.entirex.jms.TextFormatter
```

All Implemented Interfaces:
JMSFormatter

```
public class TextFormatterextends java.lang.Objectimplements JMSFormatter
```

Standard implementation for a formatter class for the EntireX JMS layer. Used to format JMS messages in a standard text message format. The text of a `TextMessage` is send as the payload of an ACI message contained in a unit of work.

Constructor Summary

`TextFormatter()`	

Method Summary

`byte[]`	**`fromJMSMessage`**`(javax.jms.Session session, javax.jms.Message message)` Formats a byte array from a JMS Message.
`javax.jms.Message`	**`toJMSMessage`**`(javax.jms.Session session, byte[] buffer)` Create a JMS message from the byte array received from the broker.

Methods inherited from class java.lang.Object

`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructor Detail

TextFormatter

```
public TextFormatter()
```

Method Detail

toJMSMessage

```
public javax.jms.Message toJMSMessage(javax.jms.Session session,
                                      byte[] buffer)
                               throws javax.jms.JMSException
```

Create a JMS message from the byte array received from the broker.

Specified by:
`toJMSMessage` in interface `JMSFormatter`
Parameters:
`session` - the JMS session, used to create the JMS message.
`buffer` - the message from the broker.
Returns:
the JMS message created from the byte array.
Throws:
`javax.jms.JMSException` - if the message is not properly created.

---

fromJMSMessage

```
public byte[] fromJMSMessage(javax.jms.Session session,
                             javax.jms.Message message)
                      throws javax.jms.JMSException
```

Formats a byte array from a JMS Message. This method assumes the message is a TextMessage.

Specified by:
`fromJMSMessage` in interface `JMSFormatter`
Parameters:
`session` - the JMS session (currently not used).
`message` - the JMS message to format.
Returns:
the message formatted as a byte array.
Throws:
`javax.jms.JMSException` - if the message is not properly created.

---

com.softwareag.entirex.jms

Class TextFormatterReplyQueue

```
java.lang.Object
  └ com.softwareag.entirex.jms.TextFormatterReplyQueue
```

All Implemented Interfaces:

JMSFormatter

```
public class TextFormatterReplyQueueextends java.lang.Objectimplements JMSFormatter
```

Standard implementation for a formatter class for the EntireX JMS layer. Used to format JMS messages in a standard text message format. The text of a `TextMessage` is send as the payload of an ACI message contained in a unit of work. The ReplyTo destination is included in this ACI message. The type of the ReplyTo destination is coded in a single character: ’1’ for queue (use ’JMS/<queue name>/QUEUE’ as service), ’2’ for temporary queue (use ’JMS/<queue name>/TMPQUEUE’ as service), ’3’ for topic (use ’<topic name>’ as topic), ’4’ for temporary topic (use ’<topic name>’ as topic).

Constructor Summary

TextFormatterReplyQueue()	

Method Summary

byte[]	**fromJMSMessage**(javax.jms.Session session, javax.jms.Message message) Formats a byte array from a JMS Message.
javax.jms.Message	**toJMSMessage**(javax.jms.Session session, byte[] buffer) Create a JMS message from the byte array received from the broker.

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

TextFormatterReplyQueue

```
public TextFormatterReplyQueue()
```

Method Detail

toJMSMessage

```
public javax.jms.Message toJMSMessage(javax.jms.Session session,
                                      byte[] buffer)
                               throws javax.jms.JMSException
```

Create a JMS message from the byte array received from the broker.

Specified by:
`toJMSMessage` in interface `JMSFormatter`
Parameters:
`session` - the JMS session, used to create the JMS message.
`buffer` - the message from the broker.
Returns:
the JMS message created from the byte array.
Throws:
`javax.jms.JMSException` - if the message is not properly created.

---

fromJMSMessage

