Advanced Message Queuing Protocol (AMQP) Management Version 1.0

Working Draft 16   
5 July 2019

Technical Committee:

[OASIS Advanced Message Queuing Protocol (AMQP) TC](http://www.oasis-open.org/committees/amqp/)

Chairs:

Clemens Vasters ([clemensv@microsoft.com](mailto:Ram.Jeyaraman@microsoft.com)), [Microsoft](http://www.microsoft.com/)

Robert Godfrey ([rgodfrey@redhat.com](mailto:rgodfrey@redhat.com)), Red Hat

Editors:

Clemens Vasters (clemensv@microsoft.com)

Related work:

This specification is related to:

* *OASIS Advanced Message Queuing Protocol (AMQP) Version 1.0 Part 0: Overview*. 29 October 2012. OASIS Standard. <http://docs.oasis-open.org/amqp/core/v1.0/os/amqp-core-overview-v1.0-os.html>.

Abstract:

This specification defines a set of management operations for entities inside of AMQP containers as a layer on top of the AMQP 1.0 protocol.

Management operations are performed by sending command messages to management nodes. Management commands are sent in the body of messages encoded using the AMQP Type System. The results of management operations are returned using the AMQP Request/Response pattern. This specification defines four standard operations which are expected to be common to all types of manageable entities: Create, Read, Update and Delete. Additionally manageable entities may support entity specific operations.

Management nodes also support discovery operations. These operations allow discovery of: manageable entities, the operations which can be performed on them, and other management nodes within the system.

Status:

This [Working Draft](http://www.oasis-open.org/committees/process.php#dWorkingDraft) (WD) has been produced by one or more TC Members; it has not yet been voted on by the TC or [approved](http://www.oasis-open.org/committees/process.php#committeeDraft) as a Committee Draft (Committee Specification Draft or a Committee Note Draft). The OASIS document [Approval Process](http://www.oasis-open.org/committees/process.php#standApprovProcess) begins officially with a TC vote to approve a WD as a Committee Draft. A TC may approve a Working Draft, revise it, and re-approve it any number of times as a Committee Draft.

Initial URI pattern:

<http://docs.oasis-open.org/amqp/amqp-man/v1.0/csd01/amqp-man-v1.0-csd01.doc>

(Managed by OASIS TC Administration; please don’t modify.)

Copyright © OASIS Open 2014. All Rights Reserved.

All capitalized terms in the following text have the meanings assigned to them in the OASIS Intellectual Property Rights Policy (the "OASIS IPR Policy"). The full [Policy](http://www.oasis-open.org/policies-guidelines/ipr) may be found at the OASIS website.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published, and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this section are included on all such copies and derivative works. However, this document itself may not be modified in any way, including by removing the copyright notice or references to OASIS, except as needed for the purpose of developing any document or deliverable produced by an OASIS Technical Committee (in which case the rules applicable to copyrights, as set forth in the OASIS IPR Policy, must be followed) or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by OASIS or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and OASIS DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY OWNERSHIP RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Table of Contents

[1 Introduction 6](#_Toc12625853)

[1.1 Terminology 6](#_Toc12625854)

[1.2 Normative References 6](#_Toc12625855)

[1.3 Non-Normative References 7](#_Toc12625856)

[2 Overview 8](#_Toc12625857)

[2.1 Discovery Document 8](#_Toc12625858)

[2.1.1 Collections 8](#_Toc12625859)

[2.1.2 Types 9](#_Toc12625860)

[2.1.3 Configuration 9](#_Toc12625861)

[2.1.4 Operations 9](#_Toc12625862)

[2.2 Generic Factory 10](#_Toc12625863)

[2.3 Navigating Collections 11](#_Toc12625864)

[2.4 "Self" Management Address 11](#_Toc12625865)

[2.5 Manageable Sub-Entities 11](#_Toc12625866)

[3 Management Node 12](#_Toc12625867)

[3.1 AMQP Connection Capability 12](#_Toc12625868)

[3.2 HTTP Discovery 12](#_Toc12625869)

[3.3 Discovery Document 12](#_Toc12625870)

[3.3.1 Collections Type 13](#_Toc12625871)

[3.3.1.1 Entity-Collection-Ref Type 13](#_Toc12625872)

[3.3.2 Entity-Type-Collection-Ref Type 13](#_Toc12625873)

[3.3.3 Configuration-Ref Type 14](#_Toc12625874)

[3.3.4 Operations Type 14](#_Toc12625875)

[3.3.4.1 Operation Type 14](#_Toc12625876)

[4 Entity Types and Collections 16](#_Toc12625877)

[4.1 Entity Collection 16](#_Toc12625878)

[4.1.1 Entity Type 16](#_Toc12625879)

[4.1.2 Entity Type Versioning 16](#_Toc12625880)

[4.1.3 Entity Type Constraints and Notation 16](#_Toc12625881)

[4.1.4 Entity Base Type 17](#_Toc12625882)

[4.2 Entity Metatype Collection 18](#_Toc12625883)

[4.2.1 Entity Metatype 18](#_Toc12625884)

[4.3 Entity Archetypes 19](#_Toc12625885)

[4.3.1 $anycast 19](#_Toc12625886)

[4.3.2 $multicast 20](#_Toc12625887)

[4.3.3 $queue 20](#_Toc12625888)

[4.3.4 $topic 20](#_Toc12625889)

[5 Resource Operations 21](#_Toc12625890)

[5.1 Content 21](#_Toc12625891)

[5.1.1 JSON Content 21](#_Toc12625892)

[5.2 Common Request Rules 21](#_Toc12625893)

[5.3 Management Node 22](#_Toc12625894)

[5.4 Entity Type Collection 22](#_Toc12625895)

[5.5 Entity Collections 22](#_Toc12625896)

[5.5.1 Enumerating Entities 22](#_Toc12625897)

[5.5.2 Creating Entities 22](#_Toc12625898)

[5.6 Entities 23](#_Toc12625899)

[5.6.1 Getting the Entity Description 23](#_Toc12625900)

[5.6.2 Updating the Entity 23](#_Toc12625901)

[5.6.3 Deleting the Entity 23](#_Toc12625902)

[5.6.4 Entity Discovery Document 24](#_Toc12625903)

[6 Custom Operations 25](#_Toc12625904)

[7 Examples 26](#_Toc12625905)

[8 Conformance 27](#_Toc12625906)

[9 Appendix – OpenAPI Definition 28](#_Toc12625907)

[Acknowledgments 38](#_Toc12625908)

[Revision History 40](#_Toc12625909)

# Introduction

The Advanced Message Queuing Protocol is an open internet protocol for business messaging.

This specification, AMQP Management, enables discovery and management of the internal topology and entities of an AMQP [container](http://docs.oasis-open.org/amqp/core/v1.0/os/amqp-core-transport-v1.0-os.html#doc-idp2176) [AMQP].

Such entities might include queues, topics, or streams, and their relationships inside a message broker, as well as users, groups, and related access control rules. Not all managed entities need to be AMQP-addressable.

Unlike prior attempts to provide an interoperable management model for messaging systems, such as in early drafts of AMQP (0.9 and earlier) or in the Java Message Service (JMS) specifications, this specification does not mandate a specific topology model, but is designed to provide flexibility and extensibility.

While AMQP is popular for queueing and pub-sub message brokers, it is also being used, for instance, in IoT gateways, telemetry ingestion brokers, relays and routers, event dispatchers, and for peer-to-peer transfers. This management model embraces the diversity of these use-cases and permits further evolution.

The management operations are defined as a resource-oriented API that embodies the principles of "Representational State Transfer" [REST]. The API builds on the elements of HTTP 1.1 [RFC7231], with the AMQP implementation being realized via the "HTTP over AMQP" [AMQPHTTP] protocol mapping.

The operation payloads are defined as simple key/value pair maps, with defined encodings for the AMQP type system [AMQP] and for JSON [JSON].

Therefore, the AMQP Management model can implemented either over AMQP with AMQP-encoded data, or over HTTP with JSON-encoded data, or both. This aligns with many of today’s messaging systems providing control-plane access over HTTP for easier integration with web-based tooling.

## Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in **[RFC2119].**

**Entity Type:** A class of entities that can be managed, for instance a "queue".

**Entity:** An instance of an entity type on which management operations can be performed, e.g., a queue "queue1".

**Entity Description:** Metadata describing the entity. The content of the description is defined by the entity type.

**Management Operation**: An action that can be performed on a manageable entity.

**Management Node**: An AMQP node to which management operations are addressed.

**Management node**: The AMQP container in which entities are being managed.

## Normative References

[AMQP] Godfrey, Robert; Ingham, David; Schloming, Rafael, "Advanced Message Queueing Protocol (AMQP) Version 1.0", October 2012. OASIS Standard. <https://www.oasis-open.org/standards#amqpv1.0>

[AMQPHTTP] Vasters, Clemens, "HTTP over AMQP".

[OpenAPI] Miller, D., Whitlock, J., Gardner, M., Ralphson, M., Ratovsky, R., Sarid, U., "OpenAPI Specification". Linux Foundation. http://spec.openapis.org/oas/v3.0.2

[BCP47] Phillips, A., Ed., Davis, M., Ed., "Tags for Identifying Languages", September 2009. <http://tools.ietf.org/html/bcp47>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997. <http://www.ietf.org/rfc/rfc2119.txt>.

[RFC2606] Eastlake, D., Panitz, Al, "Reserved Top Level DNS Names", RFC2606, June 1999. <http://tools.ietf.org/html/rfc2606>.

**[RFC4648]**  Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <<https://www.rfc-editor.org/info/rfc4648>>.

[RFC7231] Fielding, R., Ed., and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <<https://www.rfc-editor.org/info/rfc7231>>.

[CORS] Cross-Origin Resource Sharing, A. van Kesteren, Editor, W3C Recommendation, January 16, 2014, http://www.w3.org/TR/2014/REC-cors-20140116/. Latest version available at <http://www.w3.org/TR/cors/>.

