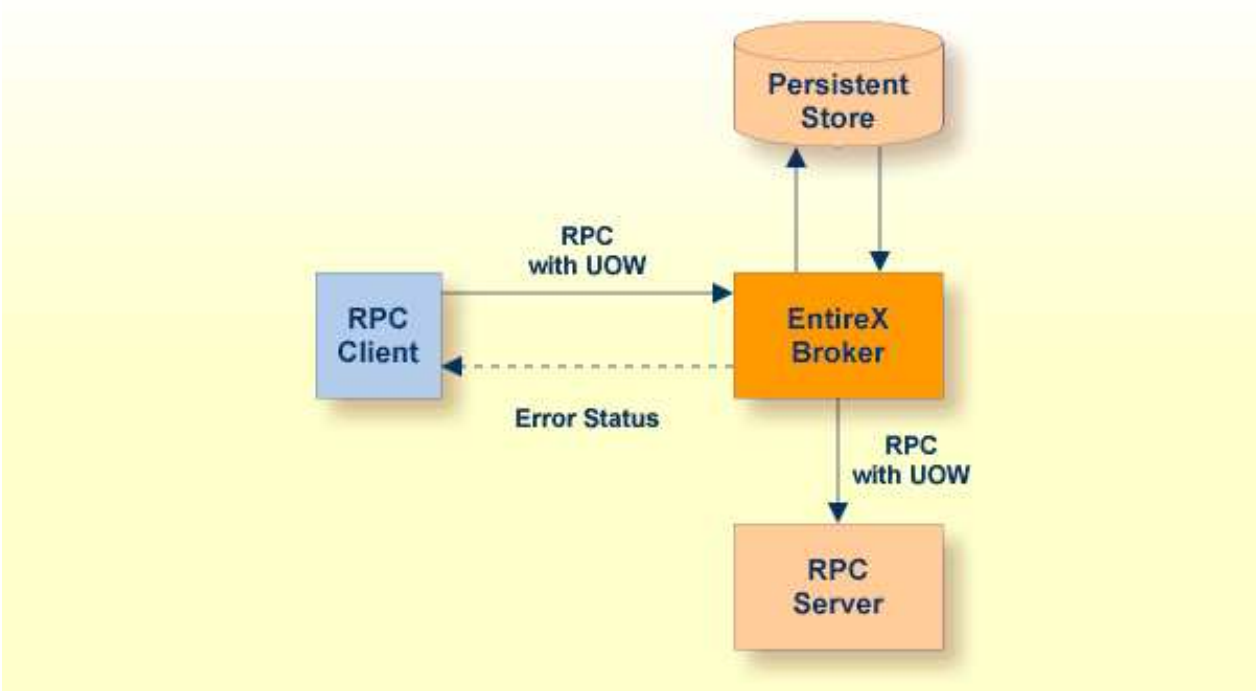
Reliable RPC for COBOL Wrapper

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 COBOL 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

The following steps describe how to write a COBOL reliable RPC client program with the scenario *Using the COBOL Wrapper for CICS with `DFHCOMMAREA` Calling Convention (z/OS and z/VSE)* and Linkage access to RPC communication.

Reliable RPC requires an explicit broker logon. See *Using Broker Logon and Logoff*.

Step 1: Declare the Data Structures for RPC Client Interface Objects

For every program definition in the Software AG IDL file, the templates will generate a copybook file that describes the customer data of the interface as a COBOL structure. For ease of use, the copybook can be embedded into the RPC client program.

However, if more appropriate, customer data structures can be used. In this case the COBOL data types and structures must match the interfaces of the generated client interface objects, otherwise unpredictable results will occur.

```
* Declare the customer data of the generated RPC interface
 01 SENDMAIL.
    02 SM-COMA.
       03 SM-TOADDRESS          PIC  X(60).
       03 SM-SUBJECT            PIC  X(20).
       03 SM-TEXT               PIC X(100).
```

