Red Hat AMQ 7.2

AMQ Clients 2.0 Release Notes

Release Notes for Red Hat AMQ Clients

Last Updated: 2018-06-14

Red Hat AMQ 7.2 AMQ Clients 2.0 Release Notes

Release Notes for Red Hat AMQ Clients

Legal Notice

Copyright © 2018 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at
http://creativecommons.org/licenses/by-sa/3.0/
. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux ® is the registered trademark of Linus Torvalds in the United States and other countries.

Java ® is a registered trademark of Oracle and/or its affiliates.

XFS ® is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL ® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js ® is an official trademark of Joyent. Red Hat Software Collections is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack ® Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

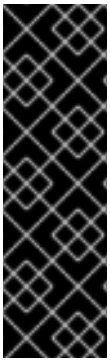
Abstract

These release notes contain the latest information about new features, enhancements, fixes, and issues contained in the AMQ Clients 2.0 release.

Table of Contents

CHAPTER 1. FEATURES

- AMQ C++ now supports multithreaded operation on Windows.
- AMQ JavaScript now supports Node.js versions 4, 6, and 8.
- AMQ JMS now offers more fine-grained control of message acknowledgment.
- AMQ .NET now supports .NET Core on Windows and Red Hat Enterprise Linux.
- This release improves the performance and efficiency of AMQP encoding and decoding.
- A new Ruby messaging client, AMQ Ruby, is now available in Technology Preview.



IMPORTANT

The AMQ Ruby client is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

For more information about the support scope of Red Hat Technology Preview features, see https://access.redhat.com/support/offerings/techpreview/.

CHAPTER 2. ENHANCEMENTS

2.1. AMQ C++

- **ENTMQCL-536 - Provide a way to intercept authentication failure**
  Authentication failures are now distinguished from other connection failures during reconnect. If authentication fails, the `on_transport_error` handler is fired with an AMQP `unauthorized-access` error. Connections configured to reconnect will terminate on this error but continue to retry after other connection errors.

- **AMQ C++ is now based on Qpid Proton 0.22.0**

2.2. AMQ JAVASCRIPT

- **AMQ JavaScript is now based on Rhea 0.2.9**

2.3. AMQ JMS

- **ENTMQCL-605 - Support acknowledging individual messages**
  The JMS `CLIENT_ACKNOWLEDGE` mode enables explicit message acknowledgment using the `Message.acknowledge()` method. This method acknowledges all the messages previously received on the session at the time it is called.

  AMQ JMS now has an "individual acknowledge" mode that makes `Message.acknowledge()` apply only to the message it is called on.

- **ENTMQCL-568 - Support a no-acknowledge mode at the session level**
  The client now offers a "no acknowledge" mode to enable presettled (at-most-once) message delivery on a per-session basis.

- **ENTMQCL-556 - Try each endpoint before delaying for next reconnect cycle**
  The client now tries to connect to each endpoint in the failover set before delaying between reconnect cycles, rather than delaying between each endpoint.

- **AMQ JMS is now based on Qpid JMS 0.31.0**

2.4. AMQ PYTHON

- **ENTMQCL-573 - Exclude user and password in logging of connection URLs**
  If a user or password are provided in a connection URL, they are now omitted in any printed representation of the connection.

- **AMQ Python is now based on Qpid Proton 0.22.0**

2.5. AMQ .NET

- **AMQ .NET is now based on AMQP.Net Lite 2.1.1**

CHAPTER 3. RESOLVED ISSUES

3.1. AMQ C++

- **ENTMQCL-584 - Kerberos authentication cannot proceed if the user option is not set**
  In earlier releases of the product, it was necessary to supply a value to the `user` connection option in order to use Kerberos authentication.

  In this release, it is no longer necessary to set the `user` option.

- **ENTMQCL-565 - Windows build does not compile the examples that require C++11**
  In earlier releases of the product, compiling the C++11 examples was not supported on Windows.

  In this release, the C++11 examples are supported on Windows.

- **ENTMQCL-600 - Epoll proactor pointer NULL in `pn_connection_wake` causes segmentation fault**
  In earlier releases of the product, it was possible for the `pn_connection_wake` operation to fail during reconnect and trigger a segmentation fault.

  In this release, the operation no longer fails.

- **ENTMQCL-601 - Container continues to run after `container.stop()` called from main thread**
  In earlier releases of the product, the `container.stop()` operation failed when called before any connections were established.

  In this release, the operation can safely be called before connecting.

3.2. AMQ JMS

- **ENTMQCL-571 - Missed connect error on start can lead to hung failover reconnect cycle**
  In earlier releases of the product, it was possible for the client to miss a connection error and hang during reconnect.

  In this release, the client no longer hangs due to the missed error.

- **ENTMQCL-606 - Connection fails using Kerberos against Interconnect**
  In earlier releases of the product, the client did not handle an empty SASL challenge from AMQ Interconnect during Kerberos authentication. This prevented the client from establishing the connection.

  In this release, the client handles the empty challenge, and the connection can be established.

3.3. AMQ PYTHON

- **ENTMQCL-620 - Application property keys are incorrectly encoded as AMQP binary**
  In earlier releases of the product, the client encoded the keys of message application properties as AMQP binary data. AMQP specifies that they be AMQP string data instead.

  In this release, the keys are encoded as AMQP strings.

3.4. AMQ .NET

- **ENTMQCL-500 - Receive does not raise exception when link closed with error**
  In earlier releases of the product, it was possible for pipelined protocol state events to prevent the client from raising link errors to the API user.

  In this release, the link errors are raised when the link is closed.

CHAPTER 4. KNOWN ISSUES

4.1. AMQ C++

- **ENTMQCL-604 - Receiver name is not set when using the `container.create_receiver()` method**
  Due to a flaw in the option processing of the `container.create_receiver()` method, the receiver name is not set when using the `name` option.

  Workaround: Set the receiver name using the `name` option on the `connection.create_receiver()` method instead of on the `container.create_receiver()` method.

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 RUBY

- **ENTMQCL-690 - `Container.schedule()` fails if called from an event handler**
  The `Container.schedule()` method will have no effect if it is called from inside a `MessagingHandler` callback function.

  Workaround: Call `schedule()` from outside the `MessagingHandler` callback functions.

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.
- **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.

5.3. 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.

CHAPTER 6. IMPORTANT LINKS

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

Revised on 2018-06-14 11:27:14 EDT