[JMS] Java Message Service Specification 2.0. Oracle. <https://www.oracle.com/technetwork/java/jms/index.html>

## Non-Normative References

[REST] Fielding, Roy Thomas (2000). "Chapter 6: Experience and Evaluation". Architectural Styles and the Design of Network-based Software Architectures (Ph.D.). University of California, Irvine.

# Overview

AMQP Management provides a resource-oriented API for managing all aspects of an AMQP container.

The API builds on the HTTP content and semantics model [RFC7231] and can be implemented over AMQP using the HTTP-over-AMQP [AMQPHTTP] mapping and directly over HTTP 1.1 or newer versions.

There are no restrictions on combining this API with other HTTP features. For instance, an implementation might benefit from implementing Cross-Origin Resource Sharing [CORS] to allow browser-based management applications to interact with the resources directly from the client context.

The entry-point for all management interactions with an AMQP container is the "management node". The management node has a well-known default address, but that address can be overridden by the container. Upon request, the management node returns a discovery document that provides references to the container’s manageable entity collections.

Any entity MAY have a private management node that allows for managing its internal structure and that management node has the same functions as the "main" management node, but it is scoped to the entity.

The design goal of the AMQP Management model is for it to be extensible and adaptive to the needs of a wide variety of messaging infrastructures, and suitable for use with interactive as well as programmatic management tools.

## Discovery Document

The management node (see section 3) answers to GET requests and returns a discovery document. The discovery document is a map.

The discovery map has four well-known entries, "collections", "types", "configuration", and "operations".

The value of "collections" is a map that contains references to collections of manageable entities.

The value of each entry is a map that includes the *address* of the collection’s endpoint in the container and a localizable *label*.

The value of "types" is a map that includes the *address* of an endpoint that returns a list of all entity types supported by this management node and a localizable *label*.

The value of "configuration" is a map that includes the *address* of an endpoint providing access to a configuration document and a localizable *label*.

The value of "operations" is a map that enumerates the operations that this management node supports. The key identifies the operation name and the value is a map that includes the *address*, the *request* and response arguments, and a localizable *label*.

### Collections

In the "collections" map (see 3.2.1), the reserved "entities" key identifies the main collection of all entities in the container.

In the following example, "streams", and "queues" represent container-specific entity types, and "priority-queues" is a filtered view on the collection of queues where the priority feature is enabled.

The exemplary German-language labels are for displaying the collections in a localized German user experience. The locale is based on negotiation at the transport layer or in the request.

{

"collections" : {  
 *"entities" : {  
 "label" : "Entitäten",  
 "address", "entities"  
 }*,  
 "streams" : {  
 "label" : "Datenströme",  
 "address", "streams"  
 },  
 "queues" : {  
 "label" : "Warteschlangen",  
 "address", "queues"  
 },  
 "priority-queues" : {  
 "label" : "Prioritätswarteschlangen",  
 "address", "priority-queues"  
 }  
 }  
}

The base-address for the addresses in the collection map is the management node’s address.

In the example, all collection addresses are relative to the management node, meaning that if the management node address were "$management", the effective address of the "queue" collection were "$management/queues". The collection addresses could also be absolute paths or absolute URIs.

### Types

The “types” entry in the discovery document points to an endpoint that returns the available entity types as a list of entity-metatype objects.

{  
 "types" : {  
 "label" : "Typen",  
 "address", "types"  
 }  
}

### Configuration

The “configuration” entry in the discovery document points to a configuration endpoint for the management node’s scope. For a discovery document for the AMQP container’s maion management node, the address might point to a configuration endpoint that allows for managing configuration aspects of the container. The details of the endpoint are application-specific.

{  
 "configuration" : {  
 "label" : "Server-Konfiguration",  
 "address", "config"  
 }  
}

### Operations

The "operations" map contains references to management operations that are not easily or intuitively modeled as simple GET/PUT/POST/DELETE resource operations. An example for this might be temporarily enabling or disabling a server endpoint. While this can arguably be performed by editing a document and setting a configuration value, some implementations might find doing so too obscure.

Example:

{

"operations" : {  
 "enable" : {  
 "label" : "Server Aktivieren",  
 "address", "operations/enable"  
 },  
 "disable" : {  
 "label" : "Server Deaktivieren",  
 "address", "operations/disable"  
 }

}  
}

The example shown here does not define any details for the operation arguments or expected response, which means the request MUST be executed as a POST request against the given address without any parameters or entity body and success is indicated by a 2xx status code.

If arguments are required or a response is expected, the "request" and/or "response" properties contain a map with the name of the arguments and their AMQP types:

{

"operations" : {  
 "setState" : {  
 "label" : "Server Status ändern",  
 "address" : "operations/setstate",  
 "request" : {  
 "state": "integer"  
 },  
 "response": {  
 "state": "integer"  
 }

}  
 }  
}

## Generic Factory

New entities are created with a POST request against the main “entities” collection, which acts as a generic factory that can not only create instances of explicitly supported entity types, but can also resolve archetypes to the appropriate concrete implementation types.

In section 4.2.7, this specification defines the "$anycast", "$multicast", "$queue" and "$topic" archetypes, which describe a generic queue entity and a generic topic entity similar to the definitions found in JMS [JMS].

If an implementation chooses to support any or all of these archetypes, it will accept a request asking to create one of the these archetypes and then respond with an instance of the concrete entity type that best matches the archetype definition. This allows a completely generic client to create an archetypical queue or topic without any further knowledge of the accessed product.

When successful, the 201 response to a POST request against the main collection will contain the full description of the created entity including its concrete type and a "Location" header that points to the "self" management address of the new entity, typically residing within one of the type-specific collections.

Nonnormative Example:

|  |  |
| --- | --- |
| Request | Response |
| POST HTTP/1.1 /$management/entities Content-Type: application/amqp-management+json; type=entity {  "type" : "$queue",  "name" : "myqueue" } | HTTP/1.1 201 Created Location: /$management/basicqueues/3939473  Content-Type: application/amqp-management+json; type=basicqueue  {  "type" : "basicqueue",  "name" : "myqueue",  "target" : "myqueue-in",  "source" : "myqueue",  "id": "3939473",  "self" : "/$management/basicqueues/3939473"  } |

## Navigating Collections

Collections are returned as response to GET requests on the collection address (see X.X). The returned payload is list of entity descriptions. Paging is supported (see 5.5)

## "Self" Management Address

Each entity has a "self" management address where it can be managed directly. Requests to the address will typically return the entity description on GET, might allow for modification of properties with PUT, and will remove the entity in response to DELETE. The supported methods can be discovered with the OPTIONS method.

## Manageable Sub-Entities

Any entity MAY have a local management node that returns a discovery document pointing to collections of manageable entities within the entity itself and to entity-scoped operations. Sub-entities MAY have further sub-entities. The address of the management node is optionally held in the "management" property of the entity description. The base address for the management address is the "self" address.

{  
 "type" : "topic",  
 "name" : "mytopic",  
 "target-address" : "mytopic",  
 "id": "3939474",  
 "self" : "/$management/basicqueues/3939474"  
 "management" : "$management"   
}

For a topic with dependent durable subscriptions, the discovery document might point to the collection of those subscriptions:

{  
 "collections": {  
 "entities" : {  
 "label" : "Entitäten",  
 "address", "entities"  
 },  
 "subscriptions" : {  
 "label" : "Dauerhafte Abonnements",  
 "address", "subscriptions"  
 }

}

}

# Management Node

This section formally defines the management node as introduced in the overview section.

## AMQP Connection Capability

On connection establishment, a partner MUST indicate whether it supports AMQP management through the exchange of connection capabilities (see Section 2.7.1 [AMQP]).

|  |  |
| --- | --- |
| Capability Name | Definition |
| AMQP\_MANAGEMENT\_V1\_0 | If present in the offered-capabilities field of the open frame, the sender supports AMQP Management and offers a management node. |

The container MAY tell the partner the address of the management node with a connection property:

|  |  |  |
| --- | --- | --- |
| Connection property name | Type | Definition |
| managementnode | address | Address of the management node. If omitted or null-valued, the management node address defaults to $management. |

The container offering the AMQP\_MANAGEMENT\_V1\_0 capability either MUST provide a management node address in the connection property or it MUST use the reserved name $management for the management node.

## HTTP Discovery

If the API is implemented over HTTP 1.1 or later, the management node SHOULD be discoverable using a GET request on the $management path underneath the container’s application path, which is often at the site root. The GET request MAY redirect to a different path for the management node.

The management node address MAY also be discoverable

* as the value of HTML metadata in form of a “meta” tag with the name “amqp-managementnode” contained in a document returned by a GET from the container’s application path.
* as the value of a custom HTTP response header named “amqp-managementnode” returned when requesting a document returned by a GET from the container’s application path.

## Discovery Document

The discovery document is returned in response to a "GET" request sent to the management node address.

The rules for the request and response operations are defined in section 6.

The discovery document provides information about the following elements in the scope of this management node

* The addresses of the collections of manageable entities
* The collection of supported entity types
* The operations supported by the management node
* The address of the configuration endpoint of the management node

The normative schema for this document is given in the AMQP type system notation used in the AMQP 1.0 specification [AMQP]. A non-normative JSON Schema representation is included in the implementation notes section.

The discovery document is a map where the keys are restricted to be of type symbol (this excludes the possibility of a null key), and which has three reserved entries. Applications MAY add further entries.

<type name="discovery-document" class="restricted" source="map">  
 <field name="collections" type="collections"   
 mandatory="true" multiple="false" />  
 <field name="types" type="entity-type-collection-ref"   
 mandatory="true" multiple="false" />  
 <field name="operations" type="operations"   
 mandatory="false" multiple="false" />  
 <field name="configuration" type="configuration-ref"   
 mandatory="false" multiple="false" />  
</type>

### Collections Type

The "collections" type is a is a map where the keys are restricted to be of type symbol, and which has one predefined, required entry:

* "entities": The value MUST be of type "entity-collection-ref" points to the main entity collection that also acts as the generic factory.

