

# Reliable RPC for Java Wrapper

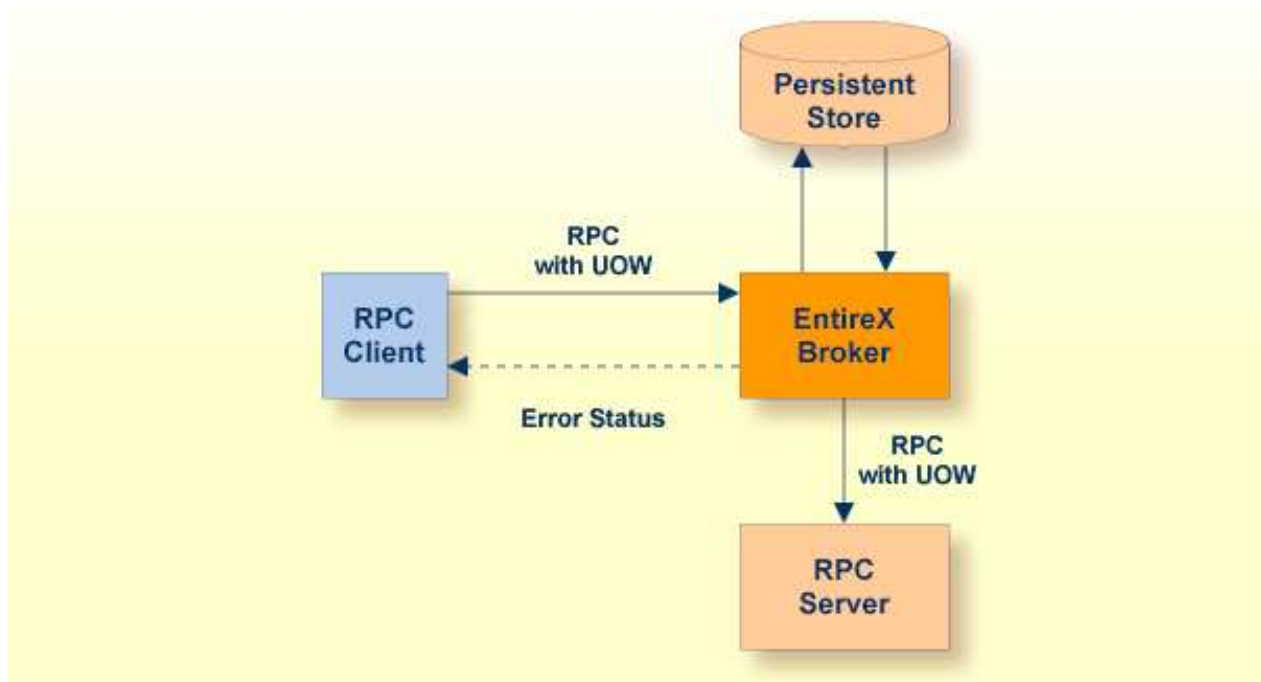
- Introduction to Reliable RPC
  - Writing a Client
  - Writing a Server
  - Broker Configuration
- 

## Introduction to Reliable RPC

In the architecture of modern e-business applications (such as SOA), loosely coupled systems are becoming more and more important. Reliable messaging is one important technology for this type of system.

Reliable RPC is the EntireX implementation of a reliable messaging system. It combines EntireX RPC technology and persistence, which is implemented with units of work (UOWs).

- Reliable RPC allows asynchronous calls ("fire and forget")
- Reliable RPC is supported by most EntireX wrappers
- Reliable RPC messages are stored in the Broker's persistent store until a server is available
- Reliable RPC clients are able to request the status of the messages they have sent



Reliable RPC is used to send messages to a persisted Broker service. The messages are described by an IDL program that contains only IN parameters. The client interface object and the server interface object are generated from this IDL file, using the EntireX Java Wrapper.

