Red Hat AMQ 7.2

AMQ Clients Overview

For Use with AMQ Clients 2.3

Last Updated: 2019-03-18

Red Hat AMQ 7.2 AMQ Clients Overview

For Use with AMQ Clients 2.3

Legal Notice

Abstract

This document highlights features and components of AMQ Clients 2.3. It also demonstrates common use cases and design patterns supported in this release.

Table of Contents

PREFACE

AMQ Clients is a collection of AMQP 1.0 messaging APIs for multiple languages and platforms. It includes JMS 2.0 support and new, event-driven APIs to enable integration into existing applications.

AMQ Clients is part of Red Hat AMQ. For more information, see Introducing Red Hat AMQ 7.

CHAPTER 1. KEY FEATURES

- An open standard protocol - AMQP 1.0
- Industry-standard APIs - JMS 1.1 and 2.0
- New event-driven APIs - Fast, efficient messaging that integrates everywhere
- Broad language support - C++, Java, JavaScript, Python, Ruby, and .NET
- Wide availability - Linux, Windows, and JVM-based environments

CHAPTER 2. COMPONENTS

2.1. AMQP CLIENTS

AMQ Clients includes a suite of AMQP 1.0 messaging APIs. AMQP is an ISO-standard, general-purpose messaging protocol with rich messaging capabilities. Both AMQ Broker and AMQ Interconnect offer AMQP 1.0 support and therefore interoperate with any AMQP 1.0 client.

- Using the AMQ JMS Client
- Using the AMQ C++ Client
- Using the AMQ JavaScript Client
- Using the AMQ .NET Client
- Using the AMQ Python Client
- Using the AMQ Ruby Client

2.2. JMS CLIENTS AND LIBRARIES

AMQ Clients offers multiple implementations of the widely used Java Message Service (JMS) API.

AMQ JMS provides full AMQP 1.0 support and works with any AMQ server. For more information, see Using the AMQ JMS Client.

To support existing applications based on A-MQ 6, AMQ 7 includes the AMQ OpenWire JMS client. For more information, see Using the AMQ OpenWire JMS Client.

To support existing applications based on the ActiveMQ Artemis Core protocol, AMQ includes the AMQ Core Protocol JMS client. For more information, see Using the AMQ Core Protocol JMS Client.

To support efficient use of JMS resources, AMQ includes the AMQ JMS Pool library. It enables reuse of connection resources beyond the standard lifecycle defined by the JMS API. For more information, see Using the AMQ JMS Pool Library.

CHAPTER 3. EVENT-DRIVEN APIS

Many of the APIs provided with AMQ Clients are asynchronous, event-driven APIs. These include the C++, JavaScript, Python, and Ruby APIs.

These APIs work by executing application event-handling functions in response to network activity. The library monitors network I/O and fires events. The event handlers run sequentially on the main library thread.

Because the event handlers run on the main library thread, the handler code must not contain any long-running blocking operations. Blocking in an event handler blocks all library execution. If you need to execute a long blocking operation, you must call it on a separate thread. The event-driven APIs include cross-thread communication facilities to support coordination between the library thread and application threads.

AVOID BLOCKING IN EVENT HANDLERS

Long-running blocking calls in event handlers stop all library execution, preventing the library from handling other events and performing periodic tasks. Always start long-running blocking procedures in a separate application thread.

CHAPTER 4. AMQP

AMQP is an open internet protocol for reliably sending and receiving messages. It is supported by multiple software vendors and major institutions. AMQP 1.0 became an OASIS standard in 2012 and an ISO standard in 2014.

- A framed protocol with session multiplexing
- Supports peer-to-peer and client-server connections
- Provides a standard type system for lossless data exchange
- Offers flow control, heartbeating, and resource limits for increased reliability in distributed systems
- Uses a space-efficient binary encoding and pipelining to reduce latency

4.1. AMQP DELIVERY GUARANTEES

The AMQP model for settlement is based on the lifecycle of a message delivery. At each end of a link, an entity representing a message transfer is created, it exists for some period of time, and finally it is "settled", meaning it can be forgotten. There are four events of interest in the combined lifecycle of a delivery.

- The delivery is created at the sender.
- The delivery is created at the receiver.
- The delivery is settled at the sender.
- The delivery is settled at the receiver.

Because the sender and receiver are operating concurrently, these events can occur in various orders, and the order of these events results in differing message delivery guarantees.

At-most-once delivery

At-most-once delivery is also known as "presettled" or "fire and forget" delivery.

1. The delivery is created at the sender.
2. The delivery is settled at the sender.
3. The delivery is created at the receiver.
4. The delivery is settled at the receiver.

In this configuration the sender settles (that is, forgets) the delivery before it reaches the receiver, and if anything happens to the delivery in flight, the sender has no basis for resending.

This mode is suited to applications where temporary message loss is acceptable, such as for periodic sensor data, or when the application itself can detect the failure and resend.

At-least-once delivery

1. The delivery is created at the sender.
2. The delivery is created at the receiver.

3. The delivery is settled at the receiver.

4. The delivery is settled at the sender.

In this configuration, the receiver settles the delivery when it has received it, and the sender settles once it sees the receiver has settled. If anything happens to the delivery in flight, the sender can resend. The receiver, however, has already forgotten the delivery, so a resend will result in a duplicate message delivery. Applications can use unique message IDs to filter out duplicates.

CHAPTER 5. IMPORTANT NOTES

5.1. PREFERRED CLIENTS

In general, AMQ clients that support the AMQP 1.0 standard are preferred for new application development. However, the following exceptions apply:

- If your implementation requires distributed transactions, use the AMQ Core Protocol JMS client.
- If you require MQTT or STOMP in your domain (for IoT applications, for instance), use community-supported MQTT or STOMP clients.

The considerations above do not necessarily apply if you are already using:

- The AMQ OpenWire JMS client (the JMS implementation previously provided in A-MQ 6)
- The AMQ Core Protocol JMS client (the JMS implementation previously provided with HornetQ)

5.2. LEGACY CLIENTS

- **Deprecation of the CMS and NMS APIs**
  The ActiveMQ CMS and NMS messaging APIs are deprecated in AMQ 7. It is recommended that users of the CMS API migrate to AMQ C++, and users of the NMS API migrate to AMQ .NET. The CMS and NMS APIs might have reduced functionality in AMQ 7.
- **Deprecation of the legacy AMQ C++ client**
  The legacy AMQ C[]+ client (the C+ client previously provided in MRG Messaging) is deprecated in AMQ 7. It is recommended that users of this API migrate to AMQ C++.
- **The Core API is unsupported**
  The Artemis Core API client is not supported. This client is distinct from the AMQ Core Protocol JMS client, which is supported.

CHAPTER 6. IMPORTANT LINKS

- Red Hat AMQ 7 Supported Configurations
- Red Hat AMQ 7 Component Details

Revised on 2019-03-18 15:33:23 UTC