<type name="collections" class="restricted" source="map">  
 <field name="entities" type="entity-collection-ref"   
 mandatory="true" multiple="false" />  
</type>

The values for entries for any other key MUST be of type "entity-collection-ref". The key might reflect an entity type name or an archetype name or an application-defined subset of entities but has no semantic meaning.

#### Entity-Collection-Ref Type

The type "entity-collection-ref" is an extensible map where the keys are restricted to be of type symbol and which has two required field entries:

* the *address* of the entity collection in the container. The address MUST refer to an endpoint for an [entity collection](#_Entity_Collection) (see 4.1).
* a localizable *label*.

<type name="entity-collection-ref" class="restricted" source="map">  
 <field name="label" type="string"   
 label="Localized label for this collection"   
 mandatory="false" multiple="false" />  
 <field name="address" type="address"   
 label="Address of this collection"   
 mandatory="true" multiple="false" />  
</type>

### Entity-Type-Collection-Ref Type

The type "entity-type-collection-ref" is an extensible map where the keys are restricted to be of type symbol and which has two required field entries:

* the *address* of the metatypes collection in the container. The address MUST refer to an endpoint for a types collection (see [4.2](#_Entity_Metatype_Collection)).
* a localizable *label*.

<type name="entity-type-collection-ref" class="restricted" source="map">  
 <field name="label" type="string"   
 label="Localized label for this collection"   
 mandatory="false" multiple="false" />  
 <field name="address" type="address"   
 label="Address of this collection"   
 mandatory="true" multiple="false" />  
</type>

### Configuration-Ref Type

The type "configuration-ref" is an extensible map where the keys are restricted to be of type symbol and which has two required field entries:

* the *address* of the configuration endpoint in the container.
* a localizable *label*.

<type name="configuration-ref" class="restricted" source="map">  
 <field name="label" type="string"   
 label="Localized label for this configuration reference"   
 mandatory="false" multiple="false" />  
 <field name="address" type="address"   
 label="Address of the configuration document"   
 mandatory="true" multiple="false" />  
</type>

### Operations Type

The value of "operations" is a list of "operation" that enumerates the operations that this management node supports. The list enumerates all management operations that are not easily modeled as resource operations as defined in section 5 or that require explicit representation in a control user experience.

An example might be to temporarily enable or disable a server endpoint. While these operations can be modeled as an operation that modifies a configuration property, these operations might be considered so important that they are special-cased and explicitly presented as operations.

An management node MUST NOT offer operations that are equivalent to or are trivial specializations of existing resource operations, e.g. "create\_queue" or "delete\_topic".

This specification prescribes no specific operation.

<type name="operations" class="restricted" source="list">

#### Operation Type

The type "operation" is defined as a map with the following entries:

* the *name* of the operation.
* a localizable *label* that a tool might display for the operation
* the address of the operation endpoint
* an optional formal description of request and response, using the JSON representation of an [OpenAPI 3.0 Operation Object](http://spec.openapis.org/oas/v3.0.2#operation-object) [OpenAPI]

Details about the request and response types and about how operations are invoked is described in section 6.

<type name="operation" class="restricted" source="map">  
 <field name="name" type="symbol"   
 label="Name of the operation"   
 mandatory="false" multiple="false" />  
 <field name="label" type="string"   
 label="Localized label for this collection"   
 mandatory="false" multiple="false" />  
 <field name="address" type="address"   
 label="Address of this operation"   
 mandatory="true" multiple="false" />  
 <field name="request" type="request"   
 label="Request definition"   
 mandatory="false" multiple="false" />  
 <field name="response" type="response"   
 label="Response definition"   
 mandatory="false" multiple="false" />  
</type>

# Entity Types and Collections

The objects under management inside a container are called "entity". The definition of the set of properties used to describe an entity is called "entity type".

All entity type definitions share a common base definition to allow for uniform handling in tooling, such as displaying a list of entities.

An entity type definition can be expressed in an "entity metatype" for use in tooling and is exposed through the entity type collection on the management node.

Entities in the sense of this specification are all elements of the internal topology of an AMQP container that can be managed. Entities can be configured via management operations that carry entity descriptions. Entity descriptions are based on entity type definitions.

An entity might be an addressable AMQP node, like a queue, or it might be a type of entity that is not AMQP-addressable, like a user account.

The operations permitted on an entity depend on its type; some entities or some elements of entities might be read-only; some entities might be "built-in" and cannot be created or deleted.

## Entity Collection

An entity collection is an list of entity type instances. Entity collections are manageable via entity collection endpoints (see 4.2).

<type name="entity-collection" class="restricted" source="list">

### Entity Type

An entity type defines how a particular kind of entity is described and configured. All entity types are derived from the common entity base type defined in this section.

### Entity Type Versioning

A container MAY support multiple versions of the same entity type concurrently.

This is specifically useful in cloud services where a core construct like a "queue" undergoes a long lifecycle where features are introduced and retired, and the underlying implementation might indeed change substantially for new deployments while existing deployments remain unchanged.

In such scenarios, version 1 of a "queue" might coexist with a version 2 of "queue" and clients will need continued access to features on existing version 1 queues that have been deprecated or outright removed from version 2 queues.

Encoding the version inside the type name or forcing the "next version" queue to take on a wholly new type identity is problematic because it will generally prevent seamless upgrades for clients that do not have a dependency on deprecated niche features.

Whether a type ought to be a new version of an existing type or an entirely new type depends on whether the desired behavior is that the new version does supersede the prior version when the type name is given and no version is specified. If an implementation needs to permanently provide coexistent entities that are very similar but where one does not supersede the other in the default case, it should maintain separate types.

### Entity Type Constraints and Notation

The notation for entity types leans on the AMQP [composite type notation model](http://docs.oasis-open.org/amqp/core/v1.0/os/amqp-core-types-v1.0-os.html#doc-idp108672) [AMQP]. The notation is provided for consistency in defining entity types.

Different from AMQP composite types, the defined properties for entity types are encoded as map entries so that receivers are not required to have schema access to decode the data.

All entity descriptions are extensible. Any implementation MAY add extra properties to a description, even if that metadata is not formally declared or documented in the entity type.

All concrete entity type definitions MUST be derived from the "entity" base type, which provides common properties that allow for generic handling of entities in tools.

<type name="..." class="entity" source="entity" ...>   
 ...   
 <property name="..." type="..."   
 default="..." label="..." mandatory="true|false" />   
 ...   
</type>

Figure 2.1: Property Deﬁnitions

A property is defined with the following attributes:

* **name** – Name of the field. REQUIRED.
* **type** – The AMQP data type of the ﬁeld. It might name a speciﬁc type, in which case the values are restricted to that type, or it might be the special character "\*", in which case a value of any type is permitted. In the latter case the range of values might be further restricted by the requires attribute. Composite AMQP types SHOULD NOT be used. Maps, arrays, and lists are permitted. REQUIRED.
* **default** – The default value for the ﬁeld if no value is given. OPTIONAL.
* **label** – A short description of the ﬁeld. OPTIONAL.
* **mandatory** – "true" if omitting the field or a null value for the ﬁeld is not permitted. OPTIONAL. Defaults to false.

### Entity Base Type

The base type "entity" MUST be used as the source type for all entity type definitions. The base type contains properties that all MUST be understood for all entities in order to provide uniform handling for tooling:

* **id** – An immutable, case-sensitive, management-node assigned symbol identifying the entity. The identity MUST be unique within the scope of the management node. REQUIRED.
* **etag** – The management-node assigned entity tag of this entity. The tag is used to prevent concurrent access conflicts while either updating or deleting entities. REQUIRED.
* **self** – the address for managing the entity. OPTIONAL.
* **managementnode** – the address of the entity’s local management discovery document. OPTIONAL.
* **type** – A case-sensitive string identifying the entity type. REQUIRED.
* **typeversion** – Entity type version. If this field is omitted, the default assumption is that the latest version of the type is used. OPTIONAL.

<type name="entity" class="restricted" source="map">   
 <property name="id" type="symbol"   
 label="Identity of this entity"   
 mandatory="false" />  
 <property name="etag" type="symbol"   
 label="Version-etag of this entity"   
 mandatory="false" />   
 <property name="type" type="symbol"   
 label="Type of this entity"   
 mandatory="true" />  
 <property name="typeversion" type="symbol"   
 label="Type version of this entity"   
 mandatory="false"/>  
 <property name="self" type="address"   
 label="Address for managing the entity policy"   
 mandatory="false"   
 multiple="false" />  
 <property name="managementnode" type="address"   
 label="Local management node address"   
 mandatory="false"   
 multiple="false" />  
</type>

## Entity Metatype Collection

An entity type collection is an array of entity meta-type instances. The entity type collection allows enumeration of all the types that a management node supports and is always read-only.

<type name="entity-type-collection" class="restricted" source="list">

### Entity Metatype

Because dynamic management clients require access to entity type definitions, the entity metatype allows packaging an entity type definition into an AMQP type.

* name – Name of the entity type
* version – Version of the entity type
* properties – Properties defined for this type

<type name="entity-type" class="restricted" source="map">   
 <field name="name" type="symbol"   
 label="Name of this type"   
 mandatory="true" multiple="false" />  
 <field name="version" type="symbol"   
 label="Version of this type"   
 mandatory="true" multiple="false" />  
 <field name="properties" type=" entity-property"   
 label="Properties defined for this type"   
 mandatory="true" multiple="true" />   
</type>

The "entity-property" type used for properties within the entity metatype is defined as:

* **name** – Name of the property
* **type** – AMQP type of the property
* **label** – Descriptive label of the property.
* **default** – default value to be used when the property is omitted.
* **mandatory** – Indicates whether the property is mandatory.
* **multiple** – multiple flag. Controls whether the value is an array of multiple values.

