Advanced Message Queuing Protocol (AMQP) Management Version 1.0

Working Draft 09

2 November 2014

Technical Committee:

OASIS Advanced Message Queuing Protocol (AMQP) TC

Chairs:

Ram Jeyaraman (Ram.Jeyaraman@microsoft.com), Microsoft
Robert Godfrey (robert.godfrey@jpmorgan.com), JPMorgan Chase & Co.

Editors:

Robert Godfrey (robert.godfrey@jpmorgan.com), JPMorgan Chase & Co.
David Ingham (dingham@redhat.com), Red Hat
Rob Dolin (RobDolin@microsoft.com), Microsoft
Ted Ross (tross@redhat.com), Red Hat

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:

AMQP Management is layered on top of the AMQP 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 (WD) has been produced by one or more TC Members; it has not yet been voted on by the TC or approved as a Committee Draft (Committee Specification Draft or a Committee Note Draft). The OASIS document Approval Process 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.)

Table of Contents

1 Introduction

[All text is normative unless otherwise labeled]

TODO: Write introduction

1.1 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].**

1.2 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

[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.

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., "Hypertext Transfer Protocol -- HTTP/1.1", RFC2616, June 1999. http://www.w3.org/Protocols/rfc2616/rfc2616.html.

[RFC7159] Crockford, D., “The application/json Media Type for JavaScript Object Notation (JSON)”, RFC7159, July 2006. http://tools.ietf.org/html/rfc7159.

1.3 Non-Normative References

[SLPv2] Guttman, E., Perkins, C., Veizades, J., Day, M., “Service Location Protocol, Version 2”, RFC 2608, June 1999. http://www.ietf.org/rfc/rfc2608.txt.

[WS-Man] Cohen, Josh and Lamers, Larry, “Web Services for Management (WS-Management) Specification”, DSP0226, May 2010. http://www.dmtf.org/sites/default/files/standards/documents/DSP0226_1.1.pdf.

NOTE: The proper format for citation of technical work produced by an OASIS TC (whether Standards Track or Non-Standards Track) is:

[Citation Label]

Work Product title (italicized). Approval date (DD Month YYYY). OASIS Stage Identifier and Revision Number (*e.g.*, OASIS Committee Specification Draft 01). Principal URI (version-specific URI, *e.g.*, with filename component: `somespec-v1.0-csd01.html`).

For example:

[OpenDoc-1.2] *Open Document Format for Office Applications (OpenDocument) Version 1.2*. 19 January 2011. OASIS Committee Specification Draft 07. http://docs.oasis-open.org/office/v1.2/csd07/OpenDocument-v1.2-csd07.html.

[CAP-1.2] *Common Alerting Protocol Version 1.2*. 01 July 2010. OASIS Standard. http://docs.oasis-open.org/emergency/cap/v1.2/CAP-v1.2-os.html.

2 Concepts

TODO: Intro para required here.

2.1 Summary

TODO: introduce concept of base and concrete Manageable Entity Types – is this still meaningful?

Concept	Description
Manageable Entity Type	A class of entities that can be managed, e.g., "queue".
Type Annotations	An Type Annotation associates a type with externally defined semantics (and possibly a set of attributes and operations related to those semantics).
Manageable Entity	An object on which management operations can be performed, e.g., a queue "q1".
Management Operation	An action that can be performed on a Manageable Entity, e.g., "DELETE". Management Operations have a unique case-sensitive name.
Management Node	A Node which executes Management Operations.
Management Address	An Address of a Management Node. TODO: Add a reference to the AMQP Global Addressing spec when available.

2.2 Manageable Entity Type

A Manageable Entity Type defines a class of entities that can be managed using the protocol defined in this specification. All entities of the same type MUST support the operations, and have the attributes, defined by that Manageable Entity Type.

Each Manageable Entity is an instance of a specific Manageable Entity Type, e.g., “com.example.broker.priorityqueue.” However, this Manageable Entity Type MAY extend another Manageable Entity Type, e.g., an entity with Manageable Entity Type "com.example.broker.priorityqueue" could extend the Manageable Entity Type: "com.example.broker.queue" which itself could extend "org.amqp.queue".

Manageable Entity Types are named using a case-sensitive `string`. Manageable Entity Type names that are not of the form of a reverse domain name prefix and names prefixed with "org.amqp." are reserved. Implementers MAY define their own Manageable Entity Types which MUST be named using a reverse domain name (e.g., "com.example.broker.priorityqueue") for a domain name owned by the implementer.

2.3 Type Annotations

A Type Annotations associates a Manageable Entity Type with an externally defined set of semantics (along with any attributes and operations defined in that Type Annotation). A Manageable Entity Type may implement multiple Type Annotations. A Manageable Entity Type that implements a Type Annotation incorporates all the attributes and operations of the Type Annotation into its own definition.

Two Manageable Entity Types that are not otherwise related MAY implement the same Type Annotation. For example, the Manageable Entity Types “com.example.broker.priorityqueue” and “com.example.useraccount” might both implement the Type Annotation “com.example.stoppable”.

Type Annotations are named using a case-sensitive `string`. Type Annotation names that are not of the form of a reverse domain name prefix and names prefixed with "org.amqp." are reserved. Implementers MAY define their own Type Annotations that MUST be named using a reverse domain name (e.g., "com.example.stoppable") for a domain name owned by the implementer. To avoid confusion implementers SHOULD NOT use the same name for both a Type Annotation and a Manageable Entity Type.