Reliable RPC is enabled at runtime. The client has to set one of two different modes before issuing a reliable RPC request:

- `AUTO_COMMIT`
- `CLIENT_COMMIT`

While `AUTO_COMMIT` commits each RPC message implicitly after sending it, a series of RPC messages sent in a unit of work (UOW) can be committed or rolled back explicitly using `CLIENT_COMMIT` mode.

The server is implemented and configured in the same way as for normal RPC.

## Writing a Client

All methods for reliable RPC are available on the interface object. See `RPCService` for details. The methods are:

- `RPCService.setReliable`
- `RPCService.getReliable`
- `RPCService.reliableCommit`
- `RPCService.reliableRollback`
- `RPCService.getMessageId`
- `RPCService.getStatusOfMessage`

Example (this example is included as source in the *examples/RPC/reliable/JavaClient* folder):

Create Broker object and interface object.

```
Broker broker = new Broker(Mail.DEFAULT_BROKERID, userID);
Mail mail = new Mail(broker);
broker.logon();
```

Enable reliable RPC with `CLIENT_COMMIT`

```
mail.setReliable(RPCService.RELIABLE_CLIENT_COMMIT);
```

The first RPC message.

```
mail.sendmail("mail receiver", "Subject 1", "Text 1");
```

Check the status: get the message ID first and use it to retrieve the status.

```
String messageId = mail.getMessageID();
String messageStatus = mail.getStatusOfMessage(messageID);
System.out.println("Status: " + messageStatus + ", id: " + messageId);
```

The second RPC message.

```
mail.sendmail("mail receiver", "Subject 2", "Text 2");
```

Commit the two messages.

```
mail.reliableCommit();
```

Check the status again for the same message ID.

```
messageStatus = mail.getStatusOfMessage(messageID);
System.out.println("Status: " + messageStatus + ", id: " + messageID);
```

The third RPC message.

```
mail.sendmail("mail receiver", "Subject 3", "Text 3");
```

Check the status: get the new message ID and use it to retrieve the status.

```
messageID = mail.getMessageID();
messageStatus = mail.getStatusOfMessage(messageID);
System.out.println("Status: " + messageStatus + ", id: " + messageID);
```

Roll back the third message and check status.

```
mail.reliableRollback();
messageStatus = mail.getStatusOfMessage(messageID);
System.out.println("Status: " + messageStatus + ", id: " + messageID);
broker.logoff();
```

## Limitations

1. All program calls that are called in the same transaction (CLIENT\_COMMIT) must be in the same IDL library.
2. It is not allowed to switch from CLIENT\_COMMIT to AUTO\_COMMIT in a transaction.
3. Messages (IDL programs) have IN parameters only.

## Writing a Server

The server implementation consist of the four classes:

- Abstract<IDL library name>Server
- <IDL library name>
- <IDL library name>Server
- <IDL library name>Stub

Add your implementation to the class <IDL library name>Server. There are no server-side methods for reliable RPC. The server does not send back a message to the client. The server can run deferred, thus client and server do not necessarily run at the same time. If the server fails, it throws an exception. This causes a cancel of the transaction (unit of work inside the Broker) and the error code is written to the user status field of the unit of work.

## Broker Configuration

A Broker configuration with `PSTORE` is recommended. This enables the Broker to store the messages for more than one Broker session. These messages are still available after Broker restart. The attributes `STORE`, `PSTORE`, and `PSTORE-TYPE` in the Broker attribute file can be used to configure this feature. The lifetime of the messages and the status information can be configured with the attributes `UWTIME` and `UWSTAT-LIFETIME`. Other attributes such as `MAX-MESSAGES-IN-UOW`, `MAX-UOWS` and `MAX-UOW-MESSAGE-LENGTH` may be used in addition to configure the units of work. See *Broker Attributes*.

The result of the method `RPCService.getStatusOfMessage` depends on the configuration of the unit of work status lifetime. If the status is not stored longer than the message, the method returns `(not available)`.