WS-ReliableMessaging

WS-Reliable Messaging

CXF supports both the official 1.1/1.2 Web Services Reliable Messaging (WS-ReliableMessaging) protocol using the http://docs.oasis-open.org/ws-rx/wsrm/200702 namespace and the outdated February 2005 submission version using the http://schemas.xmlsoap.org/ws/2005/02/rm/ namespace.

The submission version specified an outdated version of WS-Addressing, using the http://schemas.xmlsoap.org/ws/2004/08/addressing namespace, which has since been replaced by the official http://www.w3.org/2005/08/addressing namespace. Most other web services implementations supporting the submission version of WS-ReliableMessaging have deviated from the specification by moving to the official WS-Addressing release with the http://www.w3.org/2005/08/addressing namespace. CXF supports the submission version of RM with either WS-Addressing namespace.

For compatibility with older versions of CXF, the default is to use the February 2005 submission version of RM with the submission version of WS-Addressing. On the client side, you can configure CXF for whichever version of WS-ReliableMessaging you want to use (see Reliable Messaging Configuration Guide, along with the Runtime control properties below). On the provider side, CXF adapts to whichever version of WS-ReliableMessaging is used by the client and responds appropriately.

Like most other features in CXF, it is interceptor based. The WS-Reliable Messaging implementation consists of 6 interceptors in total:

Interceptor	Task
org.apache.cxf.ws.rm.RMOutInterceptor	Responsible for sending CreateSequence requests and waiting for their CreateSequenceResponse responses, and and aggregating the sequence properties (id and message number) for an application message.
org.apache.cxf.ws.rm.RMInInterceptor	Intercepting and processing RM protocol messages (these will not the application level), as well as SequenceAcknowledgments piggybacked on application messages.
org.apache.cxf.ws.rm.RMCaptureInInterceptor	Caching incoming messages for persistent storage.
org.apache.cxf.ws.rm.RMDeliveryInterceptor	Assuring InOrder delivery of messages to the application.
org.apache.cxf.ws.rm.soap.RMSoapInterceptor	Encoding and decoding the RM headers
org.apache.cxf.ws.rm.soap.RetransmissionInterceptor	Responsible for creating copies of application messages for future resends.

Interceptor Based QOS

The presence of the RM interceptors on the respective interceptor chains alone will take care that RM protocol messages are exchanged when necessary. For example, upon intercepting the first application message on the outbound interceptor chain, the RMOutInterceptor will send a CreateSequence request and only proceed with processing the original application message after it has the CreateSequenceResponse response. Furthermore, the RM interceptors are responsible for adding the Sequence headers to the application messages and, on the destination side, extracting them from the message.

This means that no changes to application code are required to make the message exchange reliable!

You can still control sequence demarcation and other aspects of the reliable exchange through configuration however. For example, while CXF by default attempts to maximize the lifetime of a sequence, thus reducing the overhead incurred by the RM protocol messages, you can enforce the use of a separate sequence per application message by configuring the RM source's sequence termination policy (setting the maximum sequence length to 1). See the Reliable Messaging Configuration Guide for more details on configuring this and other aspects of the reliable exchange.

Runtime control

Several message context property values can be set in client code to control the RM operation at runtime, with key values defined by public constants in org.apache.cxf.ws.rm.RMManager:

Key	Value
WSRM_VERSION_PROPERTY	String WS-RM version namespace (http://schemas.xmlsoap.org/ws/2005/02/rm/ or http://docs.oasis-open.org/ws-rx/wsrm/200702)
WSRM_WSA_VERSION_PROPERTY	String WS-Addressing version namespace (http://schemas.xmlsoap.org/ws/2004/08/addressing or http://www.w3.org/2005/08/addressing) - this property is ignored unless you're using the http://schemas.xmlsoap.org/ws/2005/02/rm/ RM namespace)
WSRM_LAST_MESSAGE_PROPERTY	Boolean value TRUE to tell the RM code that the last message is being sent, allowing the code to close the RM sequence and release resources (as of the 3.0.0 version of CXF the RM code will by default close the RM sequence when you close your client; earlier versions of CXF did not close the sequence unless told to using this flag, or if configured with a source policy <wsrm-mgr:sequenceTerminationPolicy terminateOnShutdown="true"/>)
WSRM_INACTIVITY_TIMEOUT_PROPERTY	Long inactivity timeout in milliseconds

WSRM_RETRANSMISSION_INTERVAL_PROPERTY	Long base retransmission interval in milliseconds
WSRM_EXPONENTIAL_BACKOFF_PROPERTY	Boolean exponential backoff flag
WSRM_ACKNOWLEDGEMENT_INTERVAL_PROPERTY	Long acknowledgement interval in milliseconds

You can also monitor and control many aspects of RM using the JMX Management features of CXF. The full list of JMX operations is defined by org.apache.cxf.ws.rm.ManagedRMManager and org.apache.cxf.ws.rm.ManagedRMEndpoint, but these operations include viewing the current RM state down to the individual message level. You can also use JXM to close and/or terminate an RM sequence, and to receive notification of when previously-sent messages are acknowledged by the remote RM endpoint.

For example, if you have the JMX server enabled in your client configuration you could use this code to track the last acknowledgement number received:

```
private static class AcknowledgementListener implements NotificationListener {
    private volatile long lastAcknowledgement;

    @Override
    public void handleNotification(Notification notification, Object handback) {
        if (notification instanceof AcknowledgementNotification) {
            AcknowledgementNotification ack = (AcknowledgementNotification)notification;
            lastAcknowledgement = ack.getMessageNumber();
        }
    }

    // initialize client
...
    // attach to JMX bean for notifications
    //  NOTE: you must have sent at least one message to initialize RM before executing this code
    Endpoint ep = ClientProxy.getClient(client).getEndpoint();
    InstrumentationManager im = bus.getExtension(InstrumentationManager.class);
    MBeanServer mbs = im.getMBeanServer();
    RMManager clientManager = bus.getExtension(RMManager.class);
    ObjectName name = RMUtils.getManagedObjectName(clientManager, ep);
    System.out.println("Looking for endpoint name " + name);
    AcknowledgementListener listener = new AcknowledgementListener();
    mbs.addNotificationListener(name, listener, null, null);

    // send messages using RM with acknowledgement status reported to listener
...
```