2.4 Management Nodes

A Management Node acts as a service that processes Management Operations. Management Operations are transferred to, and responses are received from, a Management Node using the request/response pattern.

Each AMQP container MUST provide a Management Node with an address $management. A container MAY provide other Management Nodes with arbitrary addresses.

2.5 Manageable Entities

A Manageable Entity MAY be an addressable Node (e.g., a queue), or may be a type of entity that is not addressable (e.g., a user). The operations permitted on a Manageable Entity will depend on its type. This specification does not define the collection of supported Manageable Entity Types. This specification does define a set of standard Management Operations that MAY be augmented with additional type specific operations.

Every Management Node MUST contain a Manageable Entity named “self” and of type “org.amqp.management”. This entity represents the Management Node itself.

2.5.1 Attributes

Manageable entities MUST have the following common attributes:

Attribute	Value Type	Description
name	`string`	A case-sensitive `string` identifying the entity. It MUST be unique within the Management Node through which it is accessed. It MAY change during its lifetime. When a new Manageable Entity is created the name MUST be supplied via the CREATE operation.
identity	`string`	An immutable, case-sensitive `string` identifying the entity. When a new Manageable Entity is created the identity is not supplied by via the CREATE operation, rather the identity is generated by the system under management. The identity MUST be unique within the Management Node through which it is accessed (and within a given implementation may be guaranteed to be unique within a larger domain).
type	`string`	A case-sensitive `string` identifying the Manageable Entity Type for the entity.

2.6 Case sensitivity

TODO – add note on case sensitivity

3 Operations

All manageable entities SHOULD support standard manageable entity operations such as CREATE, READ, UPDATE, and DELETE.

Implementers of manageable entities MAY define their own operations but SHOULD use these standard operations (e. g.: “CREATE”) rather than defining their own entity-specific operations for similar tasks (ex: “CREATE-NEW-TOPIC”.)

TODO: need to talk about the distinction between operations on manageable entities vs. operations on management nodes.

3.1 Request Messages

Request messages have the following `application-properties`:

Key	Value Type	Mandatory?	Description
operation	`string`	Yes	The management operation to be performed. This is case-sensitive.
type	`string`	Yes	The Manageable Entity Type of the Manageable Entity to be managed. This is case-sensitive.
locales	`string`	No	A listing of locales that the sending peer permits for incoming informational text in response messages. The value MUST be of the form (presented in the augmented BNF defined in section 2 of [RFC2616]): `#Language-Tag` where Language-Tag is defined in [BCP47]. That is a sequence of language tags separated by one or more commas and OPTIONAL linear white space, such as “de-CH, it-CH, fr-CH” This locales MUST be ordered in decreasing level of preference. The receiving partner will choose the first (most preferred) incoming locale from those which it supports. If none of the requested locales are supported, "en-US" MUST be chosen. Note that "en-US" need not be supplied in the locales as it is always the fallback. The `string` is not case-sensitive.

Other application-properties MAY provide additional context. If an application-property is not recognized then it MUST be ignored.

3.2 Response Messages

The correlation-id of the response message MUST be the correlation-id from the request message (if present), else the message-id from the request message.

Response messages have the following `application-properties`:

Key	Value Type	Mandatory?	Description
statusCode	`integer`	Yes	HTTP response code [RFC2616]

statusDescription	`string`	No	Description of the status.

The type and contents of the body are operation-specific.

3.2.1 Successful Operations

Successful operations MUST result in a statusCode in the 2xx range as defined in Section 10.2 of **[RFC2616]**. Further details including the form of the body are provided in the definition of each operation.

3.2.2 Unsuccessful Operations

Unsuccessful operations MUST NOT result in a statusCode in the 2xx range as defined in Section 10.2 of **[RFC2616]**. The following error status code SHOULD be used for the following common failure scenarios:

statusCode	Label	Meaning
501	Not Implemented	The operation is not supported.
404	Not Found	The Manageable Entity on which to perform the operation could not be found

Further details of operation-specific codes are provided in the definition of each operation.

The statusDescription of a response to an unsuccessful operation SHOULD provide further information on the nature of the failure.

The form of the body of a response to an unsuccessful operation is unspecified and MAY be implementation-dependent. Clients SHOULD ignore the body of response message if the statusCode is not in the 2xx range.

3.3 Standard Manageable Entity Operations

3.3.1 CREATE

Create a new Manageable Entity.

3.3.1.1 Request

Additional application-properties

Key	Value Type	Mandatory?	Description
name	`string`	Yes	The name of the Manageable Entity to be managed. This is case-sensitive.

Body

The body MUST consist of an `amqp-value` section containing a `map`. The `map` consists of key-value pairs where the key represents the name of an attribute of the entity and the value represents the initial value it SHOULD take.

The absence of an attribute name implies that the entity should take its default value, if defined.

If the map contains a key-value pair where the value is null then the created entity should have no value for that attribute, overriding any default.

Where the attribute value provided is of type `string`, but the expected AMQP type of the attribute value is not `string`, conversion into the correct type MUST be performed according to the following rules:

- A `string` that consists solely of characters from the ASCII character-set, will be converted into a `symbol` if so required.