Step 2: Declare and Initialize the RPC Communication Area

The RPC communication area must be declared and initialized in your RPC client program as follows:

```
* Declare RPC communication area
 02 ERX-COMMUNICATION-AREA.
    COPY ERXCOMM.
 . . . . . .

* Initialize RPC communication area
 INITIALIZE ERX-COMMUNICATION-AREA.
 MOVE "2000"           to COMM-VERSION.
```

Step 3: Required Settings in the RPC Communication Area

The following settings to the RPC communication area are required as a minimum to use the COBOL Wrapper. These settings have to be applied in your RPC client program. It is not possible to generate any defaults into your client interface objects:

```
* assign the broker to talk with
 MOVE "localhost:1971" to COMM-ETB-BROKER-ID.

* assign the server to talk with
 MOVE "RPC"            to COMM-ETB-SERVER-CLASS.
 MOVE "SRV1"           to COMM-ETB-SERVER-NAME.
 MOVE "CALLNAT"        to COMM-ETB-SERVICE-NAME.
* assign the user ID for Broker logon
 MOVE "ERXUSER"        to COMM-USERID.
 MOVE "PASSWORD"       to COMM-PASSWORD.
```

Step 4a: Perform a Broker Logon

```
MOVE "LO" TO COMM-FUNCTION.
EXEC CICS LINK
  PROGRAM  ("COBSRVI")
  COMMAREA (ERX-COMMUNICATION-AREA)
  LENGTH   (LENGTH OF ERX-COMMUNICATION-AREA)
  RESP     (CICS-RESP1)
  RESP2    (CICS-RESP2)
END-EXEC.
```

Step 4b: Examine the Error Code

Check whether the logon call was successful or not.

Step 5: Enable Reliable RPC with CLIENT_COMMIT

Before reliable RPC can be used, the reliable state must be set to either `ERX_RELIABLE_CLIENT_COMMIT` or `ERX_RELIABLE_AUTO_COMMIT`.

- "C" - `CLIENT_COMMIT`
- "A" - `AUTO_COMMIT`

```
* Set the reliable RPC mode
 MOVE "C" TO COMM-RELIABLE-STATE.
```

Step 6a: Send the RPC Message

The RPC message is sent using the `EXEC CICS LINK` interface.

```
* Send the RPC message
 MOVE DFHRESP(NORMAL) TO CICS-RESP1.
 MOVE DFHRESP(NORMAL) TO CICS-RESP2.
 MOVE ZEROES           TO COMM-RETURN-CODE.
 EXEC CICS LINK
   PROGRAM  ("SENDMAIL")
   RESP     (CICS-RESP1)
   RESP2    (CICS-RESP2)
   COMMAREA (SENDMAIL)
   LENGTH   (LENGTH OF SENDMAIL)
 END-EXEC.
```

Step 6b: Examine the Error Code

When the RPC message is returned, it needs to be checked whether it was successful or not:

```
 IF COMM-RETURN-CODE IS = ZERO
    Perform success-handling
 ELSE
    Perform error-handling
 END-IF.
```

The field COMM-RETURN-CODE in the RPC communication area contains the error provided by the COBOL Wrapper. For the error messages returned, see *Error Messages and Codes*.

Note:
After successful call (Step 6a) the `UOWID` is available in the RPC communication area field `COMM-ETB-UOW-ID`. See *The RPC Communication Area (Reference)*.

Step 7a: Check the Reliable RPC Message Status

To determine that reliable RPC messages are delivered, the reliable RPC message status can be queried. See *Understanding UOW Status* and *Broker UOW Status Transition* for more information.

```
 MOVE DFHRESP(NORMAL) TO CICS-RESP1.
 MOVE DFHRESP(NORMAL) TO CICS-RESP2.
 MOVE "RS" TO COMM-FUNCTION.
 MOVE ZEROES TO COMM-RETURN-CODE.
 EXEC CICS LINK
   PROGRAM  ("COBSRVI")
   RESP     (CICS-RESP1)
   RESP2    (CICS-RESP2)
   COMMAREA (ERX-COMMUNICATION-AREA)
   LENGTH   (LENGTH OF ERX-COMMUNICATION-AREA)
 END-EXEC.
```

Note:
After successful call the UOW status is available in the RPC communication area field `COMM-RELIABLE-STATUS`. See *The RPC Communication Area (Reference)*.