<type name="entity-properties" class="restricted" source="map">   
 <field name="name" type="symbol"   
 label="Name of the property"   
 mandatory="true" multiple="false" />  
 <field name="type" type="symbol"   
 label="AMQP type"   
 mandatory="true" multiple="false" />  
 <field name="label" type="string"   
 label="Label"   
 mandatory="false" multiple="false" />  
 <field name="default" type="\*"   
 label="Default value"   
 mandatory="false" multiple="false" />  
 <field name="mandatory" type="boolean"   
 label="Mandatory"   
 default="false" mandatory="false" multiple="false" />  
 <field name="multiple" type="boolean"   
 label="Multiple values permitted"   
 default="false" mandatory="false" multiple="false" />  
</type>

## Entity Archetypes

Even though AMQP generally does not impose topology details on messaging infrastructures, this specification introduces four optional entity archetypes for use with the "[generic factory](#_Generic_Factory)" (see 2.2) mechanism:

$anycast and $multicast as archetypes for entities with a particular distribution pattern and $queue and $topic as derived archetypes with storage implications.

Those entity archetypes provide a way to create instances of the two most commonly used messaging pattern constructs through the AMQP management interface without any specific product knowledge.

### $anycast

The $anycast archetype can be used with the generic factory to create an entity with the following characteristics:

* The entity MUST provide a "target" address that accepts message transfers
* The entity MUST provide a "source" address that originates message transfers.
* All messages transferred into the entity via the “target” address SHOULD be made available for delivery out of the "source" address.
* The default source distribution-mode [3.5.2, AMQP] MUST be "move" and all delivery state changes for a message affect a shared copy of the message, independent of the number of active source links, meaning that each message is acquired for delivery over one link only.
* Browsing of the entity’s message content MAY be implemented with distribution-mode "copy", whereby the delivery state of messages MUST remain unaffected.

There is no message storage implication with the $anycast archetype and messages might not be retained for later transfers if no receiving links are attached to the “source” address.

Support for all other potential capabilities are an application-specific choice, even those with explicit mention in the core AMQP specification such as durability, priority, or transaction support.

When the $anycast archetype is used with the generic factory, the container selects a concrete entity type, creates and returns it. The concrete entity type MUST provide "source" and "target" properties when providing the $anycast archetype. The supported types for "source" and "target" are identical to those for the AMQP attach performative [2.7.3, 3.5 AMQP].

<type name="..." class="restricted" source="map" provides="$anycast">   
 <property name="source" type="\*"  
 label="Source address"   
 mandatory="true" />  
 <property name="target" type="\*"   
 label="Target address"   
 mandatory="true" />   
</type>

### $multicast

The $multicast archetype can be used with the generic factory to create an entity with the following characteristics:

* The entity MUST provide a "target" address that accepts message transfers
* The entity MUST provide a "source" address that originates message transfers.
* All messages transferred into the entity via the “target” address SHOULD be made available for delivery out of the "source" address.
* The default source distribution-mode [3.5.2, AMQP] MUST be "copy" and all message delivery state changes affect private message copies associated with the link terminus, meaning that each link can acquire independent copies of each arriving message.

Support for all other potential capabilities are an application-specific choice, even those with explicit mention in the core AMQP specification such as durability, priority, filters, or transaction support.

When the $multicast archetype is used with the generic factory, the container selects a concrete entity type, creates and returns it. The concrete entity type MUST provide "source" and "target" properties when providing the $multicast archetype. The supported types for "source" and "target" are identical to those for the AMQP attach performative [2.7.3, 3.5 AMQP].

<type name="..." class="restricted" source="map" provides="$multicast">   
 <property name="source" type="\*"  
 label="Source address"   
 mandatory="true" />  
 <property name="target" type="\*"   
 label="Target address"   
 mandatory="true" />   
</type>

### $queue

The $queue archetype is a derivative of the $anycast archetype. All characteristics enumerated for the $anycast archetype apply, and also:

* The entity SHOULD store messages transferred via the target address for later retrieval via the “source” address, even when no receiving links are attached to the “source” address.

### $topic

The $topic is a derivative of the $multicast archetype. All characteristics enumerated for the $multicast archetype apply, and also:

* The entity MUST support [terminus durability](http://docs.oasis-open.org/amqp/core/v1.0/os/amqp-core-messaging-v1.0-os.html#type-terminus-durability) at the “configuration” level. [AMQP, 3.5.5]

# Resource Operations

This section defined standard operations for interacting with entities and management nodes. All operations are defined as HTTP [RFC7231] request/response interactions, requested by the managing client and responded to by the management node.

The mapping HTTP requests and responses to AMQP and correlation of requests and responses is performed according to the rules of the "HTTP over AMQP" mapping [AMQPHTTP]. Using this mapping, all described operations can be performed peer-to-peer, or via multi-hop routes, using paired links, or even via store-and-forward or queue intermediaries.

Implementers MAY define their own operations, but they MUST NOT define custom operations that are trivial specializations of the predefined operations, e.g. defining a "CREATE-QUEUE" operation is not permitted.

The sender indicates the desired locale for localizable content with the ["Accept-Language"](https://tools.ietf.org/html/rfc7231#section-5.3.5) HTTP header [5.3.5, RFC7231]. If not sent and if the connection has been established over HTTP, the effective locale is the one [negotiated for the AMQP connection](http://docs.oasis-open.org/amqp/core/v1.0/os/amqp-core-transport-v1.0-os.html#type-open) [2.7.1, AMQP].

## Content

When using AMQP encoding for the entity body of requests or responses using the types defined in this specification, clients MUST use the "application/amqp-management+amqp" media type in the content type declaration.

When using JSON encoding, clients MUST use the "application/amqp-management+json" media type.

The type of the top-level entity enclosed in the entity body SHOULD be added as a content type parameter using the "type" key, e.g. "application/amqp-management+json; type=entity-collection".

### JSON Content

All types in this document are defined with AMQP type definitions, but they can also be encoded as JSON [RFC8259] following these rules:

* AMQP maps map to JSON objects, with map-entry keys becoming member names and map-entry values becoming member values.
* AMQP lists map to JSON arrays
* AMQP string expressions map to JSON string expressions
* AMQP binary expressions map to JSON string expressions using Base64 encoding compliant with RFC4648, section 4 [RFC4648].
* AMQP timestamp expressions map to JSON string expressions using the RFC3339 format
* AMQP numeric expressions map to JSON number expressions
* AMQP boolean expressions map to JSON boolean expressions
* AMQP null expressions map to JSON null expressions

All JSON expressions MUST be handled based on the AMQP type system and this mapping.

For instance, if a JSON-encoded entity description contains a member which is defined as to carry an AMQP timestamp value in the schema, the JSON value MUST be a string and it MUST carry a RFC3339 timestamp expression.

## Common Request Rules

The following headers SHOULD be set for all requests:

* **Accept-Language** to indicate the desired locale for descriptive text and labels.
* **Accept** to indicate the content type preferences for the response (see 5.1)
* For GET only: **If-None-Match** [RFC7232]if the document or collection has already been retrieved once and has been cached, and the previously returned **ETag** value had been stored. The ETag value is then used for this header to allow the server to respond with 304 indicating that the content of the document or collection hasn’t changed since it was last retrieved.

The response SHOULD carry the **ETag** header to indicate the version of this document. If ETag is returned, the ETag MUST change whenever the content of the document or collection changes.

If the client sends the "Accept" header and the desired content type is not available, the request SHOULD fail with a 406 status code without handling the request.

For requests carrying an entity body, the Content-Type header MUST be set to the content type used for the entity body. If the content type is not supported, the management node MUST respond with a 415 status code.

If the management node retains knowledge of previously deleted entities and an attempt is made to retrieve or update or delete the entity description, the management node SHOULD respond with a 410 status code. Other requests on unknown addresses MUST be responded with a 404 status code.

## Management Node

The management node returns the discovery document replying to a "GET" request on the management node address.

A successful response MUST carry a 200 status code.

The entity body MUST contain a "discovery-document" instance.

## Entity Type Collection

The entity type collection is available via a GET at the address indicated by the "types" entry in the discovery document. The collection is read-only.

A successful response MUST carry a 200 status code.

The entity body MUST contain a "entity-type-collection" instance.

## Entity Collections

Entity collections are available via an address indicated by one of values of the "collections" map carried the discovery document.

The reserved "entities" collection provides access to all entities and functions as the generic factory.

### Enumerating Entities

The available entities of a collection are enumerated with a GET request on the collection’s address.

A successful response MUST carry a 200 status code if content is returned, or a 204 content code if the collection or collection segment to be returned is empty and nothing is returned.

The entity body MUST contain an "entity-collection" instance.

The number of entity descriptions contained in a single response MAY be limited for paging with the integer-typed $top query parameter that dictates how many descriptions shall at most be returned. The integer-typed $skip query parameter dictates how many descriptions shall be skipped from the start of the collection. For example, $top=10 and $skip=20 return 10 collection elements, from positions 21 through 30.

The order of elements in the collection SHOULD be deterministic when paging is applied, but the order criterion is left to the application. Implementations MAY introduce sort, search, or filtering capabilities.

### Creating Entities

New entities MAY be added to a collection with a POST request to the collection’s address.

The POST request MUST carry an entity description indicating which entity shall be created, including all properties that the client proposes to be set at creation time.

The main collection of the management scope MUST allow creating instances of any type offered in the types collection.

The main collection MAY support creating entities based on archetypes defined in 4.4, in which case the entity description's "type" property is set to the archetype name.

Any collection other than the "entities" collection MAY restrict the kinds of entities that can be created.

The management node MUST return a 400 status code for any unsupported type for the given collection.

When the request is successful, the status code MUST be 201, the **Location** header MUST be set to the created entity’s "self" management address, and the response MUST carry the effective entity description.

The property values in the effective entity description MAY deviate from the proposed property values passed by the client.

## Entities

Each entity has its own "self" management address. From this address, the client can obtain the effective entity description, can propose updates to the description, and can delete the entity.

The “self” management address SHOULD contain the current “etag” value of the entity as a query parameter, such that version conflicts can be detected easily with the update and delete operations.