- A `string` that can be parsed as a number according to **[RFC7159]** will be converted to a `ubyte`, `ushort`, `uint`, `ulong`, `byte`, `short`, `int`, or `long` if so required and the number lies within the domain of the given AMQP type and represents an integral number
- A `string` which can be parsed as a number according to **[RFC7159]** will be converted to an float, double, decimal32, decimal64 or decimal128 if so required and the number lies within the domain of the given AMQP type.
- A `string` which can be parsed as true or false according to **[RFC7159]** will be converted to a `boolean` value if so required.
- A `string` which can be parsed as an array according to **[RFC7159]** will be converted into a `list` (with the values type-converted into elements as necessary according to the same rules) if so required.
- A `string` which can be parsed as an object according to **[RFC7159]** will be converted into a `map` (with the values type-converted into `map` values as necessary according to the same rules) if so required.

3.3.1.2 Response

If the request was successful then the statusCode MUST be 201 (Created) and the body of the message MUST consist of an `amqp-value` section that contains a `map` containing the actual attributes of the entity created. These MAY differ from those requested in two ways:

- Default values may be returned for values not specified
- Specific/concrete values may be returned for generic/base values specified
- The value associated with an attribute may have been converted into the correct amqp type (e.g. the `string` "2" into the `integer` value 2)

A map containing attributes that are not applicable for the entity being created, or invalid values for a given attribute, MUST result in a failure response with a statusCode of 400 (Bad Request).

3.3.2 READ

Retrieve the attributes of a Manageable Entity.

3.3.2.1 Request

Additional application-properties

Key	Value Type	Mandatory?	Description
name	`string`	Exactly one of name or identity MUST be provided.	The name of the Manageable Entity to be managed. This is case-sensitive.
identity	`string`		The identity of the Manageable Entity to be managed. This is case-sensitive.

Body

No information is carried in the message body therefore any message body is valid and MUST be ignored.

3.3.2.2 Response

If the request was successful, then the statusCode MUST contain 200 (OK) and the body of the message MUST consist of an `amqp-value` section containing a `map` containing the attributes of the Manageable Entity. Note that in certain situations the map might not contain the full set of attributes due to security considerations.

3.3.3 UPDATE

Update a Manageable Entity.

3.3.3.1 Request

Additional application-properties

Key	Value Type	Mandatory?	Description
name	`string`	Exactly one of name or identity MUST be provided.	The name of the Manageable Entity to be managed. This is case-sensitive.
identity	`string`		The identity of the Manageable Entity to be managed. This is case-sensitive.

Body

The body MUST consist of an `amqp-value` section containing a `map`. The map consists of key-value pairs where the key represents the name of an attribute of the entity and the value represents the initial value it SHOULD take. The absence of an attribute name implies that the entity should retain its existing value.

If the map contains a key-value pair where the value is null then the updated entity should have no value for that attribute, removing any previous value.

In the case where the supplied map contains multiple attributes, then the update MUST be treated as a single, atomic operation so if any of the changes cannot be applied, none of the attributes in the map should be updated and this MUST result in a failure response.

Where the type of the attribute value provided is not as required, type conversion as per the rules in 3.3.1.1 MUST be provided.

3.3.3.2 Response

If the request was successful then the statusCode MUST contain 200 (OK) and the body of the message MUST consists of an `amqp-value` section containing a `map` of the actual attributes of the entity updated. These MAY differ from those requested.

A map containing attributes that are not applicable for the entity being created, or an invalid value for a given attribute (excepting type conversion as above), MUST result in a failure response with a statusCode of 400 (Bad Request).

3.3.4 DELETE

Delete a Manageable Entity.

3.3.4.1 Request

Additional application-properties

Key	Value Type	Mandatory?	Description
name	`string`	Exactly one of name or identity MUST be provided.	The name of the Manageable Entity to be managed. This is case-sensitive.
identity	`string`		The identity of the Manageable Entity to be managed. This is case-sensitive.

Body

No information is carried in the message body therefore any message body is valid and MUST be ignored.

3.3.4.2 Response

The body of the message MUST consist of an `amqp-value` section containing a `map` with zero entries.

If the request was successful then the statusCode MUST be 204 (No Content).

3.4 Standard Management Node Operations

A Management Node Operation is an operation directed to the Management Node itself rather than an entity it is managing.

Of the standard application-properties (see Section 3.1), name MUST be provided with a value of “self”, type MUST be provided with a value of “org.amqp.management” and identity MUST NOT be provided.

The following Management Node Operations SHOULD be supported:

- QUERY
- GET-TYPES
- GET-ANNOTATIONS
- GET-ATTRIBUTES
- GET-OPERATIONS
- GET-MGMT-NODES

The following Management Node Operations MAY be supported:

- REGISTER
- DEREGISTER

3.4.1 QUERY

Retrieve selected attributes of Manageable Entities that can be read at this Management Node.

Since the query operation could potentially return a large number of results, this operation supports pagination through which a request can specify a subset of the results to be returned.

A result set of size N can be considered to containing elements numbered from 0 to N-1. The elements of the result set returned in a particular request are controlled by specifying offset and count values. By setting an offset of M then only the elements numbered from M onwards will be returned. If M is greater than the number of elements in the result set then no elements will be returned. By additionally setting a count of C, only the elements numbered from M to Min(M+C-1, N-1) will be returned. Pagination is achieved via two application-properties, offset and count.

If pagination is used then it cannot be guaranteed that the result set remains consistent between requests for successive pages. That is, the set of entities matching the query may have changed between requests. However, stable order MUST be provided, that is, for any two queries for the same parameters (except those related to pagination) then the results MUST be provided in the same order. Thus, if there are no changes to the set of entities that match the query then consistency MUST be maintained between requests for successive pages.