```
public byte[] fromJMSMessage(javax.jms.Session session,
                             javax.jms.Message message)
                      throws javax.jms.JMSException
```

Formats a byte array from a JMS Message. This method assumes the message is a TextMessage. The ReplyTo destination is coded in the first 32 characters of the message. Position 33 is the type of the ReplyTo destination. The byte array is build with the default encoding from the String.

Specified by:
`fromJMSMessage` in interface `JMSFormatter`
Parameters:
`session` - the JMS session (currently not used).
`message` - the JMS message to format. A `TextMessage` is needed.
Returns:
the message formatted as a byte array.
Throws:
`javax.jms.JMSException` - if the message is not properly created.

com.softwareag.entirex.xml.rt

Class XMLException

```
java.lang.Object
  └ java.lang.Throwable
      └ java.lang.Exception
          └ com.softwareag.entirex.xml.rt.XMLException
```

All Implemented Interfaces:

java.io.Serializable

```
public class XMLExceptionextends java.lang.Exception
```

Class XMLException is a checked exception for all sorts of problems
occurring in the XML/SOAP runtime component. Besides an exception text,
XMLException objects can hold an exception "kind" which describes the
subcomponent in which problems arose.
Exception class thrown by EntireX XML/SOAP Runtime classes.
Use *toString()* to retrieve the error message including the error class and error code.
Use *getMessage()* to retrieve the error message only.
Use *getErrorClass()* to retrieve the error class only.
Use *getNumber()* to retrieve the error code only.

See Also:

Serialized Form

Field Summary

static int	**XMLRUNTIME_CLASS** Error class for EntireX XML/SOAP Runtime

Method Summary

int	**getErrorClass**() Returns the error class part of the XML Runtime error.
int	**getErrorCode**() Returns the error code part of the XML Runtime error.
java.lang.String	**getErrorText**() Returns the error text part of the XML Runtime error.
java.lang.String	**getMessage**() Debugging method to write this exception text and kind onto stdout.
java.lang.String	**toString**() Returns the complete error information as string Format: Error-class Error-code Error-text Detail

Methods inherited from class java.lang.Throwable

`fillInStackTrace, getCause, getLocalizedMessage, getStackTrace, initCause, printStackTrace, printStackTrace, printStackTrace, setStackTrace`

Methods inherited from class java.lang.Object

`clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait`

Field Detail

XMLRUNTIME_CLASS

```
public static final int XMLRUNTIME_CLASS
```

Error class for EntireX XML/SOAP Runtime

See Also:

Constant Field Values

Method Detail

getErrorText

```
public java.lang.String getErrorText()
```

Returns the error text part of the XML Runtime error.

Returns:
XML Runtime error text as string.

---

getErrorCode

```
public int getErrorCode()
```

Returns the error code part of the XML Runtime error.

Returns:
XML Runtime error code as int.

---

getErrorClass

```
public int getErrorClass()
```

Returns the error class part of the XML Runtime error.

Returns:
XML Runtime error class as int.

---

getMessage

```
public java.lang.String getMessage()
```

Debugging method to write this exception text and kind onto stdout.

Overrides:
`getMessage` in class `java.lang.Throwable`
Returns:
String The string representation of this exception.

---

toString

```
public java.lang.String toString()
```

Returns the complete error information as string
Format: Error-class Error-code Error-text Detail

Overrides:
`toString` in class `java.lang.Throwable`
Returns:
The complete error information.

com.softwareag.entirex.xml.rt

Class XMLRPCServer

```
java.lang.Object
  └ com.softwareag.entirex.aci.Server
      └ com.softwareag.entirex.xml.rt.XMLRPCServer
```

All Implemented Interfaces:

com.softwareag.entirex.aci.MonitorInfoInterface, com.softwareag.entirex.aci.ServerMBean, com.softwareag.entirex.aci.ServerProperties