Some entities MAY have further manageable internal structure. In this case, the entity description contains an "management" address that returns a discovery document scoped to the entity. The "entities" collection of that discovery document is only scoped to the internal structure of the entity.

### Getting the Entity Description

The entity description MUST be obtainable using a GET request to the "self" management address. If an “etag” query parameter is present, that parameter is ignored for the GET request operation. GET always returns the current state of the entity.

A successful response MUST carry a “200” status code.

The entity body MUST contain the entity description.

### Updating the Entity

The entity MAY be modified with a PUT request to the "self" management address.

The PUT request MUST carry a modified instance of the entity description previously obtained from the "self" address or via an entity collection, with all properties that the client proposes to be modified.

The etag property and its value MUST be included and MUST remain unmodified from the previously retrieved entity.

The etagvalue serves to help protect the entity from conflicts resulting from concurrent modifications and MUST change whenever the entity description changes. If the value does not match the etag of the entity description held by the management node, the management node MUST NOT make any modifications and MUST respond with a “409” status code and include the current, effective entity description.

Properties omitted from the entity description MUST not be modified by the management node. Properties that are set to the null value MUST be set to their default value by the management node.

When the request is successful, the status code MUST be 200, and the response MUST carry the effective entity description. The property values in the effective entity description MAY deviate from the proposed property values passed by the client.

### Deleting the Entity

The entity MAY be deleted a DELETE request to the "self" management address. The “self” management address MUST have been obtained from a prior GET request.

If the “self” management address contains a “etag” query parameter, that parameter’s value MAY be used to avoid conflicts. If the value is provided and does not match the etag of the entity description held by the management node, the management node MUST NOT make any modifications and MUST respond with a 409 status code and include the current, effective entity description.

If the “self” management address does not contain a “etag” query parameter, the implementation MAY refuse to delete the entity with a 409 status code.

When the request is successful, the status code MUST be 204.

### Entity Discovery Document

The entity management node whose address is provided via the "management" property in the entity description, returns a discovery document replying to a "GET" request.

A successful response MUST carry a 200 status code.

The entity body MUST contain a "discovery-document" instance.

# Custom Operations

Custom operations are declared in the "operations" section of the discovery document using the "operation" type. Each operation declaration includes an "address" property. Multiple operations SHOULD NOT share the same address.

The "description" attribute is OPTIONAL and contains information about the expected request and response arguments and payloads. If set, the attribute MUST contain the JSON representation of an [OpenAPI 3.0 Operation Object](http://spec.openapis.org/oas/v3.0.2#operation-object) [OpenAPI].

Custom operations MUST be executed as HTTP POST requests on the given operation address.

If no "description" is provided, the operation is executed without parameters or entity body and success or failure of the operation is communicated via the status code.

A complete OpenAPI document can be trivially composed from the operations list by mapping the operation addresses to OpenAPI Path objects, with the "description" becoming the value of the "post" field.

# Examples

Since this specification defines a fairly abstract management model, it will be beneficial to consider a few more concrete yet hypothetical examples to illustrate the design. Three example container topologies will be presented:

1. Example A: A message broker with a concept of exchanges, bindings, and queues as defined by early, initial AMQP drafts.
2. Example B: A JMS-style message broker with standalone queues and topics that have dependent subscriptions.
3. Example C: An event broker with streaming topics and consumer groups.

The documents are presented in JSON notation. This section is non-normative.

## Example A – Exchanges and Queues Broker

A message broker with a concept of exchanges, bindings, and queues as defined by early, initial AMQP drafts.

### Discovery Document

Returned by GET on /$management

{  
 “collections”: {  
 “entities” : {  
 “address” : “entities”,   
 },  
 “exchanges” : {  
 “address” : “exchanges”,  
 “label” : “Exchanges”  
 },  
 “queues” : {  
 “address” : “queues”,  
 “label” : “Queues”  
 }  
 },  
 “types” : {  
 “address” : “types”,  
 “label” : “Supported Types”  
 },  
 “operations” : {  
  
 }   
}

### The main types Collection

Returned by GET on /$management/types

[  
 {  
 “name” : “Queue”,  
 “version” : “1”,  
 “properties” : [  
 {  
 “name” : “source”,  
 “type” : “address”,  
 “label” : “Source address for this queue”,  
 “mandatory” : “true”,  
 “multiple” : “false”,   
 },  
 {  
 “name” : “durable”,  
 “type” : “boolean”,  
 “label” : “Whether the queue is persisted across restarts”,  
 “default” : “true”,  
 “mandatory” : “false”,  
 “multiple” : “false”,   
 },  
 {  
 “name” : “defaulttimetolive”,  
 “type” : “integer”,  
 “label” : “Default message expiration if not explicitly set”,  
 “default” : “120”,  
 “mandatory” : “false”,  
 “multiple” : “false”,   
 },  
 {  
 “name” : “autodelete”,  
 “type” : “boolean”,  
 “label” : “Whether the queue is automatically deleted when unused”,  
 “default” : “false”,  
 “mandatory” : “false”,  
 “multiple” : “false”,   
 },  
 {  
 “name” : “autodeleteexpiration”,  
 “type” : “integer”,  
 “label” : “Timeout after which an unused queue gets deleted”,  
 “default” : “120”,  
 “mandatory” : “false”,  
 “multiple” : “false”,   
 },  
 {  
 “name” : “exclusive”,  
 “type” : “boolean”,  
 “label” : “Whether the queue is exclusively associated with this connection and deleted when the connection is dropped”,  
 “default” : “false”,  
 “mandatory” : “false”,  
 “multiple” : “false”,   
 }  
 ]  
 },  
 {  
 “name” : “Exchange”,  
 “version” : “1”,  
 “properties” : [  
 {  
 “name” : “target”,  
 “type” : “address”,  
 “label” : “Target address for this exchange”,  
 “mandatory” : “true”,  
 “multiple” : “false”,   
 },  
 {  
 “name” : “type”,  
 “type” : “string”,  
 “label” : “Exchange type”,  
 “default” : “multicast”,  
 “mandatory” : “false”,  
 “multiple” : “false”,   
 },  
 {  
 “name” : “durable”,  
 “type” : “boolean”,  
 “label” : “Whether the exchange is persisted across restarts”,  
 “default” : “true”,  
 “mandatory” : “false”,  
 “multiple” : “false”,   
 },,  
 {  
 “name” : “autodelete”,  
 “type” : “boolean”,  
 “label” : “Whether the exchange is automatically deleted when unused”,  
 “default” : “false”,  
 “mandatory” : “false”,  
 “multiple” : “false”,   
 },  
 {  
 “name” : “autodeleteexpiration”,  
 “type” : “integer”,  
 “label” : “Timeout after which an unused exchange gets deleted”,  
 “default” : “120”,  
 “mandatory” : “false”,  
 “multiple” : “false”,   
 },  
 {  
 “name” : “exclusive”,  
 “type” : “boolean”,  
 “label” : “Whether the exchange is exclusively associated with this connection and deleted when the connection is dropped”,  
 “default” : “false”,  
 “mandatory” : “false”,  
 “multiple” : “false”,   
 }  
 ]  
 }   
]

### The entities collection

Returned by a GET on /$management/entities

The collection reflects the existence of two exchanges, E1 and E2 and three queues, Q1, Q2, and Q3. Each of the queues has a local management node where bindings to exchanges are managed.

[  
 {  
 “id” : “Q1”,  
 “etag” : “97261882”,  
 “type” : “Queue“,  
 “self” : “/$management/Queues/Q1?etag=97261882”,  
 “managementnode” : “/$management/Queues/Q1/$management”,  
 “source” : “/Q1”,  
 “durable” : true  
 },  
 {  
 “id” : “Q2”,  
 “etag” : “277272”,  
 “type” : “Queue“,  
 “self” : “/$management/Queues/Q2?etag=277272” ,  
 “managementnode” : “/$management/Queues/Q1/$management”,  
 “source” : “/Q2”,  
 “durable” : true  
 },  
 {  
 “id” : “Q3”,  
 “etag” : “26662121”,  
 “type” : “Queue“,  
 “self” : “/$management/Queues/Q3?etag=26662121” ,  
 “managementnode” : “/$management/Queues/Q1/$management”,  
 “source” : “/Q3”,  
 “durable” : true  
 },  
 {  
 “id” : “E1”,  
 “etag” : “123421”,  
 “type” : “Exchange“,  
 “self” : “/$management/Exchanges/E1?etag=123421”   
 “target” : “/E1”,  
 “durable” : true  
 },  
 {  
 “id” : “E2”,  
 “etag” : “123421”,  
 “type” : “Exchange“,  
 “self” : “/$management/Exchanges/E2?etag=123421”   
 “target” : “/E2”,  
 “durable” : true  
 }  
   
The specialized collections /$management/Queues and /$management/Exchanges return a filtered view on the above, with only Queues in the first and only Exchanges on the latter.

### A queue’s local management node discovery document

Queue Q1 has a local management node. The discovery document returned by a GET on its address /$management/Queues/Q1/$management yields this local discovery document:

{  
 “collections”: {  
 “entities” : {  
 “address” : “entities”,   
 },  
 “bindings” : {  
 “address” : “bindings”,  
 “label” : “Bindings”  
 }  
 },  
 “types” : {  
 “address” : “types”,  
 “label” : “Supported Types”  
 }  
 “operations” : [  
 {  
 “name” : “purge”,  
 “label” : “Purges the queue. All content is removed.”,  
 “address” : “purge”  
 },  
 {  
 “name” : “suspend”,  
 “label” : “Suspends the queue. No messages are sent or received.”,  
 “address” : “suspend”  
 },  
 {  
 “name” : “resume”,  
 “label” : “Resumes the queue. Sends and receives resume after suspension.”,  
 “address” : “resume”  
 }  
 ]  
}

### A queue’s local types collection

The queue (and all queues of the same type) defines a local type, namely that of the binding to an exchange.