3.4.1.1 Request

Additional application-properties

Key	Value Type	Mandatory?	Description
entityType	`string`	No	If set, restricts the set of Manageable Entities requested to those that extend (directly or indirectly) the given Manageable Entity Type.
offset	`integer`	No	If set, specifies the number of the first element of the result set to be returned. If not provided, a default of 0 MUST be assumed.

count	integer	No	If set, specifies the number of entries from the result set to return. If not provided, all results from 'offset' onwards MUST be returned.

Body

The body of the message MUST consist of an amqp-value section containing a map which MUST have the following entries, where all keys MUST be of type `string`:

Key	Value Type	Description
attributeNames	`list` (of `strings`)	A `list` of `strings` representing the names of the attributes of the Manageable Entities being requested. The `list` MUST NOT contain duplicate elements. If the `list` contains no elements then this indicates that all attributes are being requested.

3.4.1.2 Response

If the request was successful, then the statusCode MUST be 200 (OK) and the `application-properties` MUST contain the same set of `application-properties` that was provided in the request. The values MUST be the same as those requested except the following:

Key	Value Type	Description
count	`integer`	Specifies the number of entries from the result set being returned. Note that the value of count MUST be the same as number of elements in the `list` value associated with the results key in the body of the response message.

The body of the message MUST consist of an `amqp-value` section containing a `map` which MUST have the following entries, where all keys MUST be of type `string`:

Key	Value Type	Description
attributeNames	`list` (of `strings`)	A `list` of `strings` where each element represents an attribute name. If the attributeNames passed in the body of the request contained a non-empty `list` then this value MUST consist of the exact same sequence of `strings`. If the body of the request did not contain an attributeNames entry then this value MUST contain the union of all attribute names for all Manageable Entity Types that match the query.
results	`list` (of `lists`)	This value provides the portion of the result set being requested (as controlled by offset and count). Each element MUST provide the `list` of attribute values for a single Manageable Entity where the values are positionally-correlated with the names in the attributeNames entry. In the case where an attribute name is not applicable for a particular Manageable Entity then the corresponding value should be `null`. If the result set is empty then this value MUST be a `list` of zero elements.

3.4.2 GET-TYPES

Retrieve the list of Manageable Entity Types that can be managed via this Management Node.

3.4.2.1 Request

Additional application-properties

Key	Value Type	Mandatory?	Description
entityType	`string`	No	If set, restricts the list of Manageable Entity Types requested to those that extend (directly or indirectly) the given Manageable Entity Type.

Body

No information is carried in the message body therefore any message body is valid and MUST be ignored.

3.4.2.2 Response

If the request was successful then the statusCode MUST be 200 (OK) and the body of the message MUST consist of an `amqp-value` section containing a `map`. The keys in the map MUST be the set of Manageable Entity Types on which Management Operations can be performed. For any given key, the value MUST be a `list` of `strings` representing the Manageable Entity Types that this Manageable Entity Type extends.

3.4.3 GET-ANNOTATIONS

Retrieve the list of Type Annotations implemented by each Manageable Entity Type.

3.4.3.1 Request

Additional application-properties

Key	Value Type	Mandatory?	Description
entityType	`string`	No	If set, restricts the list of Manageable Entity Types for which data is returned to those that extend (directly or indirectly) the given Manageable Entity Type. If not set then the request is for all Manageable Entity Types that can be managed via this Management Node.

Body

No information is carried in the message body therefore any message body is valid and MUST be ignored.

3.4.3.2 Response

If the request was successful then the statusCode MUST be 200 (OK) and the body of the message MUST consist of an `amqp-value` section containing a `map`. The keys in the map MUST be the set of Manageable Entity Types on which Management Operations can be performed (or that subset of Manageable Entity Types for which information was requested). For any given key, the value MUST be a `list` of `strings` representing the names of the Type Annotations that this Manageable Entity Type implements.

3.4.4 GET-ATTRIBUTES

Retrieve the lists of attribute names for the given Manageable Entity Types.

3.4.4.1 Request

Additional application-properties

Key	Value Type	Mandatory?	Description
entityType	`string`	No	If set, restricts the request to include entries only for the Manageable Entity Type given.

Body

No information is carried in the message body therefore any message body is valid and MUST be ignored.

3.4.4.2 Response

If the request was successful then the statusCode MUST be 200 (OK) and the body of the message MUST contain a map. The keys in the map MUST be the set of Manageable Entity Types for which attribute names are being provided. For any given key, the value MUST be a `list` of `strings` representing the attribute names that this Manageable Entity Type possesses. It should be noted that for each entry in the `map`, the attribute names returned MUST be only those defined by the associated Manageable Entity Type rather than those that are defined by other Manageable Entity Types that extend it. For any given Manageable Entity Type, the set of attribute names returned MUST include every attribute name defined by Manageable Entity Types that it extends, either directly or indirectly.

3.4.5 GET-OPERATIONS

Retrieve the list of Management Operations (and the arguments they take) which can be performed via this Management Node.

3.4.5.1 Request

Additional application-properties

Key	Value Type	Mandatory?	Description
entityType	`string`	No	If set, restricts the request to include entries only for the Manageable Entity Type given.

Body

No information is carried in the message body therefore any message body is valid and MUST be ignored.