```
public class XMLRPCServerextends com.softwareag.entirex.aci.Server
```

XMLRPCServer extends com.softwareag.entirex.aci.Server.

Field Summary

Fields inherited from class com.softwareag.entirex.aci.Server

activeServers, COMMUNICATION_MODE_ALL, COMMUNICATION_MODE_MESSAGE, COMMUNICATION_MODE_RELIABLE, STATE_ERROR, STATE_INIT, STATE_RETRY, STATE_RUNNING, STATE_SHUTDOWN, verbose

Fields inherited from interface com.softwareag.entirex.aci.ServerProperties

BROKER_ID, CODE_PAGE, COMPRESS_LEVEL, CUSTOM_CLASS, ENCRYPT, ENCRYPT_LEVEL, ENVIRONMENT, FIXED_SERVERS, JMX_ENABLED, LOC_TRANS_CONFIG, LOC_TRANS_INI, LOC_TRANS_SET, LOGFILE, LOGICAL_BROKER_ID, LOGICAL_SERVICE, MAX_SERVERS, MAXRESTARTCYCLES, MIN_SERVERS, MONITOR_PORT, MONITOR_REMOTE, NAME, PASSWORD, PASSWORD_ENCRYPT, PROPERTIES_FILE, SECURITY, SERVER_ADR, SERVERLOGFILE, TIMEOUT, TRACE, USE_CODE_PAGE, USER_ID, VERBOSE, WAIT_ATTACH, WAIT_SERVER

Constructor Summary

XMLRPCServer() Constructor of XMLRPCServer	

Method Summary

void	**registerXMLRPCServerClass**(XMLRPCServerInterface xmlrpcserverclass) Register the implementation of XMLRPCServerInterface called if Java API for XML RPC Server is defined in configuration file.
void	**start**(java.lang.String[] args) Starts the XML RPC Server with an implementation of XMLRPCServerInterface.

Methods inherited from class com.softwareag.entirex.aci.Server

createCallHandler, getActiveServers, getArgusPort, getArgusTimeout, getBooleanProperty, getIntProperty, getMaxServers, getMinServers, getNumberBusyWorkers, getNumberWorkers, getProperties, getProperty, getPropertyFilename, getServerInfo, getServerName, getState, getTimestamp, getTraceLevel, getVerbose, getWorkersHighWatermark, isArgusMonitoringEnabled, remoteMonitoring, setCommandlineParser, setPropertyFilename, setServerName, setState, setTraceLevel, setVerbose, startServer, stop, stopServer

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

XMLRPCServer

public **XMLRPCServer**()

Constructor of XMLRPCServer

Method Detail

registerXMLRPCServerClass

```
public void registerXMLRPCServerClass(XMLRPCServerInterface xmlrpcserverclass)
```

Register the implementation of XMLRPCServerInterface called
if Java API for XML RPC Server is defined in configuration file.
Method must be called before starting the server.
To use implementation of XMLRPCServerInterface the configuration
file must define
<TargetServer name="xmlrpcServerClass">

Parameters:

`xmlrpcserverclass` - An implementation of XMLRPCServerInterface

---

start

```
public void start(java.lang.String[] args)
          throws java.lang.Exception
```

Starts the XML RPC Server with an implementation of XMLRPCServerInterface.

Parameters:

`args` - Command line arguments

Throws:

`java.lang.Exception` - on configuration errors and other critical errors.

---

com.softwareag.entirex.xml.rt

Interface XMLRPCServerInterface

```
public interface XMLRPCServerInterface
```

Definition of interface for Java-API of XML RPC Server

Method Summary

byte[]	**invoke**(byte[] request, java.util.Properties properties) Method must be implemented by application using Java-API of XML/SOAP RPC Server.

Method Detail

invoke

```
byte[] invoke(byte[] request,
              java.util.Properties properties)
              throws java.lang.Exception
```

Method must be implemented by application using Java-API of XML/SOAP RPC Server.

Parameters:

`request` - XML/SOAP request document

`properties` - Information of request and response document

Returns:

An byte array with with response document

Throws:

`java.lang.Exception`

com.softwareag.entirex.xml.rt

Class XMLRPCService

```
java.lang.Object
  └ com.softwareag.entirex.aci.BrokerService
      └ com.softwareag.entirex.aci.RPCService
          └ com.softwareag.entirex.xml.rt.XMLRPCService
```