Returned by a GET on /$management/Queues/Q1/$management/types

[  
 {  
 “name” : “Binding”,  
 “version” : “1”,  
 “properties” : [  
 {  
 “name” : “exchange”,  
 “type” : “string”,  
 “label” : “Id of exchange to bind to”,  
 “mandatory” : “true”,  
 “multiple” : “false”,   
 },  
 {  
 “name” : “filter”,  
 “type” : “string”,  
 “label” : “Filter expression”,  
 “mandatory” : “false”,  
 “multiple” : “false”,   
 }  
 ]  
 }  
]

### A queue’s entity collection

Queue Q1 is bound to exchange E1 and this is reflected here. Returned on a GET on /$management/Queues/Q1/$management/entities

[  
 {  
 “id” : “Binding1”,  
 “etag” : “97261882”,  
 “type” : “Binding“,  
 “self” : “/$management/Queues/Q1/Bindings/Binding1?etag=97261882”   
 “exchange” : “E1”  
 }  
]

### Updating a queue’s default TTL

To update Q2’s default TTL, we can take the description that was previously returned in the entity collection or we can re-retrieve it from the “self” address to ensure that it is current.

Returned by GET on /$management/Queues/Q2

{  
 “id” : “Q2”,  
 “etag” : “277273”,  
 “type” : “Queue“,  
 “self” : “/$management/Queues/Q2?etag=277273” ,  
 “managementnode” : “/$management/Queues/Q1/$management”,  
 “source” : “/Q2”,  
 “durable” : true  
 }

Set the new value for “defaulttimetolive” by adding the property. Properties that are omitted retain their present values, but “id” and “etag” MUST be included. Then set on the entity “self” address with PUT.

Setting with PUT on /$management/Queues/Q2?etag=277273

{  
 “id” : “Q2”,  
 “etag” : “277273”,   
 “defaulttimetolive” : 120  
 }

The returned result is the new, effective description of the entity with its new etag:

{  
 “id” : “Q2”,  
 “etag” : “278244”,  
 “type” : “Queue“,  
 “self” : “/$management/Queues/Q2?etag=278244” ,  
 “managementnode” : “/$management/Queues/Q1/$management”,   
 “defaulttimetolive” : 120,  
 “source” : “/Q2”,  
 “durable” : true  
 }

### Deleting a queue

Q3 can be deleted with a DELETE on its “self” address, including the current etag as a query parameter as returned in collections. DELETE on /$management/Queues/Q3?etag=26662121 yields a 204 status code if successful.

### Creating a new exchange

Creating a new exchange is performed with a POST of a new description on the main entities collection /$management/entities. The “etag” and “self” address are omitted.

{  
 “id” : “E3”,  
 “type” : “Exchange“,  
 “durable” : true  
 }

If the request succeeds, it returns with a 201 status code and with a Location header set to the “self” address of the newly created entity, here exemplary /$management/Exchanges/E3?etag=7726622 and the response includes the effective description:

{  
 “id” : “E3”,  
 “etag” : “123421”,  
 “type” : “Exchange“,  
 “self” : “/$management/Exchanges/E3?etag=7726622”   
 “target” : “/E3”,  
 “durable” : true  
 }

### Creating a new $queue archetype instance

The $queue archetype differs from the entity types available in this example scenario, because it defines a single entity that has both a message source and a message target, while the scenario types only allow for queues bound to exchanges, meaning the archetype requirements can only be satisfied with a composite of two entities.

To still allow for creation and subsequent deletion of such an archetypical queue, the strategy employed here is to provide a management wrapper around the composite such that the created entities can also be deleted together if desired.

Creating the new archetypical queue is performed with a POST of a new description on the main entities collection /$management/entities.

{  
 “id” : “myqueue”,  
 “type” : “$queue“,  
 “durable” : true  
 }

If the request succeeds, it returns with a 201 status code and with a Location header set to the “self” address of the newly created management wrapper for the created entities, here exemplary /$management/entities/myqueue?etag=99182772 and the response includes the effective description:

{  
 “id” : “myqueue”,  
 “etag” : “99182772”,  
 “type” : “$queue“,  
 “self” : “/$management/entities/myqueue?etag=7726622”   
 “target” : “/exchange\_myqueue”,  
 “source” : “/queue\_myqueue”,  
 “exchange\_self” : “/$management/Exchanges/exchange\_myqueue?etag=9124642”,  
 “queue\_self” : “/$management/Queues/queue\_myqueue?etag=42927622”,   
 }

The description contains the mandated “target” and “source” attributes and otherwise points to the “self” management endpoints of the new created exchange and the created queue. The only function of the “self” address of this returned entity is to allow for deleting both exchange and queue together. The implementation might choose to protect the entities such that deletion of the individual entities is not allowed.

## Example B – JMS-style broker

A message broker with standalone queues and topics with subscriptions, whereby the subscriptions are modeled as sub-entities of the topic for this example as we’ve covered subscriptions in form of bindings in the prior example.

This example omits operations like property updates or entity deletion because they don’t differ from the previous example. Instead, it illustrates a configuration endpoint. The configuration endpoint shows here is using the type “Broker” defined in the types collection. That association is implementation specific.

### Discovery Document

Returned by GET on /$management

{  
 “collections”: {  
 “entities” : {  
 “address” : “entities”,   
 },  
 “exchanges” : {  
 “address” : “topics”,  
 “label” : “Topics”  
 },  
 “queues” : {  
 “address” : “queues”,  
 “label” : “Queues”  
 }  
 },  
 “types” : {  
 “address” : “types”,  
 “label” : “Supported Types”  
 },  
 “configuration” : {  
 “address” : “/$broker”,  
 “label” : “Broker configuration endpoint”  
 },  
 “operations” : [

{  
 “name” : “restart”,  
 “label” : “Restarts the broker. Admin privilege required”,  
 “address” : “/$broker/restart”  
 }  
 ]   
}

### The main types Collection

Returned by GET on /$management/types

[  
 {  
 “name” : “Queue”,  
 “version” : “1”,  
 “properties” : [  
 {  
 “name” : “source”,  
 “type” : “address”,  
 “label” : “Source address for this queue”,  
 “mandatory” : “false”,  
 “multiple” : “false”,   
 },  
 {  
 “name” : “target”,  
 “type” : “address”,  
 “label” : “Target address for this queue”,  
 “mandatory” : “false”,  
 “multiple” : “false”,   
 },  
 {  
 “name” : “durable”,  
 “type” : “boolean”,  
 “label” : “Whether the queue is persisted across restarts”,  
 “default” : “true”,  
 “mandatory” : “false”,  
 “multiple” : “false”,   
 },  
 {  
 “name” : “defaulttimetolive”,  
 “type” : “integer”,  
 “label” : “Default message expiration if not explicitly set”,  
 “default” : “120”,  
 “mandatory” : “false”,  
 “multiple” : “false”,   
 },  
 {  
 “name” : “autodelete”,  
 “type” : “boolean”,  
 “label” : “Whether the queue is automatically deleted when unused”,  
 “default” : “false”,  
 “mandatory” : “false”,  
 “multiple” : “false”,   
 },  
 {  
 “name” : “autodeleteexpiration”,  
 “type” : “integer”,  
 “label” : “Timeout after which an unused queue gets deleted”,  
 “default” : “120”,  
 “mandatory” : “false”,  
 “multiple” : “false”,   
 },  
 {  
 “name” : “exclusive”,  
 “type” : “boolean”,  
 “label” : “Whether the queue is exclusively associated with this connection and deleted when the connection is dropped”,  
 “default” : “false”,  
 “mandatory” : “false”,  
 “multiple” : “false”,   
 }  
 ]  
 },  
 {  
 “name” : “Topic”,  
 “version” : “1”,  
 “properties” : [  
 {  
 “name” : “target”,  
 “type” : “address”,  
 “label” : “Target address for this topic”,  
 “mandatory” : “true”,  
 “multiple” : “false”,   
 },  
 {  
 “name” : “durable”,  
 “type” : “boolean”,  
 “label” : “Whether the topic is persisted across restarts”,  
 “default” : “true”,  
 “mandatory” : “false”,  
 “multiple” : “false”,   
 },  
 {  
 “name” : “autodelete”,  
 “type” : “boolean”,  
 “label” : “Whether the topic is automatically deleted when unused”,  
 “default” : “false”,  
 “mandatory” : “false”,  
 “multiple” : “false”,   
 },  
 {  
 “name” : “autodeleteexpiration”,  
 “type” : “integer”,  
 “label” : “Timeout after which an unused topic gets deleted”,  
 “default” : “120”,  
 “mandatory” : “false”,  
 “multiple” : “false”,   
 },  
 {  
 “name” : “exclusive”,  
 “type” : “boolean”,  
 “label” : “Whether the topic is exclusively associated with this connection and deleted when the connection is dropped”,  
 “default” : “false”,  
 “mandatory” : “false”,  
 “multiple” : “false”,   
 }  
 ]  
 },  
 {  
 “name” : “Broker”,  
 “version” : “1”,  
 “properties” : [  
 {  
 “name” : “host”,  
 “type” : “string”,  
 “label” : “Hostname or IPv4 or IPv6 address to listen on”,  
 “mandatory” : “true”,  
 “multiple” : “false”,   
 },  
 {  
 “name” : “port”,  
 “type” : “string”,  
 “label” : “Port number to listen on”,  
 “default” : “”,  
 “mandatory” : “true”,  
 “multiple” : “false”,   
 }  
 ]  
 }   
]

### The entities collection

Returned by a GET on /$management/entities

The collection reflects the existence of two topics, T1 and T2 and three queues, Q1, Q2, and Q3. Each of the topics has a local management node where subscriptions are managed.