3.4.5.2 Response

If the request was successful then the statusCode MUST be 200 (OK) and the body of the message MUST consist of an `amqp-value` section containing a `map`. The keys in the map MUST be the set of Manageable Entity Types for which the list of Management Operations is being provided. For any given key, the value MUST itself be a `map`, where each key is the `string` name of a Management Operation that can be performed against this Manageable Entity Type via this Management Node, and the value for a given key is a `list` of `strings` giving the names of the arguments (passed via the `application-properties` of a request message) which the operation defines. For any given Manageable Entity Type, the set of operations returned MUST include every operation supported by Manageable Entity Types that it extends, either directly or indirectly.

3.4.6 GET-MGMT-NODES

Retrieve the list of addresses of other Management Nodes which this Management Node is aware of.

3.4.6.1 Request

Additional application-properties

None

Body

No information is carried in the message body therefore any message body is valid and MUST be ignored.

3.4.6.2 Response

If the request was successful then the statusCode MUST be 200 (OK) and the body of the message MUST consist of an `amqp-value` section containing a `list` of addresses of other Management Nodes known by this Management Node (each element of the `list` thus being a `string`). If no other Management Nodes are known then the `amqp-value` section MUST contain a `list` of zero elements.

3.4.7 REGISTER

Register a Management Node.

3.4.7.1 Request

Additional application-properties

Key	Value Type	Mandatory?	Description
address	`string`	Yes	Defines the address of the node being registered.

Body

No information is carried in the message body therefore any message body is valid and MUST be ignored.

3.4.7.2 Response

No information is carried in the message body therefore any message body is valid and MUST be ignored.

If the request was successful then the statusCode MUST be 200 (OK). Upon a successful registration, the address of the registered Management Node will be present in the `list` of known Management Nodes returned by subsequent GET-MGMT-NODES operations.

3.4.8 DEREGISTER

Delete the registration of a Management Node.

3.4.8.1 Request

Additional application-properties

Key	Value Type	Mandatory?	Description
address	`string`	Yes	Defines the address of the node being deregistered.

Body

The body of the message MUST be empty.

3.4.8.2 Response

No information is carried in the message body therefore any message body is valid and MUST be ignored.

If the request was successful then the statusCode MUST be 200 (OK). Upon a successful deregistration, the address of the unregistered Management Node will not be present in the `list` of known Management Nodes returned by subsequent GET-MGMT-NODES operations.

4 Request / Response Pattern

AMQP Management Operations follow a request response pattern using Transfer messages after a Connection, Session, and Links have been established

TODO: Is this section sufficiently documented in “Attach to the Management Node” (5.1) ?

- Create link to request node
  - attach(src=null; tgt=”q1”)
- Link to response queue/creating the back channel
  - attach(tgt=<client_container>$<client_generated_id>; src=”q1”)
- Message.reply_to=”<client_container>$<client_generated_id>”
- Separator is a “topological separator”
  - Need to define separator, $ is a placeholder
- Note this is not synchronous correlated request/response. This is not only for RPC.
  - Multiple “response” messages may be initiated for a single “request”

5 Examples

TODO – this section needs a refresh following changes made above. Ignore for now.

This section is non-normative.

The following examples use pseudo code. AMQP performative and type names correspond to definitions in the **[AMQP]** specification.

5.1 Attach to the Management Node

```
// create a link to the management node for sending management requests
requestLink = session.attach(
       role: SENDER,
       target: { address: "$management" }
)

// create a link for receiving responses from the management node
responseLink = session.attach(
       role: RECEIVER,
       target: { address: "$management" },
       source: { address: "/myaddress" }
)
```

TODO: need explanation of /myaddress address; refer to global addressing.

5.2 Create a Resource

The below example illustrates successful creation of a resource.

```
// transfer a request message
requestLink.sendTransfer(
       Message(
              properties: {
                     correlation-id: 1,
                     to: "$management",
                     reply-to: "/myaddress"
              },
              application-properties: {
                     "name" -> "newQueue",
                     "operation" -> "CREATE",
                     "type" -> "org.example.queue"
```

```
            },
            application-data: AmqpValue(
                  Map(
                        // type specific properties
                        "max_size" -> "2000Mb"
                  )
            )
      )
)

// Receive the response message from the response link
responseMessage = responseLink.receiveTransfer()

// responseMessage will be of the form:
//    Message(
//          properties: {
//                correlation-id: 1,
//                to: "/myaddress"
//          },
//          application-properties: {
//                "operation" -> "CREATE",
//                "statusCode" -> 201,
//                "statusDescription" -> "Created",
//                "type" -> "com.example.broker.queue"
//          },
//          application-data: AmqpValue(
//                Map(
//                      // type specific properties
//                      "name" -> "newQueue",
//                      "identity" -> "1234567",
//                      "type" -> "com.example.broker.queue",
//                      "num_priorities" -> 4,
//                      "max_size" -> "2000Mb"
//                )
//          )
//    )
// )
```

