Table of Contents

1 Introduction

This document intended for application developers who are going to integrate this API with other external services such as OpenStack. It documents in detail a RESTful API to configure virtual computing resources and to manage their state.
It documents in detail a RESTful API to configure relay functions and access control lists of a virtual Layer 2 switch.

2 Scope

This document specifies the interface to configure, access and manage virtual computing resources.

Through its abstract interface it aims to simplify the control and management of such resources.

This document specifies the interface to configure, access and manage virtual resources to give users a granular control over routing decisions to either restrict or allow traffic.

This API does not support state management of resources hence no mechanism exists to subscribe to receive events from managed resources.

The API does not support any guards preventing access to objects created by other clients. It could be that higher level APIs provide the necessary authentication but this is outside the scope of this API.

This API although it is in the best interests does not support yet state management of resources hence no mechanism exists to subscribe to receive events from managed resources.

3 Overview

The virtual network management system (VNMS) consists of a stack of resources controlled in a hierarchically way. Resources at a lower level can not be accessed unless resources at the higher level are accessible. Such dependencies should be well understood because side effects might occur. For example deletion of a network can result in all other dependent instances disappearing as well. Users interact with this API through its supported addressing model and are unaware of the underlying network infrastructure that are using. Briefly this network infrastructure is a virtual layer 2 designed within the Trema framework supporting a flexible and extensible configuration

management. It aims to communicate between VMs on a single physical host or to connect VMs to an external network.

4 Definitions

For the purpose of this document the following terms and definitions apply.

4.1 Virtual network management system (VNMS)

It is a conceptual term used to describe the system that this API is targeted for.

4.2 Tenant

Within the VNMS a top level entity defining a large pool of network resources. A tenant may span to cover one or more network entities. Is this true?

4.3 Network

Models the relationship of a virtual resource pool (ports/attachments) that this entity may have.

4.4 Port

Mapping of port attributes to this logical entity to enable network connectivity.

4.5 Attachment

An entity that as its name suggests can either be attached to a network or port entity objects.

This entity represents a mapping to a value that exposes a control attribute to the entity being attached (network/port).

4.6 Entity/Element

Terms used interchangeably referring to objects that this API manipulates.

4.7 Container

A simple grouping of objects of certain type with the same characteristics.

5 Types of resources in the VNMS

Resource type	Description
Tenant	A container of one or more objects uniquely addressable by its tenant identifier.
Network	A container of one or more objects uniquely addressable by its network identifier.
Port	A container of one or more objects addressable by its logical port identifier. A port container may also have attachment container objects.
Attachment	An entity that represents a binding of key/value pairs that can be attached either to network or port. Attachment also can be viewed as a container accessible through its attachment identifier.

Resource access applies to all synchronous operations regarding getting, setting and enumerating values. Subclauses below define a mechanism for acquiring JSON based representations of entities using this RESTful API. A URI syntax may indicate the type of entity being enumerated or a particular selector object.

6 Network

A network can be considered as a container of resources within a tenant entity. A network may connect a set of virtual resources either from a single physical resource or from multiple physical resources. A network is identified by a unique id, auto-generated if not given that remains constant for the life of the entity. Every network object is associated with a tenant object from which inherits all its metadata. In the examples that will follow the URI path that signifies tenant information is omitted for clarity and hereon indicated as <root-URI>.

6.1 Create a network object

A create operation creates a network object with its initial representation. If the target network object already exists the create operation would fail returning an error.

This creates and initializes a virtual layer 2 without any functionality for connectivity yet.

The following HTTP POST creates a network object at the specified URI.

Example: POST <root-URI>/networks

```
POST /networks HTTP/1.1
Content-type: application/json
{
      “id” : “net-1”,
      “description” : “marketing department”
}
```

Where:

- root-URI – reference to tenant target object http://localhost/tenants/<tenant-id>. From now on the tenant path would be omitted.
- networks - the network child object to create in the networks container. There is no guarantee that the newly network object would be sorted while added to the list of existing network objects.