Step 7b: Examine the Error Code

Check whether the check status call was successful or not.

Step 8: Send a Second RPC Message

Send a second reliable RPC message. See Step 6a and Step 6b.

Step 9: Check the Reliable RPC Message Status

Check the reliable RPC message before the commit call. See Step 7a and Step 7b.

Step 10a: Commit both Reliable RPC Messages

Now both reliable RPC messages are committed. This will deliver all reliable RPC messages to the server if it is available.

```
MOVE DFHRESP(NORMAL) TO CICS-RESP1.
MOVE DFHRESP(NORMAL) TO CICS-RESP2.
MOVE "RC" TO COMM-FUNCTION.
MOVE ZEROES TO COMM-RETURN-CODE.
EXEC CICS LINK
  PROGRAM  ("COBSRVI")
  RESP     (CICS-RESP1)
  RESP2    (CICS-RESP2)
  COMMAREA (ERX-COMMUNICATION-AREA)
  LENGTH   (LENGTH OF ERX-COMMUNICATION-AREA)
END-EXEC.
```

Step 10b: Examine the Error Code

Check whether the commit call was successful or not.

Step 11: Send a Third RPC Message

Send a third reliable RPC message. See Step 5a and Step 5b.

Step 12: Check the Reliable RPC Message Status

Check the reliable RPC message before the rollback call. See Step 6.

Step 13a: Roll Back the Third RPC Message

Roll back the current reliable RPC message.

```
MOVE DFHRESP(NORMAL) TO CICS-RESP1.
MOVE DFHRESP(NORMAL) TO CICS-RESP2.
MOVE "RR" TO COMM-FUNCTION.
MOVE ZEROES TO COMM-RETURN-CODE.
EXEC CICS LINK
  PROGRAM  ("COBSRVI")
  RESP     (CICS-RESP1)
  RESP2    (CICS-RESP2)
  COMMAREA (ERX-COMMUNICATION-AREA)
  LENGTH   (LENGTH OF ERX-COMMUNICATION-AREA)
END-EXEC.
```

Step 13b: Examine the Error Code

When the rollback call is returned, check whether it was successful or not. If the rollback call failed, an explicit EOC needs to be sent:

```
 MOVE DFHRESP(NORMAL) TO CICS-RESP1.
 MOVE DFHRESP(NORMAL) TO CICS-RESP2.
 MOVE "RS" TO COMM-FUNCTION.
 MOVE ZEROES TO COMM-RETURN-CODE.
 EXEC CICS LINK
   PROGRAM  ("COBSRVI")
   RESP     (CICS-RESP1)
   RESP2    (CICS-RESP2)
   COMMAREA (ERX-COMMUNICATION-AREA)
   LENGTH   (LENGTH OF ERX-COMMUNICATION-AREA)
 END-EXEC.
```

Step 14a: Perform a Broker Logoff

```
MOVE "LF" TO COMM-FUNCTION.
EXEC CICS LINK
  PROGRAM  ("COBSRVI")
  COMMAREA (ERX-COMMUNICATION-AREA)
  LENGTH   (LENGTH OF ERX-COMMUNICATION-AREA)
  RESP     (CICS-RESP1)
  RESP2    (CICS-RESP2)
END-EXEC.
```

Step 14b: Examine the Error Code

Check whether the logoff call was successful or not.

Writing a 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 returns an error code greater than zero. This causes the transaction (unit of work inside the Broker) to be cancelled, and the error code is written to the user status field of the unit of work. For writing reliable RPC servers, see *Using the COBOL Wrapper for the Server Side*.

To execute a reliable RPC service with an RPC server, the parameter `logon` (`LOGN` under CICS) must be set to `YES`. See `logon` under z/OS (CICS | Batch | IMS) | MicroFocus | BS2000/OSD | z/VSE (CICS | Batch).

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 generic RPC function call "RS" - get reliable status depends on the configuration of the unit of work status lifetime in the EntireX Broker configuration. See `COMM-FUNCTION`. If the status is not stored longer than the message, the function call returns the error code 00780305 (`no matching UOW found`).