5.3 Read a Resource

The below example illustrates successful reading of a resource.

```
// transfer a request message
requestLink.sendTransfer(
       Message(
              properties: {
                     message-id: 73,
                     to: "$management",
                     reply-to: "/myaddress"
              },
              application-properties: {
                     "name" -> "myQueue",
                     "operation" -> "READ",
                     "type" -> "com.example.broker.queue"
              },
       )
)

// Receive the response message from the response link
responseMessage = responseLink.receiveTransfer()

// responseMessage will be of the form:
//     Message(
//            properties: {
//                   correlation-id: 73,
//                   to: "/myaddress"
//            },
//            application-properties: {
//                   "operation" -> "READ",
//                   "statusCode" -> 200,
//                   "statusDescription" -> "OK",
//            },
//            application-data: AmqpValue(
//                   Map(
//                          // type specific properties
//                          "name" -> "myQueue",
//                          "identity" -> "9876543",
//                          "type" -> "com.example.broker.queue",
//                          "num_priorities" -> 4,
//                          "max_size" -> "2000Mb"
//                   )
//            )
//     )
```

5.4 Update a Resource

The below example illustrates successful updating of a resource.

```
// transfer a request message
requestLink.sendTransfer(
       Message(
              properties: {
                     correlation-id: 3,
                     to: "/myQueue$management",
                     reply-to: "/myaddress"
              },
              application-properties: {
                     "name" -> "myQueue",
                     "operation" -> "UPDATE",
                     "type" -> "com.example.broker.queue"
              },
              application-data: AmqpValue(
                     Map(
                            "max_size" -> "3000Mb"
                     )
              )
       )
)

// Receive the response message from the response link
responseMessage = responseLink.receiveTransfer()

// responseMessage will be of the form:
//     Message(
//            properties: {
//                   correlation-id: 3,
//                   reply-to: "/myaddress"
//            },
//            application-properties: {
//                   "operation" -> "UPDATE",
//                   "statusCode" -> 200,
//                   "statusDescription" -> "OK",
//            },
```

```
//           application-data: AmqpValue(
//                 Map(
//                       // type specific properties
//                       "name" -> "myQueue",
//                       "identity" -> "9876543",
//                       "type" -> "com.example.broker.queue",
//                       "num_priorities" -> 4,
//                       "max_size" -> "3000Mb" // the max_size is updated
//                 )
//           )
//     )
```

5.5 Update a Resource (but fail)

The below example illustrates a response if an update operation fails.
In this case, the user attempts to change the number of priorities allowed by the queue.

```
// transfer a request message
requestLink.sendTransfer(
      Message(
            properties: {
                  correlation-id: 37,
                  to: "$management",
                  reply-to: "/myaddress"
            },
            application-properties: {
                  "name" -> "myQueue",
                  "operation" -> "UPDATE",
                  "type" -> "com.example.broker.queue"
            },
            application-data: AmqpValue(
                  Map(
                        "num_priorities" -> "5"
                  )
            )
      )
)

// Receive the response message from the response link
responseMessage = responseLink.receiveTransfer()
```

```
// responseMessage will be of the form:
//     Message(
//           properties: {
//                 correlation-id: 37,
//                 reply-to: "/myaddress"
//           },
//           application-properties: {
//                 "operation" -> "UPDATE",
//                 "statusCode" -> 400,
//                 "statusDescription" -> "Bad Request: Cannot update number of
//                                         priority levels",
//           }
//     )
```

5.6 Delete a Resource

The below example illustrates successful deleting of a resource.

```
// transfer a request message
requestLink.sendTransfer(
      Message(
            properties: {
                  correlation-id: 4,
                  to: "$management",
                  reply-to: "/myaddress"
            },
            application-properties: {
                  "name" -> "myQueue",
                  "operation" -> "DELETE",
                  "type" -> "com.example.broker.queue"
            }
      )
)

// Receive the response message from the response link
responseMessage = responseLink.receiveTransfer()

// responseMessage will be of the form:
//     Message(
```

```
//            properties: {
//                  correlation-id: 4,
//                  reply-to: "/myaddress"
//            },
//            application-properties: {
//                  "operation" -> "DELETE",
//                  "statusCode" -> 204,
//                  "statusDescription" -> "No Content",
//            }
//      )
```

5.7 Delete a Resource (but fail)

The below example illustrates a response if a delete operation fails.
In this case, the user attempts to delete a resource that does not exist.

```
// transfer a request message
requestLink.sendTransfer(
      Message(
            properties: {
                  correlation-id: 49,
                  to: "$management",
                  reply-to: "/myaddress"
            },
            application-properties: {
                  "name" -> "myQueue",
                  "operation" -> "DELETE",
                  "type" -> "com.example.broker.queue"
            }
      )
)

// Receive the response message from the response link
responseMessage = responseLink.receiveTransfer()

// responseMessage will be of the form:
//      Message(
//            properties: {
//                  correlation-id: 49,
//                  reply-to: "/myaddress"
//            },
//            application-properties: {
```

```
//                  "operation" -> "DELETE",
//                  "statusCode" -> 404,
//                  "statusDescription" -> "Not Found",
//           }
//     )
```

5.8 Read All Resources

The below example illustrates successful reading of all resources .

```
// transfer a request message
requestLink.sendTransfer(
      Message(
            properties: {
                  message-id: 105,
                  to: "$management",
                  reply-to: "/myaddress"
            },
            application-properties: {
                  "name" -> "self",
                  "operation" -> "READALL",
                  "type" -> "org.amqp.management"
            },
      )
)

// Receive the response message from the response link
responseMessage = responseLink.receiveTransfer()

// responseMessage will be of the form:
//     Message(
//           properties: {
//                 correlation-id: 105,
//                 to: "/myaddress"
//           },
//           application-properties: {
//                 "operation" -> "READALL",
//                 "statusCode" -> 200,
//                 "statusDescription" -> "OK",
//           },
//           application-data: AmqpValue(
//                 List[
```

```
//                          Map(
//                                 // type specific properties
//                                 "name" -> "myQueue",
//                                 "identity" -> "9876543",
//                                 "type" -> "com.example.broker.queue",
//                                 "num_priorities" -> 4,
//                                 "max_size" -> "3000Mb"
//                          ),
//                          Map(
//                                 // type specific properties
//                                 "name" -> "newQueue",
//                                 "identity" -> "1234567",
//                                 "type" -> "com.example.broker.queue",
//                                 "num_priorities" -> 4,
//                                 "max_size" -> "2000Mb"
//                          )
//                   ]
//            )
//     )
```