The POST request includes two name/value pairs, id and description.

Table 1 Request message body – Create a network object

Field name	Type	Description	Requirement
id	JSON String	Any textual description as long as it is unique which is used to identify the object.	Optional
description	JSON String	A name that this network is known as.	Mandatory

6.1.1 Response status

The HTTP status codes that returned when creating a network object are described in Table 2.

Table 2 HTTP status codes – Creating a network object.

HTTP Status	Description
202 Created	Operation succeeded. A new network object is created.
404 Not found	A tenant resource was not found.
422 Bad request	Invalid parameters or field names or attempt to create an existing network object.

500 Internal error	A processing error while adding a network object.

6.2 Read multiple network objects

The following HTTP GET reads one or more network objects at the specified URI. The message is targeted to return an array of network objects. If an object can not be retrieved due to conflicting locking or simultaneous access or any other conflict an error response is returned. If the request results in no data being found an empty response is returned (an empty array with no values).

Example: Perform a GET to the network container URI.

```
GET /networks HTTP/1.1
Accept: */*
```

The following shows the server's response.

```
HTTP/1.1 200 OK
Content-Type: application/json
[
      {"id":"net-1","description":"marketing department"},
      {"id":"net-2","description":"development department"}
]
```

6.2.1 Response headers

The following response headers expected as shown in Table 3.

Table 3 Response headers – Read multiple network objects

Header	Type	Requirement
Content-Type	application/json	If present a JSON array of zero, one or more network objects.
Content-Type	text/plain	If present an implementation specific error message.

6.3 Read a network object

The following HTTP GET reads from an existing network object at the specified URI.

Example: Perform a GET to the network object URI.

```
GET /networks/<net-id> HTTP/1.1
Accept: */*
```

Where:

- <net-id> is the identifier of the network object (for example net-2).

The following shows the server's response.

```
HTTP/1.1 200 OK
Content-Type: application/json
{
      "bindings":[],
      "description":"development department"
}
```

6.3.1 Response headers

The following response headers expected as shown in Table 4.

Table 4 Response headers – Read a single reference of a network object

Header	Type	Requirement
Content-Type	application/json	Optional. If present the referenced network object instance.
Content-Type	text/plain	Optional. If present the referenced network object instance is not found.

6.3.2 Response message body

The response message body for reading a specific network object is shown in Table 5.

Table 5 Response message body – Read a network object

Field name	Type	Description	Requirement
bindings	JSON Array	Any port or attachment object associated with this network instance. If there are no bindings an empty array is returned.	Mandatory
description	JSON String	A readable description of this network object instance.	Mandatory

6.4 Delete a network

It is an operation to delete a network resource in its entirety. The delete operation specifies a precondition to match in order for the operation to be carried out. This does not mean that the

operation would succeed other errors may prevent and cause the delete operation to fail.

The following HTTP DELETE deletes a network object at the specified URI. Any dependent entities visible to this network object (ports, attachments) would also be deleted.

DELETE /networks/<net-id>

Where:

- <net-id> the name of a network object to be deleted (for example net-2).

6.4.1 Response status

The HTTP status code that occur when deleting a network object shown in Table 6.

Table 6 HTTP status codes – Delete a network object

HTTP Status	Description
202 Accepted	A network object successfully deleted.
404 Not Found	A network object already deleted or is not found.
500 Internal Server Error	A processing error while deleting the network object.

6.5 Modify a network object

Ability to modify an existing network object. A successful PUT operation would modify the network object modifiable properties and leave the other properties unchanged. If the targeting PUT network representation and the actual saved representation do not differ the object is left intact. A modification to a network object should not disturb the preservation order of other objects. A client user must query again to display the updated results.