```
public final class XMLRPCServiceextends RPCService
```

XMLRPCService extends com.softwareag.entirex.aci.RPCService. In general, an XMLRPCService object is instantiated by a Broker object, a server address and an XMM file.
To create an EntireX RPC which will send input XML data and retrieve output XML data, one of the invokeXML methods can be used on an XMLRPCService object.

Since:

6.1.1.0, 7.1.1.0

Field Summary

`static java.lang.String`	**PROPERTY_DEFAULT_FAULTDOC_FORMAT** Indicates which document protocol is used if no fault document is defined.
`static java.lang.String`	**PROPERTY_THROW_JAVA_EXCEPTION** Indicates if a java exception is thrown or a fault document is returned.
`static java.lang.String`	**PROPERTY_USE_CHARACTER_REFERENCE** Indicates if character reference used in document or the binary value of these characters is used.

Fields inherited from class com.softwareag.entirex.aci.RPCService
RELIABLE_AUTO_COMMIT, RELIABLE_CLIENT_COMMIT, RELIABLE_OFF

Fields inherited from class com.softwareag.entirex.aci.BrokerService
DEFAULT_WAITTIME

Constructor Summary

XMLRPCService(Broker broker, java.lang.String serverAddr, java.io.InputStream isXmmFile) Creates an XMLRPCService object.
XMLRPCService(Broker broker, java.lang.String serverAddr, java.lang.String sXmmFile) Creates an XMLRPCService object.
XMLRPCService(Broker broker, java.lang.String serverAddr, java.lang.String sXmmFile, java.io.InputStream isXmmFile) Creates an XMLRPCService object.
XMLRPCService(Broker broker, java.lang.String serverAddress, java.lang.String logicalBroker, java.lang.String logicalService, java.lang.String logialSetName, java.lang.String sXmmFile) Creates an XMLRPCService object.
XMLRPCService(java.lang.String sXmmFile) Create an XMLRPCService object.
XMLRPCService(java.lang.String logicalService, java.lang.String logicalSetName, java.lang.String sXmmFile) Creates an XMLRPCService object.
XMLRPCService(java.lang.String serverAddress, java.lang.String logicalBroker, java.lang.String logicalSetName, java.lang.String sXmmFile) Creates an XMLRPCService object.

Method Summary

byte[]	**invokeXML**(byte[] xmlDocument) Builds an RPC from XML input and returns result as XML output.
void	**invokeXML**(java.io.InputStream inputStream, java.io.OutputStream outputStream) Builds an RPC from XML input and returns result as XML output.
void	**invokeXML**(java.io.Reader xmlDocReader, java.io.Writer xmlDocWriter) Builds an RPC from XML input and returns result as XML output.
java.lang.String	**invokeXML**(java.lang.String xmlDocument) Builds an RPC from XML input and returns result as XML output.
void	**invokeXML**(javax.xml.stream.XMLStreamReader xmlStreamReader, javax.xml.stream.XMLStreamWriter xmlStreamWriter) Builds an RPC from XML input and returns result as XML output.
void	**setUserProperty**(java.lang.String key, java.lang.String value) Sets user-specific properties for this XMLRPCService object.

Methods inherited from class com.softwareag.entirex.aci.RPCService

closeConversation, closeConversationCommit, getCompression, getConversation, getMessageID, getNaturalLogon, getReliable, getRPCPassword, getRPCUserId, getStatusOfMessage, onEnter, onException, onLeave, onRetry, ping, reliableCommit, reliableRollback, setBroker, setCompression, setConversation, setLibraryName, setNaturalLogon, setReliable, setRpcLibrary, setRPCPassword, setRpcProgram, setRPCUserId, setServerAddress

Methods inherited from class com.softwareag.entirex.aci.BrokerService

cancelallConversations, deregister, deregisterImmediate, endallConversations, getBroker, getCharacterEncoding, getDefaultWaittime, getEnvironment, getMaxReceiveLen, getServerClass, getServerName, getServiceName, isGeneric, receive, receive, receiveAny, receiveAttachInfo, receiveOld, register, registerAttach, replyError, send, sendReceive, sendReceive, setAdjustReceiveLen, setDefaultWaittime, setEnvironment, setLogicalBroker, setLogicalBroker, setLogicalBroker, setLogicalService, setLogicalService, setMaxReceiveLen, toString, useCodePage, useCodePage

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Field Detail

PROPERTY_THROW_JAVA_EXCEPTION

