Writing Advanced Applications - Java Wrapper

Each generated Java Wrapper class inherits methods from the EntireX Java ACI.

This section describes what can be performed with the methods inherited from the class RPCService.

Natural Logon or Changing the Library Name

The library name sent with the RPC request to the EntireX RPC or the Natural RPC Server is specified in the Software AG IDL file (see `library-definition` under *Software AG IDL Grammar*). When the RPC is executed, this library name can be overwritten.

To overwrite the library, a C Wrapper client must

- Call the `setLibraryName` method of the generated Java Wrapper class with the new library name as a parameter.

To force the library to be considered by Natural RPC Server

- Call the `setNaturalLogon` method of the generated Java Wrapper class with the parameter set to True.

Warning:
Natural and EntireX RPC servers behave differently regarding the library name.

See *Natural Logon or Changing the Library Name*.

Customizing the Generated Java Classes

You can extend the generated Java Wrapper Class of the client. By default, the generated client class is a subclass of `com.softwareag.entirex.aci.RPCService`. The customization component allows you to specify a class used as the superclass of the generated client class. This user-defined class (customization class) must be a subclass of `com.softwareag.entirex.aci.RPCService`.

When a customization class is specified, the calls to the user-exit methods `onEnter`, `onLeave`, `onException` and `onRetry` are generated.

To generate a customized Java Wrapper client

1. Implement your customization class. If you use a package for your customization class, specify package and class in the following step. Place the source for the customization class in the package folder, using the folder of the IDL file as package-root. The customization class needs a default constructor and one additional constructor with 4 arguments. See the example below.

2. Specify the name of your customization class in the *EntireX Workbench*, under **Tools**, **Options**, **Java**. This name is stored in the *entirex.properties* file (which is in your home directory) using the key `entirex.wrapper.custom.class`.

3. Generate the wrapper client classes

To use the customized Java Wrapper client

- Add (public) arbitrary methods and fields to your customization class. These methods and fields are inherited by the generated client class. Add your own processing instructions to these methods.

To perform all Broker-related processing in the generated Java Wrapper client

1. Overwrite the constructors of `RPCService`. You can instantiate wrapper classes without specifying a Broker object and server address as a parameter. Use the method `setbroker()` to set or change the reference to the Broker object, and the method `setServerAddress()` to set or change the server address.

2. Use the four user exit methods `onEnter`, `onLeave`, `onException` and `onRetry`. These methods have default implementations in `RPCService` and can be overwritten in your customization class. These exits are called at the beginning and the end of each generated method of the Java Wrapper class and when a broker exception is thrown. See `RPCService`.

Example of a Customization Class

```
package ExamplePackage;

import com.softwareag.entirex.aci.Broker;
import com.softwareag.entirex.aci.BrokerException;
import com.softwareag.entirex.aci.RPCService;

public class ExampleCustomization extends RPCService {
   public ExampleCustomization ()
   {
      super();
   }
   public ExampleCustomization (Broker broker, String serverAddr, String
libName, boolean compress)
   {
      super(broker, serverAddr, libName, compress);
   }
   protected void onEnter(String progname) throws BrokerException {
      // insert your implementation here.
   }

   protected void onLeave(String progname, int sendLen, int receiveLen) throws
BrokerException {
      // insert your implementation here.
   }

   protected void onException(String progname, BrokerException exception) throws
BrokerException {
      // insert your implementation here.
   }

   protected boolean onRetry(String progname, BrokerException exception) throws
BrokerException {
      // insert your implementation here.
      return false;
   }
}
```

Using RPC Compression

EntireX and Natural RPC support a feature called RPC compression to reduce network traffic. The default for compression is on. See *RPC Compression*.

To switch compression on and off

- Use the `setCompression` method of the class `RPCService`.

To check the current compression setting

- Use the `getCompression` method of the class `RPCService`.

Using Conversational RPC

It is assumed that you are familiar with the concepts of conversational and non-conversational RPC. See *Conversational RPC*.

To enable conversational RPC

1. Create a Conversation object and set this with `setConversation` on the wrapper object.
2. Different wrapper objects can participate in the same conversation if they use the same instance of a conversation object.

To abort a conversational RPC communication

- Abort an RPC conversation by calling the `closeConversation` method

To close and commit a conversational RPC communication

- Commit the RPC conversation by calling the `closeConversationCommit` method.

Warning:
Natural RPC Servers and EntireX RPC Servers behave differently when ending an RPC conversation.

See *Conversational RPC*.

Using Natural Security

A Natural RPC Server may run under Natural Security to protect RPC requests. See *Natural Security* under *Common Features of Wrappers and RPC-based Components*.

To authenticate a Java Wrapper client against Natural Security

- Specify a user ID and password in the `logon` method of class `Broker`.

  If different user IDs and/or passwords are used for EntireX Security and Natural Security, use the methods `setRPCUserId` or `setRPCPassword` to set the user IDs and/or passwords for Natural Security.

To force a Java Wrapper client to log on to a specific Natural library

1. Call the `setLibraryName` method of the generated wrapper objects with the new library name as a parameter.
2. Call the `setNaturalLogon` method of the generated wrapper objects with the parameter set to true.

See also *Natural Logon or Changing the Library Name*.

Example:

Assume that library is a wrapper object that is generated from an IDL library. This object extends `com.softwareag.entirex.aci.RPCService`. For this object, call the methods as shown:

```
library.setRPCUserId("testuser");
library.setRPCPassword("password");
library.setLibraryName("NATLIB");  // this is necessary only if the Natural Library
                                   // name is different from the library name in the IDL.
library.setNaturalLogon(true);
```

The order of the four methods is arbitrary.

Support of DVIPA

A TCP/IP connection established between stub and broker is not exclusively assigned to a particular thread. With multi-threaded applications, two or more threads may use the same connection. On the other hand, if a connection is busy, another new one is created to exchange data.

In order to access the same z/OS broker instance in a DVIPA-controlled environment, an affinity between application thread and TCP/IP connection is needed to always use the same connection within an application thread. Therefore, an environment variable is evaluated to control the handling of TCP/IP connections.

If broker ID contains the parameter "`poolsize=0`" (e.g. `ETB001?poolsize=0`), an affinity between threads and TCP/IP connections is established. All requests to one particular broker will use the same TCP/IP connection.

See also *Support of Clustering in a High Availability Scenario* under z/OS | UNIX | Windows.