Example: PUT to the network object URI to alter its description.

```
PUT /networks/net-1 HTTP/1.1
Content-type: application/json
{“description”:”another department”}
```

The following shows the server's response.

```
HTTP/1.1 202 Accepted
Content-Type: text/plain; charset=ISO-8859-1
```

6.5.1 Response status

The HTTP status code that occur when modifying a network object shown in Table 7.

Table 7 HTTP status codes – Modifying a network object

HTTP Status	Description
202 Accepted	A network object is modified.
404 Not Found	An attempt is made to modify a non-existing network object.
500 Internal Server Error	A processing error while updating the network object.

7 Port

A port object describes a set of properties a virtual resource must have in order to connect to the network entity. This membership declaration allows frames from the virtual resource to be propagated among other registered resources.

A ports URI allows access to specific port objects by referencing its id. A client user can create inquire about the configuration or delete port object instances.

A port object describes a connection of a virtual resource to the network entity. It is a control element mapping logical ports to physical ports to be used in a much wider context than otherwise could.

A ports URI allows access to specific port objects by referencing its id. A client user can create inquire about the configuration or delete port object instances.

7.1 Create a port object

Create a port by defining a number of key/values attributes. The created port should be accessible either by the ports container or by its unique id.

Example:The following HTTP POST creates a port object at the specified URI.

```
POST /networks/net-1/ports HTTP/1.1
Content-type: application/json
{
       "id":"port-1",
       "datapath_id":"1234",
       "port":1,
       "vid":1024
}
```

The POST request contains the following name/value pairs.

Table 8 Request message body – Create a port object

Field name	Type	Description	Requirement
id	JSON String	Any textual description as long as it is unique within the scope of this network object instance. Although optional client users may prefer to set this to create a more human-readable identifier.	Optional
datapath_id	JSON String	An id that uniquely identifies the data path of the virtual resource. Shall we use the word Openflow or avoid all together?	Mandatory
port	JSON Number	An ingress physical port where frames are received.	Mandatory
vid	JSON:Number	Assign a VLAN identifier to a port. Set this value to 65535 to skip VLAN setting. The VLAN tags let you logically separate traffic on a port into multiple channels.	Mandatory

On successful creation the following HTTP response is returned.

```
HTTP/1.1 202 Accepted
Content-Type: text/plain; charset=ISO-8859-1
```

7.1.1 Response status

The HTTP status codes that returned when creating a port object are described in Table 9.

Table 9 HTPP Status – Create a port object

HTTP Status	Description
202 Accepted	The referenced port object successfully added.
422 Unprocessable Entity	Attempt to create an existing port object.
404 Not Found	Reference to non-existing network object is found.
500 Internal Server Error	A processing error while adding a port object. A common cause of this error is when the parameter list not complete.

7.2 Read multiple port objects

It is a query to return all port objects belonging to a specific network. The following HTTP GET reads one or more port objects at the specified URI. If the request results in no data being found an empty response is returned (an empty array with no values).

To find out what ports are available for a network.

Example: Perform a GET to the port container at the network's URI.

```
GET /networks/net-1/ports HTTP/1.1
Accept: */*
```

The following shows the server's response.

```
HTTP/1.1 200 OK
Content-Type: application/json
[
      {
            "vid":1024,
            "datapath_id":"1234",
            "id":"port-1","port":1
      }
]
```

7.2.1 Response headers

The following response headers expected as shown in Table 10.

Table 10 Response headers – Read multiple port objects

Header	Type	Requirement
Content-Type	application/json	If present a JSON array of zero one or more JSON port objects.
Content-Type	text/plain	If present an implementation specific error message.

7.2.2 Response message body

The response message body for enumerating multiple port objects of a JSON port object is shown in

Table 11.

Table 11 Response message body – Read multiple port objects

Field name	Type	Description	Requirement
vid	JSON Number	The VLAN id attached to a port	Mandatory
datapath_id	JSON String	A unique identifier of a virtual switch.	Mandatory
id	JSON String	A name this port is known as.	Mandatory
port	JSON Number	An index into a list of physical ports.	Mandatory

7.3 Read a port object

The following HTTP GET reads the attributes of an existing port object at the specified URL.