5.9 Discover Names

The below example illustrates successful discovery of types.

```
// transfer a request message
requestLink.sendTransfer(
       Message(
              properties: {
                     message-id: 132,
                     to: "$management",
                     reply-to: "/myaddress"
              },
              application-properties: {
                     "name" -> "self",
                     "operation" -> "DISCOVER-NAMES",
                     "type" -> "org.amqp.management"
              },
       )
)

// Receive the response message from the response link
responseMessage = responseLink.receiveTransfer()
```

```
// responseMessage will be of the form:
//     Message(
//           properties: {
//                 correlation-id: 132,
//                 to: "/myaddress"
//           },
//           application-properties: {
//                 "operation" -> "DISCOVER-NAMES",
//                 "statusCode" -> 200,
//                 "statusDescription" -> "OK",
//           },
//           application-data: AmqpValue(
//                 Map(
//                       "com.example.broker.priorityqueue" -> List[
//                                         "myQueue",
//                                         "newqueue"
//                                   ],
//
//                       "com.example.broker.queue" -> List[
//                                         ...
//                                   ],
//
//                 )
//           )
//     )
```

5.10 Discover Types

The below example illustrates successful discovery of types.

```
// transfer a request message
requestLink.sendTransfer(
      Message(
            properties: {
                  message-id: 132,
                  to: "$management",
                  reply-to: "/myaddress"
            },
            application-properties: {
                  "name" -> "self",
                  "operation" -> "DISCOVER-TYPES",
```

```
                  "type" -> "org.amqp.management"
            },
      )
)

// Receive the response message from the response link
responseMessage = responseLink.receiveTransfer()

// responseMessage will be of the form:
//    Message(
//          properties: {
//                correlation-id: 132,
//                to: "/myaddress"
//          },
//          application-properties: {
//                "operation" -> "DISCOVER-TYPES",
//                "statusCode" -> 200,
//                "statusDescription" -> "OK",
//          },
//          application-data: AmqpValue(
//                Map(
//                      "com.example.broker.priorityqueue" ->
//                                  List[
//                                        "org.amqp.queue"
//                                        "com.example.broker.priorityqueue",
//                                  ],
//
//                      "com.example.broker.queue" ->
//                                  List[
                                        ...
//                                  ],
//
//                )
//          )
//    )
```

5.11 Discover Operations

The below example illustrates successful discovery of operations.

```
// transfer a request message
requestLink.sendTransfer(
      Message(
```

```
            properties: {
                  message-id: 132,
                  to: "$management",
                  reply-to: "/myaddress"
            },
            application-properties: {
                  "name" -> "self",
                  "operation" -> "DISCOVER-OPERATIONS",
                  "type" -> "org.amqp.management"
            },
      )
)

// Receive the response message from the response link
responseMessage = responseLink.receiveTransfer()

// responseMessage will be of the form:
//    Message(
//          properties: {
//                correlation-id: 132,
//                to: "/myaddress"
//          },
//          application-properties: {
//                "operation" -> "DISCOVER-OPERATIONS",
//                "statusCode" -> 200,
//                "statusDescription" -> "OK",
//          },
//          application-data: AmqpValue(
//                Map(
//                      "com.example.broker.priorityqueue" ->
//                                  List[
//                                        "CREATE",
//                                        "DELETE",
//                                        "READ",
//                                        "UPDATE"
//                                  ]
//
//                      "com.example.broker.queue" ->
//                                  List[
//                                        ...
//                                  ]
//                )
```

```
//           )
//     )
```

5.12 Discover Management Nodes

The below example illustrates successful discovery of all management nodes.

```
// transfer a request message
requestLink.sendTransfer(
       Message(
              properties: {
                     message-id: 152,
                     to: "$management",
                     reply-to: "/myaddress"
              },
              application-properties: {
                     "name" -> "self",
                     "operation" -> "DISCOVER-MGMT-NODES",
                     "type" -> "org.amqp.management"
              },
       )
)

// Receive the response message from the response link
responseMessage = responseLink.receiveTransfer()

// responseMessage will be of the form:
//     Message(
//            properties: {
//                   correlation-id: 152,
//                   to: "/myaddress"
//            },
//            application-properties: {
//                   "operation" -> "DISCOVER-MGMT-NODES",
//                   "statusCode" -> 200,
//                   "statusDescription" -> "OK",
//            },
//            application-data: AmqpValue(
//                   List[
//                          "amqp:MasterNode",
//                          "amqp:/SuperNode",
//                          "amqp://example.com/AdminNode"
//                          // TODO: Update these when Addressing is resolved.
```

```
//                  ]
//           )
//     )
```