```
public static final java.lang.String PROPERTY_THROW_JAVA_EXCEPTION
```

Indicates if a java exception is thrown or a fault document is returned.

- yes = java exception is thrown (default)
- no = fault document is generated and returned.

See Also:

Constant Field Values

PROPERTY_USE_CHARACTER_REFERENCE

```
public static final java.lang.String PROPERTY_USE_CHARACTER_REFERENCE
```

Indicates if character reference used in document or the binary value of these characters is used. The character <,>,",',& always print as characters references independent of this flag.

- yes = using character references
- no = always write characters (default)

See Also:

Constant Field Values

PROPERTY_DEFAULT_FAULTDOC_FORMAT

```
public static final java.lang.String PROPERTY_DEFAULT_FAULTDOC_FORMAT
```

Indicates which document protocol is used if no fault document is defined.

- xml = An XML fault document
- soap = A SOAP fault document (default)

See Also:

Constant Field Values

Constructor Detail

XMLRPCService

```
public XMLRPCService(java.lang.String sXmmFile)
```

Create an XMLRPCService object.

Parameters:

`sXmmFile` - The name of the XMM contains the information to map an XML document to an EntireX RPC and vice versa.

XMLRPCService

```
public XMLRPCService(Broker broker,
                     java.lang.String serverAddr,
                     java.lang.String sXmmFile)
```

Creates an XMLRPCService object.

Parameters:

`broker` - The Broker to run the service.
`serverAddr` - The RPC server address.
`sXmmFile` - The name of the XMM contains the information to map an XML document to an EntireX RPC and vice versa.

XMLRPCService

```
public XMLRPCService(Broker broker,
                     java.lang.String serverAddr,
                     java.io.InputStream isXmmFile)
```

Creates an XMLRPCService object.

Parameters:

`broker` - The Broker to run the service.
`serverAddr` - The RPC server address.
`isXmmFile` - The inputstream of the XMM contains the information to map an XML document to an EntireX RPC and vice versa.

XMLRPCService

```
public XMLRPCService(Broker broker,
                     java.lang.String serverAddr,
                     java.lang.String sXmmFile,
                     java.io.InputStream isXmmFile)
```

Creates an XMLRPCService object.

Parameters:

`broker` - The Broker to run the service.
`serverAddr` - The RPC server address.
`sXmmFile` - The name of the XMM contains the information to map an XML document to an EntireX RPC and vice versa.

`isXmmFile` - The inputstream of specified XMM contains the information to map an XML document to an EntireX RPC and vice versa.

XMLRPCService

```
public XMLRPCService(Broker broker,
                     java.lang.String serverAddress,
                     java.lang.String logicalBroker,
                     java.lang.String logicalService,
                     java.lang.String logialSetName,
                     java.lang.String sXmmFile)
```

Creates an XMLRPCService object.

Parameters:

`broker` - The Broker to run the service.
`serverAddress` - The RPC server address.
`logicalBroker` - The logical Broker (Location Transparency).
`logicalService` - The logical service (Location Transparency).
`logialSetName` - The logical setname (Location Transparency).
`sXmmFile` - The name of the XMM contains the information to map an XML document to an EntireX RPC and vice versa.

XMLRPCService

```
public XMLRPCService(java.lang.String logicalService,
                     java.lang.String logicalSetName,
                     java.lang.String sXmmFile)
```

Creates an XMLRPCService object.

Parameters:

`logicalService` - The logical service (Location Transparency).
`logicalSetName` - The logical setname (Location Transparency).
`sXmmFile` - The name of the XMM contains the information to map an XML document to an EntireX RPC and vice versa.

XMLRPCService

```
public XMLRPCService(java.lang.String serverAddress,
                     java.lang.String logicalBroker,
                     java.lang.String logicalSetName,
                     java.lang.String sXmmFile)
```

Creates an XMLRPCService object.

Parameters:

`serverAddress` - The RPC server address.
`logicalBroker` - The logical Broker (Location Transparency).
`logicalSetName` - The logical setname (Location Transparency).
`sXmmFile` - The name of XMM contains the information to map an XML document to an EntireX RPC and vice versa.

Method Detail

setUserProperty