Example: Perform a GET to the port object URI.

```
GET /networks/net-1/ports/<port-id> HTTP/1.1
Accept: */*
```

Where:

- <port-id> is the unique port identifier (example port-1).

The following shows the server's response.

```
HTTP/1.1 200 OK
Content-Type: application/json
{
      "config":{"vid":1024,"datapath_id":"1234","port":1}
}
```

7.3.1 Response headers

The following response headers expected as shown in Table 12

Table 12 Response headers – Read a port object

Header	Type	Requirement
Content-Type	application/json	If present a JSON port object.
Content-Type	text/plain	If present an implementation specific error message.

7.3.2 Response status

The HTTP status codes returned when querying a port object are shown in Table 13

Table 13 HTTP Status – Read a port object.

HTTP Status	Description

202 OK	A port object successfully located.
404 Not Found	A port object was not found at the specified URI.
500 Internal Server Error	A processing error while querying a port object.

7.3.3 Response message body

The response message body for reading a particular port object is shown in Table 14.

Table 14 Response message body – Read a port object.

Field name	Type	Description	Requirement
config	JSON object	An object encapsulating the port instance with the following attributes: • vid • datapath_id • port	Mandatory

7.4 Delete a port object

The following HTTP DELETE deletes an existing port object at the specified URI.

DELETE /networks/<net-id>/ports/<port-id>

Where:

- <net-id> the network id which identifies a network object instance.
- <port-id> the port id which identifies a port object instance.

Example: Perform a DELETE to the port object URI.

```
DELETE /networks/net-1/ports/0000000004d2:0002:0100 HTTP/1.1
Accept: */*
```

The following shows the server's response.

```
HTTP/1.1 202 Accepted
Content-Length: 0
Content-Type: text/plain; charset=ISO-8859-1
```

This API does not define any addressing schemes or techniques for batched port delete operations. An attempt to delete all ports should fail with a 405 (Method not Allowed) error.

7.4.1 Response status

The HTTP status codes that occur when deleting a port object shown in Table 15

Table 15 Response status – Delete a port object

HTTP Status	Description
202 Accepted	A port object successfully deleted.
404 Not Found	A port object is already deleted or not found.
500 Internal Server Error	An implementation specific error message if unable to dispose the port object.

8 Attachment

Attachment is an abstraction that allows users to limit the extent of operation of virtual layer 2 by establishing boundaries dictating which specific MAC addresses are forwarded.

Attaching a network or a port object imposes a restriction on the amount of traffic since limiting frames to specific MAC addresses.

All operations and data semantics on the attachment object are the same regardless of the object they refer to (network/port). Different level of indirection achieves information independence between the two entities.

Attachment is an abstraction to create a virtual Layer 2 by registering a MAC address to a logical entity. The logical entity can either be an existing network object or a port object as described above. Each port attaches a host to a network that is capable of providing bidirectional connectivity for MAC user frames. Also a separate individual MAC address can be associated with each instance of network. That MAC address is used as the source address of frames transmitted by the attachment object.

All operations and data semantics on the attachment object are the same regardless of the object they refer to (network/port). Different level of indirection achieves information independence between the two entities.

8.1 Create an attachment object to a port object

To create a new attachment you use a HTTP POST operation.

Example: The following HTTP POST creates an attachment object at the specified URI.

```
POST /networks/net-1/ports/port-1/attachments HTTP/1.1
Content-type: application/json
{
      "id": "attachment-1",
      "mac":"11:22:33:44:55:66"
}
```

The POST request contains the following name/value pairs.

