Red Hat AMQ 7.2

AMQ Clients 2.1 Release Notes

Release Notes for Red Hat AMQ Clients

Last Updated: 2018-07-27

Red Hat AMQ 7.2 AMQ Clients 2.1 Release Notes

Release Notes for Red Hat AMQ Clients

Abstract

These release notes contain the latest information about new features, enhancements, fixes, and issues contained in the AMQ Clients 2.1 release.

Table of Contents

CHAPTER 1. FEATURES

- AMQ Python is now supported on Windows.
- A new library for caching JMS resources, AMQ JMS Pool, is now available.
- This release improves the performance of AMQ JMS.

CHAPTER 2. ENHANCEMENTS

2.1. AMQ .NET

- **ENTMQCL-555 - Provide an example implementing reconnect and failover**
  The AMQ .NET examples now include a sample implementation of client reconnect and failover.

CHAPTER 3. RESOLVED ISSUES

3.1. AMQ C++

- **ENTMQCL-604 - Receiver name is not set when using the `container.create_receiver()` method**
  In earlier releases of the product, a flaw in the option processing of the `container.create_receiver()` method prevented the `name` option from taking effect.

  In this release, the `name` option works as expected.

- **ENTMQCL-617 - Scheduling new work can starve out performing what has been scheduled before**
  In earlier releases of the product, frequent calls to `schedule()` could prevent the already-scheduled work from being processed.

  In this release, the library ensures that the presence of excessive scheduled events cannot starve other events.

3.2. AMQ JMS

- **ENTMQCL-766 - `connection:forced` leads to JMSException even though reconnect is enabled**
  In earlier releases of the product, a client configured to reconnect raised an exception if there were messages sent, but not yet acknowledged, when a connection was lost.

  In this release, a client configured to reconnect attempts to resend unacknowledged messages when a new connection is established.

3.3. AMQ RUBY

- **ENTMQCL-690 - `Container.schedule()` fails if called from an event handler**
  In earlier releases of the product, the `Container.schedule()` method had no effect if it was called from inside a `MessagingHandler` callback function.

  In this release, calling `schedule()` in an event handler works as expected.

3.4. AMQ .NET

- **ENTMQCL-707 - Credit is incremented on receipt of message instead of message acknowledgment**
  In earlier releases of the product, the client issued more message credit to the sender when a message was received, rather than when it was acknowledged.

  In this release, the client issues more credit after the message is acknowledged.

CHAPTER 4. KNOWN ISSUES

4.1. AMQ C++

- ENTMQCL-693 - **Cannot send message properties with null values**
  The library throws a `conversion_error` if a null value is decoded into a `proton::scalar`. This occurs when decoding application properties that contain null values.

4.2. AMQ PYTHON

- ENTMQCL-483 - **Selectors with backslashes are invalid in non-Unicode strings**
  The `Selector` option on `Container.create_receiver()` accepts a string. If the string is not supplied as Unicode (in Python 2, `u"somestring"`), any elements escaped with backslashes might not be processed correctly.

  Workaround: Users of Python 2 should use an explicit Unicode string in filter declarations to avoid the problem.

- ENTMQCL-546 - **Transactions introduce unexpected link events**
  Starting a transaction internally opens a sending link for controlling the transaction. This special link can trigger extra application events.

  Workaround: Code using transactions should ensure link handler functions are processing the link they expect.

4.3. AMQ .NET

- ENTMQCL-794 - **Transactions do not work with .NET Core**
  Rolling back transactions when using AMQ .NET on .NET Core is not working as expected.

  Workaround: Use .NET Framework 4.5 instead of .NET Core if you require transactions.

CHAPTER 5. IMPORTANT NOTES

5.1. AMQ C++, AMQ PYTHON, AND AMQ RUBY

This release contains a change in behavior for SASL mechanism selection. The GSSAPI and GSS-SPNEGO SASL mechanisms are now disabled by default and must be explicitly enabled if desired.

To enable GSSAPI or GSS-SPNEGO, add the mechanism using the **`allowed_mechs`** transport or connection option.

5.2. AMQ C++

- **Unsettled interfaces**
  The AMQ C++ messaging API includes classes and methods that are not yet proven and can change in future releases. Be aware that use of these interfaces might require changes to your application code in the future.

  These interfaces are marked **Unsettled API** in the API reference. They include the interfaces in the **`proton::codec`** and **`proton::io`** namespaces and the following interfaces in the **`proton`** namespace.

  - **`listen_handler`**
  - **`reconnect_options`**
  - **`ssl_certificate`**, **`ssl_client_options`**, and **`ssl_server_options`**
  - **`work_queue`** and **`work`**
  - The **`on_connection_wake`** method on **`messaging_handler`**
  - The **`wake`** method on **`connection`**
  - The **`on_sender_drain_start`** and **`on_sender_drain_finish`** methods on **`messaging_handler`**
  - The **`draining`** and **`return_credit`** methods on **`sender`**
  - The **`draining`** and **`drain`** methods on **`receiver`**

  API elements present in header files but not yet documented are considered unsettled and are subject to change.

- **Deprecated interfaces**
  Interfaces marked **Deprecated** in the API reference are scheduled for removal in a future release.

  This release deprecates the following interfaces in the **`proton`** namespace.

  - **`void_function0`** - Use the **`work`** class or C++11 lambdas instead.
  - **`default_container`** - Use the **`container`** class instead.
  - **`url`** and **`url_error`** - Use a third-party URL library instead.

5.3. 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)

5.4. 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.
- **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.

5.5. UPSTREAM VERSIONS

- AMQ C++, AMQ Python, and AMQ Ruby are now based on Qpid Proton 0.24.0
- AMQ JavaScript is now based on Rhea 0.2.15
- AMQ JMS is now based on Qpid JMS 0.34.0
- AMQ .NET is now based on AMQP.Net Lite 2.1.3

CHAPTER 6. IMPORTANT LINKS

- Red Hat AMQ 7 Supported Configurations
- Red Hat AMQ 7 Component Details
- AMQ Clients 2.0 Release Notes
- AMQ Clients 1.2 Release Notes
- AMQ Clients 1.1 Release Notes

Revised on 2018-07-27 18:01:09 EDT