Cisco Routed PON REST API

Developer Guide

CAUTION

Refer to the Install Guide before Installation

Warranty Notice: Device Attenuation Required

Do not connect OLT directly to ONUs without proper attenuation. PON transceivers will be **permanently damaged** unless connected with **minimum** 16dB attenuation (20dB recommended). **Damage from optical overload will void Ciena warranty.**

Combination of attenuator and splitters can provide required attenuation. Refer to the example:

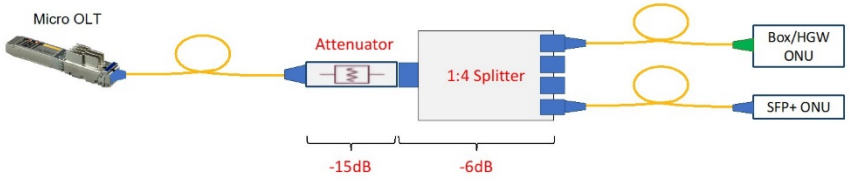


Table of Contents

References

ID	Document Description
Django	Django Documentation <https://docs.djangoproject.com/en/>.
Cisco Routed PON Installation Guide	Cisco Routed PON Installation Guide
Cisco Routed PON Manager User Guide	Cisco Routed PON Manager User Guide
Cisco Routed PON REST API	Cisco Routed PON REST API Developer Guide
Postman	Postman the API platform for building and using APIs, url: https://postman.com

Document Purpose

This document serves as the Developer Guide for the Cisco Routed PON RESTful Application Programming Interface (REST API) component. It describes the architecture and design of the API, structure of the REST endpoints, request and response formats, and reference and usage information for the REST API. This document is intended for developers building applications that leverage the REST API to manage the PON network and devices with the Tibit solution.

Although the open source MongoDB is shown as part of the Cisco Routed PON architecture, MongoDB is not provided as part of the Cisco Routed PON REST API package. MongoDB is a dependency of the REST API. Installation, maintenance, and operation of MongoDB is considered out of scope.

See References for a list of additional Cisco documentation regarding the Cisco Routed PON Manager.

Introduction

The Cisco Routed PON is the management solution for Cisco PON networks. The Cisco Routed PON management software stack is composed of the [PON Manager] Web Application graphical user interface and REST APIsoftware to ease the deployment of devices and subscriber services in the PON network. See [PON Manager] for a more complete description of the entire Cisco Routed PON solution.

The Cisco Routed PON REST API is a component of the PON Manager that provides an application programming interface over HTTPS for managing PON devices for the Cisco Routed PON solution. Customers can build device provisioning, service configuration, performance monitoring, log collection, and other applications on top of the API for managing the PON network. In addition to customer applications, the PON Manager Web App utilizes the API's PON and user management interfaces for its operation. The API implements a JSON interface that aligns directly with the PON Controller management data model and interfaces with the MongoDB datastore for accessing configuration, state, statistics, logging, and file collections.

The Cisco Routed PON REST API implements endpoints for managing the following:

- Device configuration and status for PON Controllers, OLTs, ONUs, and Switches.
- Service configuration, including ONU Service Configuration (SRV-CFG) files, VLANs, Service Level Agreements (SLAs), 802.1X Authentication, DHCP Relay, and PPPoE.
- Performance management statistics.
- Device alarms and logging.
- File management, including OLT firmware, ONU firmware, and device pictures.

Packaging and Installation

The Cisco Routed PON REST API is packaged and installed along with the Cisco Routed PON Manager software. See [PON Manager] for requirements, dependencies and instructions for installing and configuring the PON Manager Web Application and REST API.

API Overview

The Cisco Routed PON REST API is designed around best practices for a RESTful interface over a secure HTTPS transport, and provides API endpoints for provisioning and monitoring Cisco Routed PON OLT devices, as well as the subtended Cisco Routed PON ONU devices and third-party ONUs compliant with the XGS-PON and 10G-EPON standards. The API is aligned with the Cisco Routed PON Controller data model in MongoDB, where most API endpoints and data map directly to collections and documents in MongoDB. This is illustrated in Figure 1 below.