[  
 {  
 “id” : “Q1”,  
 “etag” : “97261882”,  
 “type” : “Queue“,  
 “self” : “/$management/Queues/Q1?etag=97261882”   
 “source” : “/Q1”,  
 “target” : “/Q1”,  
 “durable” : true  
 },  
 {  
 “id” : “Q2”,  
 “etag” : “277272”,  
 “type” : “Queue“,  
 “self” : “/$management/Queues/Q2?etag=277272”,   
 “source” : “/Q2”,  
 “target” : “/Q2”,  
 “durable” : true  
 },  
 {  
 “id” : “Q3”,  
 “etag” : “26662121”,  
 “type” : “Queue“,  
 “self” : “/$management/Queues/Q3?etag=26662121”,  
 “source” : “/Q3”,  
 “target” : “/Q3”,  
 “durable” : true  
 },  
 {  
 “id” : “T1”,  
 “etag” : “123421”,  
 “type” : “Exchange“,  
 “self” : “/$management/Topics/T1?etag=123421”,  
 “managementnode” : “/$management/Topics/T1/$management”,  
 “target” : “/T1”,  
 “durable” : true  
 },  
 {  
 “id” : “T2”,  
 “etag” : “123421”,  
 “type” : “Exchange“,  
 “self” : “/$management/Topics/T2?etag=123421” ,  
 “managementnode” : “/$management/Topics/T2/$management”,   
 “target” : “/T2”,  
 “durable” : true  
 }  
   
The specialized collections /$management/Queues and /$management/Topics return a filtered view on the above, with only Queues in the first and only Topics in the latter.

### A topic’s local management node discovery document

Topic T1 has a local management node. The discovery document returned by a GET on its address /$management/Topics/T1/$management yields this local discovery document:

{  
 “collections”: {  
 “entities” : {  
 “address” : “entities”,   
 },  
 “subscriptions” : {  
 “address” : “subscriptions”,  
 “label” : “Subscriptions”  
 }  
 },  
 “types” : {  
 “address” : “types”,  
 “label” : “Supported Types”  
 }  
}

### A topic’s local types collection

The topic (and all topics of the same type) defines a local type, namely that of the subscription, which is a queue-like structure attached to the topic.

Returned by a GET on /$management/Topics/T1/$management/types

[  
 {  
 “name” : “Subscription”,  
 “version” : “1”,  
 “properties” : [  
   
 {  
 “name” : “filter”,  
 “type” : “string”,  
 “label” : “Filter expression”,  
 “mandatory” : “false”,  
 “multiple” : “false”,   
 },  
 {  
 “name” : “source”,  
 “type” : “address”,  
 “label” : “Source address for this queue”,  
 “mandatory” : “false”,  
 “multiple” : “false”,   
 },  
 {  
 “name” : “durable”,  
 “type” : “boolean”,  
 “label” : “Whether the subscription is persisted across restarts”,  
 “default” : “true”,  
 “mandatory” : “false”,  
 “multiple” : “false”,   
 },  
 {  
 “name” : “defaulttimetolive”,  
 “type” : “integer”,  
 “label” : “Default message expiration if not explicitly set”,  
 “default” : “120”,  
 “mandatory” : “false”,  
 “multiple” : “false”,   
 },  
 {  
 “name” : “autodelete”,  
 “type” : “boolean”,  
 “label” : “Whether the subscription is automatically deleted when unused”,  
 “default” : “false”,  
 “mandatory” : “false”,  
 “multiple” : “false”,   
 },  
 {  
 “name” : “autodeleteexpiration”,  
 “type” : “integer”,  
 “label” : “Timeout after which an unused subscription gets deleted”,  
 “default” : “120”,  
 “mandatory” : “false”,  
 “multiple” : “false”,   
 },  
 {  
 “name” : “exclusive”,  
 “type” : “boolean”,  
 “label” : “Whether the subscription is exclusively associated with this connection and deleted when the connection is dropped”,  
 “default” : “false”,  
 “mandatory” : “false”,  
 “multiple” : “false”,   
 }  
 ]  
 }  
]

### A topic’s entity collection

Subscription S1 is attached to Topic T1 and this is reflected here. Returned on a GET on /$management/Topic/T1/$management/entities

[  
 {  
 “id” : “S1”,  
 “etag” : “97261882”,  
 “type” : “Subscription“,  
 “self” : “/$management/Topics/T1/Subscriptions/S1?etag=97261882”   
 “source” : “/T1/subscriptions/S1”,  
 “durable” : true   
 }  
]

### Updating a queue’s default TTL

Identical to 7.1.7

### Deleting a queue

Identical to 7.1.8

### Creating a new exchange

Creating a new topic is performed with a POST of a new description on the main entities collection /$management/entities. The “etag” and “self” address are omitted.

{  
 “id” : “T3”,  
 “type” : “Topic“,  
 “durable” : true  
 }

If the request succeeds, it returns with a 201 status code and with a Location header set to the “self” address of the newly created entity, here exemplary /$management/Topics/T3?etag=7726622 and the response includes the effective description:

{  
 “id” : “T3”,  
 “etag” : “123421”,  
 “type” : “Topic“,  
 “self” : “/$management/Topics/T3?etag=7726622”   
 “target” : “/T3”,  
 “durable” : true  
 }

### Creating a new $queue archetype instance

In contrast to example A, the $queue archetype shape matches the queue entity type available in this example scenario, because it defines a single entity that has both a message source and a message target.

Creating the new archetypical queue is performed with a POST of a new description on the main entities collection /$management/entities.

{  
 “id” : “myqueue”,  
 “type” : “$queue“,  
 “durable” : true  
 }

If the request succeeds, it returns with a 201 status code and with a Location header set to the “self” address of the newly created queue:

{  
 “id” : “myqueue”,  
 “etag” : “99182772”,  
 “type” : “Queue“,  
 “self” : “/$management/Queues/myqueue?etag=7726622”   
 “target” : “/myqueue”,  
 “source” : “/myqueue”,  
 “durable” : true   
 }

## Example C – Event stream broker

This example illustrates an event stream broker with streams and consumer groups.

### Discovery Document

Returned by GET on /$management

{  
 “collections”: {  
 “entities” : {  
 “address” : “entities”,   
 },  
 “streams” : {  
 “address” : “streams”,  
 “label” : “Streams”  
 }  
 },  
 “types” : {  
 “address” : “types”,  
 “label” : “Supported Types”  
 },  
 “configuration” : {  
 “address” : “/$broker”,  
 “label” : “Broker configuration endpoint”  
 },  
 “operations” : [

{  
 “name” : “restart”,  
 “label” : “Restarts the broker. Admin privilege required”,  
 “address” : “/$broker/restart”  
 }  
 ]   
}

### The main types Collection

Returned by GET on /$management/types

[  
 {  
 “name” : “Stream”,  
 “version” : “1”,  
 “properties” : [  
 {  
 “name” : “target”,  
 “type” : “address”,  
 “label” : “Target address for this stream”,  
 “mandatory” : “false”,  
 “multiple” : “false”,   
 },  
 {  
 “name” : “partitions”,  
 “type” : “integer”,  
 “label” : “Number of partitions”,  
 “default” : “4”,  
 “mandatory” : “false”,  
 “multiple” : “false”,   
 }  
 ]  
 },  
 {  
 “name” : “Broker”,  
 “version” : “1”,  
 “properties” : [  
 {  
 “name” : “host”,  
 “type” : “string”,  
 “label” : “Hostname or IPv4 or IPv6 address to listen on”,  
 “mandatory” : “true”,  
 “multiple” : “false”,   
 },  
 {  
 “name” : “port”,  
 “type” : “string”,  
 “label” : “Port number to listen on”,  
 “default” : “”,  
 “mandatory” : “true”,  
 “multiple” : “false”,   
 }  
 ]  
 }   
]

### The entities collection

Returned by a GET on /$management/entities

The collection reflects the existence of two topics, T1 and T2 and three queues, Q1, Q2, and Q3. Each of the topics has a local management node where subscriptions are managed.

[  
 {  
 “id” : “S1”,  
 “etag” : “27261882”,  
 “type” : “Stream“,  
 “self” : “/$management/Streams/S1?etag=27261882”,  
 “managementnode” : “/$management/Streams/S1/$management”,   
 “target” : “/S1”,  
 },  
 {  
 “id” : “S2”,  
 “etag” : “37261882”,  
 “type” : “Stream“,  
 “self” : “/$management/Streams/S2?etag=37261882”,  
 “managementnode” : “/$management/Streams/S3/$management”,   
 “target” : “/S3”,  
 }

### A stream’s local management node discovery document

Stream S1 has a local management node. The discovery document returned by a GET on its address /$management/Streams/S1/$management yields this local discovery document:

{  
 “collections”: {  
 “entities” : {  
 “address” : “entities”,   
 },  
 “consumergroups” : {  
 “address” : “consumergroups”,  
 “label” : “Consumer Groups”  
 }  
 },  
 “types” : {  
 “address” : “types”,  
 “label” : “Supported Types”  
 }  
}

### A stream’s local types collection

The stream (and all streams of the same type) defines a local type, namely that of the Consumergroup, which is an event source with distinct concurrency management attached to the stream entity.

Returned by a GET on /$management/Streams/S1/$management/types

[  
 {  
 “name” : “ConsumerGroup”,  
 “version” : “1”,  
 “properties” : [  
   
 {  
 “name” : “source”,  
 “type” : “address”,  
 “label” : “Source address for this queue”,  
 “mandatory” : “false”,  
 “multiple” : “false”,   
 }  
 ]  
 }  
]

### A stream’s entity collection

Consumer Group CG1 is attached to Stream S1 and this is reflected here. Returned on a GET on /$management/Streams/S1/$management/entities

[  
 {  
 “id” : “CG1”,  
 “etag” : “37261882”,  
 “type” : “ConsumerGroup“,  
 “self” : “/$management/streams/S1/consumergroups/CG1?etag=37261882”   
 “source” : “/S1/consumergroups/CG1”  
 }  
]

# Conformance

When considering this specification, we can consider two distinct roles an AMQP container may play:

Firstly, that of a Requesting Container– a container which wants to initiate management operations; secondly an Responding Container – a container which responds to such requests.

A Requesting Container is conformant with this specification if:

* management operations are only issued over AMQP if the AMQP management capability has been negotiated.
* the default “$management” is only used if no other endpoint has been offered via the “managementnode” connection property (see 3.1) or via the HTTP discovery mechanism (see 3.2).
* all request addresses are dynamically discovered starting from the discovery document. A requesting container MUST NOT use fixed assumptions about the addressing structure of the management API.