```
public void setUserProperty(java.lang.String key,
                            java.lang.String value)
```

Sets user-specific properties for this XMLRPCService object.
Defined Values:

- PROPERTY_THROW_JAVA_EXCEPTION yes|no
- PROPERTY_USE_CHARACTER_REFERENCE yes|no
- PROPERTY_DEFAULT_FAULTDOC_FORMAT xml|soap

Parameters:

`key` - Name of property to set.
`value` - Value of property to set.

---

invokeXML

```
public java.lang.String invokeXML(java.lang.String xmlDocument)
                           throws BrokerException,
                                  XMLException
```

Builds an RPC from XML input and returns result as XML output.

Parameters:

`xmlDocument` - The requesting XML document as string containing the RPC input data information.

Returns:

On success the resulting XML document is delivered as string.

Throws:

`BrokerException`
`XMLException`

---

invokeXML

```
public byte[] invokeXML(byte[] xmlDocument)
                 throws BrokerException,
                        XMLException
```

Builds an RPC from XML input and returns result as XML output.

Parameters:

`xmlDocument` - The requesting XML document as byte array containing the RPC input data information.

Returns:

On success the resulting XML document is delivered as byte array.

Throws:

`BrokerException`
`XMLException`

invokeXML

```
public void invokeXML(java.io.Reader xmlDocReader,
                      java.io.Writer xmlDocWriter)
               throws BrokerException,
                      XMLException
```

Builds an RPC from XML input and returns result as XML output.

Parameters:

`xmlDocReader` - A reader object from where the input XML data can be retrieved.
`xmlDocWriter` - A writer object the XML output can be written to after a successful RPC.

Throws:

`BrokerException`
`XMLException`

invokeXML

```
public void invokeXML(javax.xml.stream.XMLStreamReader xmlStreamReader,
                      javax.xml.stream.XMLStreamWriter xmlStreamWriter)
               throws XMLException,
                      BrokerException
```

Builds an RPC from XML input and returns result as XML output.

Parameters:

`xmlStreamReader` - The requesting XML document containing the RPC input data information.
`xmlStreamWriter` - containg the resulting XML document

Throws:

`BrokerException`
`XMLException`

invokeXML

```
public void invokeXML(java.io.InputStream inputStream,
                      java.io.OutputStream outputStream)
               throws BrokerException,
                      XMLException
```

Builds an RPC from XML input and returns result as XML output.

Parameters:

`inputStream` - An InputStream object from where the input XML data can be retrieved.
`outputStream` - A OutputStream object where the XML output can be written to after a successful RPC.

Throws:

`BrokerException`
`XMLException`

Serialized Form

Package com.softwareag.entirex.aci

Class com.softwareag.entirex.aci.BrokerException extends java.lang.Exception implements Serializable

Serialized Fields

errorClass

`int` **`errorClass`**

---

errorCode

`int` **`errorCode`**

---

callInfo

`java.lang.String` **`callInfo`**

---

errorText

`java.lang.String` **`errorText`**

---

params

`java.lang.String[]` **`params`**

errorCodeString

`java.lang.String` **`errorCodeString`**

errorClassString

`java.lang.String` **`errorClassString`**

errorDetail

`java.lang.String` **`errorDetail`**

The error detail is part of the error text. The prefix (usually 'Broker Error') and the error class and error code are stripped off.

Class com.softwareag.entirex.aci.ConversationState extends java.lang.Object implements Serializable

serialVersionUID: -6736846114269261741L

Serialization Methods

readObject