Table 16 Request message body – Create an attachment object

Field name	Type	Description	Requirement
id	JSON string	Specifies an identifier which is a user defined string that uniquely addresses this attachment.	Optional
mac	JSON string	A 6-octet MAC address value used as a unique identifier for the network/port that contains this attachment. Determines frame forwarding	Mandatory

		behavior on MAC addresses.	

On successful creation the following HTTP response is returned.

```
HTTP/1.1 202 Accepted
Content-Type: text/plain; charset=ISO-8859-1
```

The HTTP status codes that returned when creating an attachment object are described in Table 17.

Table 17 Http status – Create an attachment object to a port object

Http status	Description
202 Accepted	An attachment object successfully created.
422 Unprocessable Entity	Attempt to create an existing attachment object or invalid parameters specified or reference to non-existing port object.
404 Not Found	Reference to non-existing network object is found.
500 Internal Server Error	A processing error while adding an attachment object.

8.2 Read multiple port attachment objects

To return a list of all port attachment objects perform a HTTP GET on the specified URI.

Example: Perform a GET to the attachments container URI.

```
GET /networks/net-1/ports/port-1/attachments HTTP/1.1
Accept: */*
```

The following shows the server's response.

```
HTTP/1.1 200 OK
Content-Type: application/json
[
      {
            "id":"attachment-1",
            "mac":"11:22:33:44:55:66"
      }
]
```

8.2.1 Response headers

The following response headers expected as shown in Table 18.

Table 18 Response headers – Read multiple port attachment objects

Header	Type	Requirement
Content-Type	application/json	If present a JSON array of zero one, or more JSON attachment objects.
Content-Type	text/plain	If present an implementation specific error message.

8.3 Delete a port attachment object

The following HTTP DELETE deletes an existing attachment object at the specified network and port URI.

DELETE /networks/<net-id>/ports/<port-id>/attachments/<attachment-id>

Where:

- <net-id> the id which identifies a network object instance.
- <port-id> the id which identifies a port object instance.
- <attachment-id> the id of the port attachment object to be deleted.

Example: Perform a DELETE to the port attachment object URI.

```
DELETE /networks/net-1/ports/0000000004d2:0003:ffff/attachments/123 HTTP/1.1
Accept: */*
```

The following shows the server's response:

```
HTTP/1.1 202 Accepted
Content-Length: 0
Content-Type: text/plain; charset=ISO-8859-1
```

8.3.1 Response status

The HTTP status codes that occur when deleting a port attachment object shown in Table 19.

Table 19 Response status – Delete a port attachment object

HTTP Status	Description
202 Accepted	A port attachment object successfully deleted.
404 Not Found	A port attachment object is not found.
500 Internal Server Error	A processing error while deleting a port attachment object.

8.4 Create an attachment object to a network object

To create as new attachment object to a network object you use the HTTP POST operation. An attachment to a network can restrict or permit access across multiple networks.

Example: The following HTTP POST creates an attachment object at the specified URI.

```
POST /networks/net-1/attachments HTTP/1.1
Content-type: application/json
{
      "id":"test-attachment",
      "mac":"82:fe:91:b1:d6:36"
}
```

The POST request contains the same name/values pairs as shown in Table 16.

On successful creation the following HTTP response is returned.

```
HTTP/1.1 202 Accepted
Content-Length: 0
Content-Type: text/plain; charset=ISO-8859-1
```

For the HTTP status codes returned see Table 17.

8.5 Read multiple network attachment objects

To read a group of network attachment objects perform a HTTP GET on the specified URI.

Example: Perform a GET to the network attachments container URI.

```
GET /networks/net-1/attachments HTTP/1.1
Accept: */*
```

The following shows the server's response.

```
HTTP/1.1 200 OK
Content-Type: application/json
[
      {
            "id":"test-attachment",
            "mac":"82:fe:91:b1:d6:36"
      },
      {
            "id":"another-test-attachment",
            "mac":"52:da:f3:2a:76:de"
      }
]
```

8.5.1 Response headers

The following response headers expected as shown in Table 20

Table 20 Response headers – Read multiple network attachment objects

Header	Type	Requirement
Content-Type	application/json	If present a JSON array of zero one, or more JSON attachment objects.
Content-Type	text/plain	If present an implementation specific error message.