5.13 Register a Management Node

The below example illustrates successful registration of a management node.

```
// transfer a request message
requestLink.sendTransfer(
       Message(
              properties: {
                     correlation-id: 173,
                     to: "$management",
                     reply-to: "/myaddress"
              },
              application-properties: {
                     "name" -> "myManagedToaster",
                     "operation" -> "REGISTER",
                     "type" -> "com.example. managedtoaster"
              }
       )
)

// Receive the response message from the response link
responseMessage = responseLink.receiveTransfer()

// responseMessage will be of the form:
//     Message(
//            properties: {
//                   correlation-id: 173,
//                   reply-to: "/myaddress"
//            },
//            application-properties: {
//                   "operation" -> "REGISTER",
//                   "statusCode" -> 200,
//                   "statusDescription" -> "OK",
//            }
//     )
```

5.14 Deregister a Management Node

The below example illustrates successful deleting of registration of a management node.

```
// transfer a request message
requestLink.sendTransfer(
      Message(
            properties: {
                  correlation-id: 194,
                  to: "$management",
                  reply-to: "/myaddress"
            },
            application-properties: {
                  "name" -> "myToasterManager",
                  "operation" -> "DEREGISTER",
                  "type" -> "com.example.managedtoaster"
            }
      )
)

// Receive the response message from the response link
responseMessage = responseLink.receiveTransfer()

// responseMessage will be of the form:
//     Message(
//           properties: {
//                 correlation-id: 194,
//                 reply-to: "/myaddress"
//           },
//           application-properties: {
//                 "operation" -> "DEREGISTER",
//                 "statusCode" -> 200,
//                 "statusDescription" -> "OK",
//           }
//     )
     x
```

6 # Conformance

The last numbered section in the specification must be the Conformance section. Conformance Statements/Clauses go here. [Remove # marker]

7 # Outstanding “to dos”

- Be consistent about specifying case sensitivity
- Clarify the message body types and improve “ignore” language.
- ~~All types should be lowercase – use fixed-width font to identify~~
- Examples
  - Clean up examples and check accuracy with updated specification.
  - Query pagination
- Reformat operation response description consistent with request description, i.e., body and application-properties.
- Reformat method descriptions in the form of request, processing rules and response.
- Reference an external registry of Manageable Entity Types
- Add more clarification on ‘type system’ of Manageable Entity Types.
- ~~State that for unsuccessful operations, body SHOULD be ignored. Its contents can be implementation specific.~~

- ~~Identity vs. name~~
  - ~~Identity~~
    - ~~Immutable~~
    - ~~Server created~~
    - ~~Unique within the set of manageable entities that are accessible from a given management node~~
      - ~~Note that in reality, for a given implementation, the scope of uniqueness may be larger, e.g., across a set of management nodes or even globally unique~~
  - ~~Name~~
    - ~~User created~~
    - ~~Mutable~~
    - ~~Unique within the set of manageable entities that are accessible from a given management node~~
      - Note however that you might see the same entity with the same identity through different management nodes appearing with different names
- ~~Types~~
  - ~~Distinguish between ‘types’ and ’interfaces’~~
  - ~~Type – description of its fundamental nature, e.g., a queue or a user~~
  - ~~Interface – groups together a set of attributes and/or operations, e.g., ordered, prioritized, stoppable etc.~~
  - ~~Distinctions~~
    - ~~Makes no sense to query on interfaces~~
    - ~~Query on applies to ‘types’~~
    - ~~Can inherit from a single type but multiple interfaces, cf. Java~~
- Events

    - All down to Ted
- Document structure
    - Describe operations in terms of request, processing rules and response
- Examples
    - Re do

Appendix A. Acknowledgments

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

Participants:

Matthew Arrott, Individual

Rob Dolin, Microsoft

Robert Godfrey, JP Morgan

Steve Huston, Riverace

David Ingham, Red Hat

James Kirkland, Red Hat

Alex Kritikos, Software AG

Dale Moberg, Axway Software

Andreas Moravec, Deutsche Boerse

Rafael Schloming, Red Hat

Jakub Scholz, Deutsche Boerse

Wolf Tombe, US Department of Homeland Security

The following individuals were members of the OASIS Advanced Message Queueing Protocol (AMQP) Technical Committee during the creation of this specification and their contributions are gratefully acknowledged:

Sanjay Aiyagari, VMware, Inc.

Matthew Arrott, Individual

Allan Beck, JPMorgan Chase Bank, N.A.

Laurie Bryson, JPMorgan Chase Bank, N.A.

Raphael Cohn, Individual

Rob Dolin, Microsoft

Robert Gemmell, JPMorgan Chase Bank, N.A.

Rob Godfrey, JPMorgan Chase Bank, N.A.

William Henry, Red Hat

Steve Huston, Individual

David Ingham, Microsoft

Ram Jeyaraman, Microsoft

James Kirkland, Red Hat

Alex Kritikos, Software AG, Inc.

Dale Moberg, Axway Software

Andreas Moravec, Deutsche Boerse AG

Suryanarayanan Nagarajan, Software AG, Inc.

John O'Hara, Individual

Jonathan Poulter, Kaazing

Sandeep Puri, Cisco Systems

Oleksandr Rudyy, JPMorgan Chase Bank, N.A.

Rafael Schloming, Red Hat
Jakub Scholz, Deutsche Boerse AG
Angus Telfer, INETCO Systems Ltd.
Wolf Tombe, US Department of Homeland Security

Appendix B. Non-Normative Text

text

B.1 Subsidiary section

text

B.1.1 Sub-subsidiary section

text

Appendix C. Revision History

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