```
private void readObject(java.io.ObjectInputStream in)
                throws java.io.IOException,
                       java.lang.ClassNotFoundException
```

Override the default deserialization method to restore time and convID from ticket.

Throws:

`java.io.IOException` - if the ticket is not readable.
`java.lang.ClassNotFoundException`

Serialized Fields

ticket

`java.lang.String` **`ticket`**

Class com.softwareag.entirex.aci.RPCMessageException extends java.lang.Exception implements Serializable

serialVersionUID: 1L

Serialized Fields

rpcProtocolError

`int` **`rpcProtocolError`**

---

rpcUserError

`int` **`rpcUserError`**

---

naturalErrorStatus

`java.lang.String` **`naturalErrorStatus`**

---

naturalErrorProgramName

`java.lang.String` **`naturalErrorProgramName`**

---

naturalErrorSubroutineLevel

`java.lang.String` **`naturalErrorSubroutineLevel`**

---

naturalErrorObjectType

`java.lang.String` **`naturalErrorObjectType`**

---

naturalErrorText

`java.lang.String` **`naturalErrorText`**

---

errorMessage

`java.lang.String` **`errorMessage`**

---

errorMessageDetail

`java.lang.String` **`errorMessageDetail`**

Class com.softwareag.entirex.aci.ServerException extends BrokerException implements Serializable

Serialized Fields

linkedException

`java.lang.Throwable` **`linkedException`**

The exception wrapped into a `ServerException`.

---

severity

`int` **`severity`**

Severity of this exception. Only the constants below are allowed as values.

Class com.softwareag.entirex.aci.Tester2.AboutDialog extends com.softwareag.entirex.aci.Tester2.Tester2Dialog implements Serializable

Serialized Fields

TITLE

`java.lang.String` **`TITLE`**

OK

`java.lang.String` **`OK`**

KEY_OK

`java.lang.String` **`KEY_OK`**

VERSION

`java.lang.String` **`VERSION`**

COPYRIGHT

`java.lang.String` **`COPYRIGHT`**

RIGHTSRESERVED

java.lang.String **RIGHTSRESERVED**

---

info

java.lang.String **info**

---

pMain

javax.swing.JPanel **pMain**

---

pTop

javax.swing.JPanel **pTop**

---

pMiddle

javax.swing.JPanel **pMiddle**

---

pBottom

javax.swing.JPanel **pBottom**

---

lblVersion

javax.swing.JLabel **lblVersion**

---

lblCopyright

javax.swing.JLabel **lblCopyright**

---

lblRightsReserved

javax.swing.JLabel **lblRightsReserved**

---

txtArea

javax.swing.JTextArea **txtArea**

---

spArea

javax.swing.JScrollPane **spArea**

verticalScrollBar

javax.swing.JScrollBar **verticalScrollBar**

btnOK

javax.swing.JButton **btnOK**

Class com.softwareag.entirex.aci.Tester2.Tester2Options extends com.softwareag.entirex.aci.Tester2.Tester2Dialog implements Serializable

serialVersionUID: 1L

Serialized Fields

WINDOWTITLE

java.lang.String **WINDOWTITLE**

USERPASSWORD

java.lang.String **USERPASSWORD**

NATURALTITLE

java.lang.String **NATURALTITLE**

NATURALLIBRARY

java.lang.String **NATURALLIBRARY**

NATURALLOGON

java.lang.String **NATURALLOGON**

RPCUSERID

java.lang.String **RPCUSERID**

RPCPASSWORD

java.lang.String **RPCPASSWORD**

TRACETITLE

java.lang.String **TRACETITLE**

TRACELEVEL

java.lang.String **TRACELEVEL**

TRACEVALUES

java.lang.String[] **TRACEVALUES**

TRACEDEFAULTVALUE

int **TRACEDEFAULTVALUE**

OK

java.lang.String **OK**

CANCEL

java.lang.String **CANCEL**

KEY_USER

java.lang.String **KEY_USER**

KEY_PASSWORD

java.lang.String **KEY_PASSWORD**

KEY_NATURALLIBRARY

java.lang.String **KEY_NATURALLIBRARY**

KEY_NATURALLOGON

java.lang.String **KEY_NATURALLOGON**

KEY_RPCUSERID

java.lang.String **KEY_RPCUSERID**

KEY_RPCPASSWORD

java.lang.String **KEY_RPCPASSWORD**

KEY_TRACELEVEL

java.lang.String **KEY_TRACELEVEL**

pFrame

javax.swing.JPanel **pFrame**

pUserPassword