A Responding Container is conformant with this specification if:

* it offers the AMQP connection capability (see 3.1) if AMQP Management is supported over AMQP
* it offers management endpoint discovery over HTTP (see 3.2) if AMQP Management is offered over HTTP.
* it returns a discovery document (see 2.1) on the management endpoint.
* it offers an endpoint for the mandatory “entities” collection (see 4.1) and the endpoint returns a list of entities (including one that may be empty) conformant with the type system described herein.
* it implements an endpoint for the mandatory “types” collection (see 4.2) and the endpoint returns a list of entity types (including one that may be empty) conformant with the type system described herein.

The ability to create, modify, or delete entities is not required for conformance.

# OpenAPI Definition

This definition uses the OpenAPI 3.0.2 specification and is not normative. Mind that OpenAPI is not capable of representing dynamic, multi-segment paths in its “Paths” object and therefore this schema should only be considered a template for an actual implementation.

{

"openapi": "3.0.2",

"info": {

"title": "AMQP Management",

"version": "1.0"

},

"components": {

"schemas": {

"discovery-document": {

"type": "object",

"required": [

"collections"

],

"properties": {

"collections": {

"$ref": "#/components/schemas/collections"

},

"types": {

"$ref": "#/components/schemas/entity-type-collection-ref"

},

"configuration": {

"$ref": "#/components/schemas/configuration-ref"

},

"operations": {

"$ref": "#/components/schemas/operations"

}

}

},

"collections": {

"type": "object",

"required": [

"entities"

],

"properties": {

"entities": {

"$ref": "#/components/schemas/entity-collection-ref"

}

},

"additionalProperties": {

"$ref": "#/components/schemas/entity-collection-ref"

}

},

"operations": {

"type": "array",

"items": {

"$ref": "#/components/schemas/operation"

}

},

"configuration-ref": {

"type": "object",

"properties": {

"label": {

"type": "string"

},

"address": {

"type": "string",

"format": "uri-reference"

}

}

},

"entity-collection-ref": {

"type": "object",

"properties": {

"label": {

"type": "string"

},

"address": {

"type": "string",

"format": "uri-reference"

}

}

},

"entity-type-collection-ref": {

"type": "object",

"properties": {

"label": {

"type": "string"

},

"address": {

"type": "string",

"format": "uri-reference"

}

}

},

"operation": {

"type": "object",

"properties": {

"name": {

"type": "string"

},

"label": {

"type": "string"

},

"address": {

"type": "string",

"format": "uri-reference"

},

"description": {

"$ref": "https://raw.githubusercontent.com/OAI/OpenAPI-Specification/master/schemas/v3.0/schema.json#/definitions/Operation"

}

}

},

"entity-collection": {

"type": "array",

"items": {

"$ref": "#/components/schemas/entity"

}

},

"entity": {

"type": "object",

"required" : ["id", "type", "typeversion", "etag", "self"],

"properties": {

"id": {

"type": "string",

"description": "Identifier. Must be set to null during creation, required otherwise."

},

"type": {

"type": "string",

"description": "Entity type. May be an archetype reference during creation."

},

"typeversion": {

"type": "string",

"description": "Referenced entity type version."

},

"label": {

"type": "string",

"description": "Descriptive label"

},

"etag": {

"type": "string",

"description": "Version etag for concurrency control. Must be set to null during creation, required otherwise."

},

"self": {

"type": "string",

"format": "uri-reference",

"description": "Self management address."

},

"managementnode": {

"type": "string",

"format": "uri-reference",

"description": "Management node for internal structure of this entity"

}

}

},

"entity-type-collection": {

"type": "array",

"items": {

"$ref": "#/components/schemas/entity-type"

}

},

"entity-type": {

"type": "object",

"properties": {

"name": {

"type": "string",

"description": "Type name"

},

"version": {

"type": "string",

"description": "Version of this type"

},

"properties": {

"type": "array",

"items": {

"$ref": "#/components/schemas/entity-property"

}

}

}

},

"entity-property": {

"type": "object",

"properties": {

"name": {

"type": "string",

"description": "Property name"

},

"amqptype": {

"type": "string",

"description": "AMQP data type"

},

"label": {

"type": "string",

"description": "Descriptive label"

}

}

}

}

},

"paths": {

"/$management": {

"description": "Management discovery endpoint. Might redirect to managemt node endpoint or return the discovery document.",

"get": {

"responses": {

"302": {

"description" : "Redirect to management node",

"headers": {

"Location": {

"description": "Management node address",

"schema": {

"type": "string"

}

}

}

},

"200": {

"description": "",

"content": {

"application/amqp-management+json;type=discovery-document": {

"schema": {

"$ref": "#/components/schemas/discovery-document"

}

},

"application/amqp-management+amqp;type=discovery-document": {

"schema": {

"$ref": "#/components/schemas/discovery-document"

}

}

}

}

}

}

},

"/{management-node}": {

"description": "Management node.",

"get": {

"responses": {

"200": {

"description": "",

"content": {

"application/amqp-management+json;type=discovery-document": {

"schema": {

"$ref": "#/components/schemas/discovery-document"

}

},

"application/amqp-management+amqp;type=discovery-document": {

"schema": {

"$ref": "#/components/schemas/discovery-document"

}

}

}

}

}

}

},

"/{entity-collection}": {

"description": "Entity collection. Referred to by the discovery document.",

"get": {

"description": "Returns a list of entities",

"parameters": [

{

"name": "$top",

"in": "query",

"required": false,

"description": "limits the enumeration to the given number of entities",

"allowEmptyValue": false,

"schema": {

"type": "integer"

}

},

{

"name": "$skip",

"in": "query",

"required": false,

"description": "skips the given numnber of entities",

"allowEmptyValue": false,

"schema": {

"type": "integer"

}

}

],

"responses": {

"200": {

"description": "",

"content": {

"application/amqp-management+json;type=entity-collection": {

"schema": {

"$ref": "#/components/schemas/entity-collection"

}

},

"application/amqp-management+amqp;type=entity-collection": {

"schema": {

"$ref": "#/components/schemas/entity-collection"

}

}

}

}

}

},

"post": {

"description": "Entity factory. Creates new entities given an entity description.",

"requestBody": {

"content": {

"application/amqp-management+json;type=entity": {

"schema": {

"$ref": "#/components/schemas/entity"

}

},

"application/amqp-management+amqp;type=entity": {

"schema": {

"$ref": "#/components/schemas/entity"

}

}

}

},

"responses": {

"201": {

"description": "Successful creation of a new entity",

"headers": {

"Location": {

"description": "Self management address of the created entity",

"schema": {

"type": "string"

}

}

},

"content": {

"application/amqp-management+json;type=entity-collection": {

"schema": {

"$ref": "#/components/schemas/entity-collection"

}

},

"application/amqp-management+amqp;type=entity-collection": {

"schema": {

"$ref": "#/components/schemas/entity-collection"

}

}

}

},

"400": {

"description": "Error creating entity"

},

"406": {

"description": "Content not acceptable"

},

"419": {

"description": "Conflict with existing entity"

}

}

}

},

"/{entity}": {

"get": {

"responses": {

"200": {

"description": "",

"content": {

"application/amqp-management+json;type=entity": {

"schema": {

"$ref": "#/components/schemas/entity"

}

},

"application/amqp-management+amqp;type=entity": {

"schema": {

"$ref": "#/components/schemas/entity"

}

}

}

}

}

},

"put": {

"requestBody": {

"content": {

"application/amqp-management+json;type=entity": {

"schema": {

"$ref": "#/components/schemas/entity"

}

},

"application/amqp-management+amqp;type=entity": {

"schema": {

"$ref": "#/components/schemas/entity"

}

}

}

},

"responses": {

"200": {

"description": "Successful update of entity",

"content": {

"application/amqp-management+json;type=entity": {

"schema": {

"$ref": "#/components/schemas/entity"

}

},

"application/amqp-management+amqp;type=entity": {

"schema": {

"$ref": "#/components/schemas/entity"

}

}

}

},

"400": {

"description": "Error creating entity"

},

"406": {

"description": "Content not acceptable"

},

"419": {

"description": "Conflict with existing entity"

}

}

},

"delete" : {

"parameters": [

{

"name": "etag",

"required": false,

"in" : "query",

"description": "Concurrency etag",

"allowEmptyValue": false,

"schema": {

"type": "string"

}

}

],

"requestBody": {

"content": {

"application/amqp-management+json;type=entity": {

"schema": {

"$ref": "#/components/schemas/entity"

}

},

"application/amqp-management+amqp;type=entity": {

"schema": {

"$ref": "#/components/schemas/entity"

}

}

}

},

"responses": {

"204": {

"description": "Successful delete of entity"

},

"400": {

"description": "Error creating entity"

},

"406": {

"description": "Content not acceptable"

},

"419": {

"description": "Conflict with existing entity"

}

}

}

},

"/{type-collection}": {

"get": {

"responses": {

"default": {

"description": "",

"content": {

"application/amqp-management+json;type=entity-type-collection": {

"schema": {

"$ref": "#/components/schemas/entity-type-collection"

}

},

"application/amqp-management+amqp;type=entity-type-collection": {

"schema": {

"$ref": "#/components/schemas/entity-type-collection"

}

}

}

}

}

}

}

}

}

# Acknowledgments

The following individuals have participated in the creation of this specification and are gratefully acknowledged:

Participants:

Alan Conway, Red Hat

Rob Godfrey, Red Hat

Keith Wall, Red Hat

Robbie Gemmell, Red Hat

Justin Ross, Red Hat

Ted Ross, Red Hat

Oleksandr Rudyy, JP Morgan  
Xin Chen, Microsoft  
Clemens Vasters, Microsoft

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Revision** | **Date** | **Editor** | **Changes Made** |
| [Rev number] | [Rev Date] | [Modified By] | [Summary of Changes] |