8.6 Read a network attachment object

Allows a user client to select amongst and and retrieve network attachment instances by its identifier.

Example: Perform a GET to the specified URI.

```
GET /networks/net-1/attachments/another-test-attachment HTTP/1.1
Accept: */*
```

The following shows the server's response.

```
HTTP/1.1 200 OK
Content-Type: application/json
{
      "id":"another-test-attachment",
      "mac":"52:da:f3:2a:76:de"
```

```
}
```

8.6.1 Response status

The HTTP status codes that occur when reading a network attachment object shown in Table 21

Table 21 HTTP Status codes – Read a network attachment object

HTTP Status	Description
202 OK	A network attachment object is found.
404 Not Found	A network attachment object is not found.

8.7 Delete a network attachment object

The following HTTP DELETE deletes an existing network attachment object at the specified URI.

DELETE /networks/<net-id>/attachments/<attachment-id>

Where:

<net-id> the id which identifies a network object instance.

<attachment-id> is the id of the network attachment object to be deleted.

Example: Perform a DELETE to the network attachment object URI.

```
DELETE /networks/net-1/attachments/test-attachment HTTP/1.1
Accept: */*
```

The following shows the server's response:

```
HTTP/1.1 202 Accepted
Content-Length: 0
Content-Type: text/plain; charset=ISO-8859-1
```

8.7.1 Response status

The HTTP status codes that occur when deleting a network attachment object shown in Table 22.

Table 22 Response status – Delete a network attachment object

HTTP Status	**Description**
202 Accepted	A network attachment object successfully deleted.
404 Not Found	A network attachment object is not found.
500 Internal Server Error	A processing error while deleting a network attachment object.

An example as to how to construct a trema network:

curl -v http://127.0.0.1:8888/networks/ -X POST -d '{“id”:”sliceA”, “description”: “Layer 2-A domain”}'
Creates a sliceA domain.
curl -v http://127.0.0.1:8888/networks/sliceA/ports – X POST -d '{“id”:”slice#1”,

“datapath_id”:”0xabc”, “port”:”trema0-0”, “vid”:”1024”}'
vswitch(“slice#1”) → “trema0-0”
curl -v http://127.0.0.1:8888/networks/sliceA/ports -X POST -d '{“id”:”slice#1”,
“datapath_id”:”0xabc”, “port”:”trema1-0”, “vid”:”1024”}'
vswitch(“slice#1”) → “trema1-0”

curl -v http://127.0.0.1:8888/networks/ -X POST -d '{“id”:”sliceB”, “description”: “Layer 2-B domain”}'
Creates a slice B domain.
Curl -v http://127.0.0.1:8888/networks/sliceB/ports -X POST -d '{“id”:”slice#2”,
“datapath_id”:”0xdef”, “port”:”trema2-0”, “vid”:512}'
vswitch(“slice#2”) →”trema2-0”
curl -v http://127.0.0.1:8888/networks/sliceB/ports -X POST -d '{“id”:”slice#3”,
“datapath_id”:”0xdef”, “port”:”trema2-1”, “vid”:512}'
vswitch(“slice#3”) → “trema2-1”

curl -v http://127.0.0.1:8888/networks/sliceA/ports/trema0-0/attachments -X POST -d
'{“id”:”mac_0_0”, “mac”:”11:22:33:44:55:66”}
vswitch(“sliceable”) → “trema0-0” → “11:22:33:44:55:66”
curl -v http://127.0.0.1:8888/networks/sliceA/attachments -X POST -d
'{“id”:”vhost1”,”mac”:”a8:35:67:de:2d:47”}'
vhost(“vhost1”) {
mac “a8:35:67:de:2d:47”
}