javax.swing.JPanel **pUserPassword**

pNatural

javax.swing.JPanel **pNatural**

pTrace

javax.swing.JPanel **pTrace**

tbUserPassword

javax.swing.border.TitledBorder **tbUserPassword**

tbNatural

javax.swing.border.TitledBorder **tbNatural**

tbTrace

javax.swing.border.TitledBorder **tbTrace**

cbEnableNaturalLogon

javax.swing.JCheckBox **cbEnableNaturalLogon**

lblUser

javax.swing.JLabel **lblUser**

lblPassword

javax.swing.JLabel **lblPassword**

txtUser

javax.swing.JTextField **txtUser**

txtPassword

javax.swing.JPasswordField **txtPassword**

lblNaturalLibrary

javax.swing.JLabel **lblNaturalLibrary**

lblRPCUserID

javax.swing.JLabel **lblRPCUserID**

lblRPCPassword

javax.swing.JLabel **lblRPCPassword**

lblTracelevel

javax.swing.JLabel **lblTracelevel**

txtNaturalLibrary

javax.swing.JTextField **txtNaturalLibrary**

txtRPCUserID

javax.swing.JTextField **txtRPCUserID**

txtRPCPassword

javax.swing.JPasswordField **txtRPCPassword**

combTracelevel

javax.swing.JComboBox **combTracelevel**

btnOK

javax.swing.JButton **btnOK**

btnCancel

javax.swing.JButton **btnCancel**

traceValues

java.util.Vector<E> **traceValues**

tempNaturalLogonEnabled

boolean **tempNaturalLogonEnabled**

tempTracelevel

int **tempTracelevel**

Class com.softwareag.entirex.aci.TunnelServlet extends javax.servlet.http.HttpServlet implements Serializable

Serialized Fields

brokerId

java.lang.String **brokerId**

logicalBrokerID

java.lang.String **logicalBrokerID**

logicalSetName

java.lang.String **logicalSetName**

logEnabled

`boolean` **`logEnabled`**

---

bLocationTransparency

`boolean` **`bLocationTransparency`**

Package com.softwareag.entirex.jms

Class com.softwareag.entirex.jms.ExxReferencable extends java.lang.Object implements Serializable

Serialized Fields

name

`java.lang.String` **`name`**

---

value

`java.lang.String` **`value`**

---

optionalValue

`java.lang.String` **`optionalValue`**

Package com.softwareag.entirex.xml.rt

Class com.softwareag.entirex.xml.rt.DeploymentException extends java.lang.Exception implements Serializable

Serialized Fields

errorClass

`int` **`errorClass`**

---

errorNumber

`int` **`errorNumber`**

---

errorMessage

`java.lang.String` **`errorMessage`**

---

details

`java.util.ArrayList<E>` **`details`**

Class com.softwareag.entirex.xml.rt.XMLException extends java.lang.Exception implements Serializable

Serialized Fields

errorNumber

`int` **`errorNumber`**

---

errorClass

`int` **`errorClass`**

---

errorText

`java.lang.String` **`errorText`**

---

wrappedException

`java.lang.Exception` **`wrappedException`**

---

parameterList1

`java.util.HashMap<K,V>` **`parameterList1`**

---

printStackTrace1

`java.lang.String` **`printStackTrace1`**

---

parameterList2

`java.util.HashMap<K,V>` **`parameterList2`**

---

printStackTrace2

`java.lang.String` **`printStackTrace2`**

Class com.softwareag.entirex.xml.rt.XMLServlet extends javax.servlet.http.HttpServlet implements Serializable

---

Overview Package Class **Tree** **Deprecated** **Index** **Help**

PREV NEXT **FRAMES** **NO FRAMES** **All Classes**

Deprecated API

Contents

Deprecated Methods

com.softwareag.entirex.aci.BrokerCommunication.dispose() *This method does nothing, since no reference to the Conversation or UnitofWork object is held anymore.*
com.softwareag.entirex.aci.UnitofWork.query(String, BrokerService) *If more than one service is used by one user, the returned UnitofWork object might belong to some other service.*
com.softwareag.entirex.aci.UnitofWork.queryLast(BrokerService) *If more than one service is used by one user, the returned UnitofWork object might belong to some other service.*