Figure 1 - Cisco Routed PON REST API MongoDB Collections and Documents

In order to build applications using the REST API effectively, developers need to understand the feature set and data model used to manage the PON network with the Tibit solution. Feature descriptions and the PON management data model outside the scope of this document. See [PON Controller] for a description of the MongoDB collections and document structures defined for managing the PON network.

Architecture

The Cisco Routed PON Manager software is composed of a graphical user front-end web application (Web App) and a RESTful interface (REST API) that provides access to the MongoDB datastore. These software components integrate with the Apache2 web server and Django REST framework as shown in Figure 2.

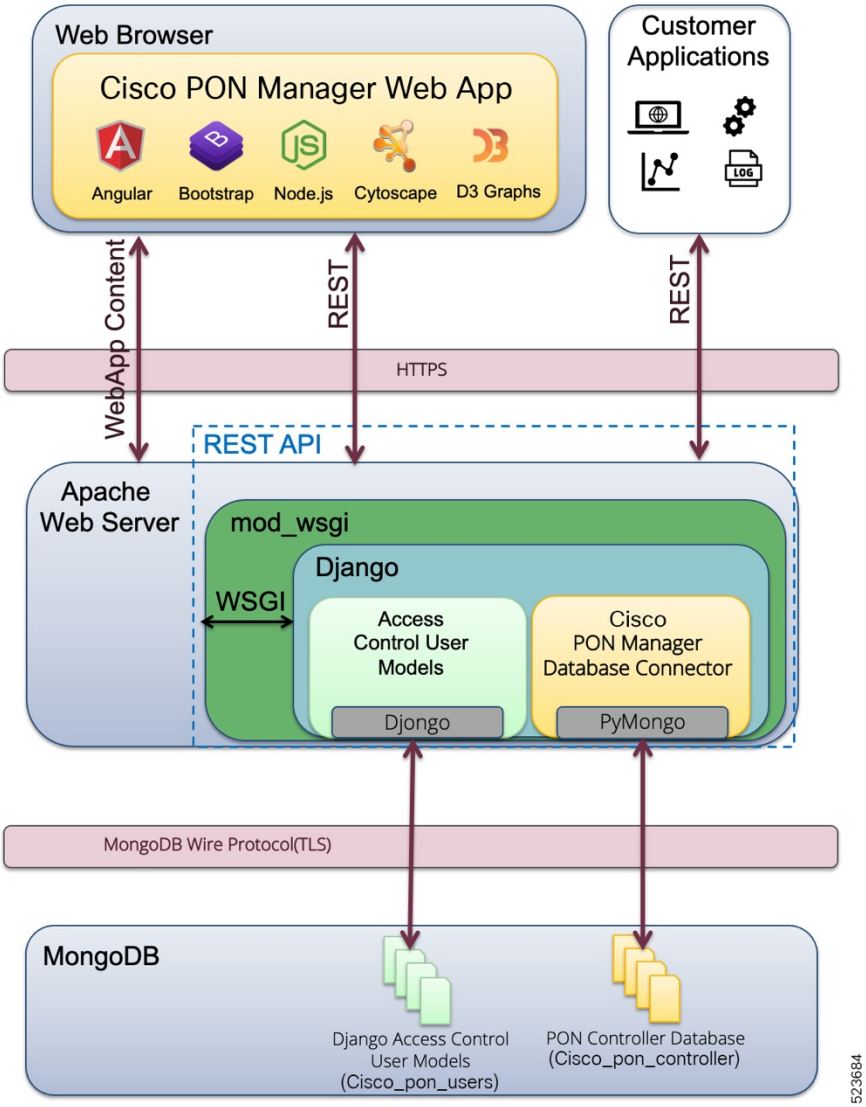


Figure 2 - Cisco Routed PON PON Manager Architecture

Cisco Routed PON Manager Web Application

The core of the Cisco Routed PON Manager Web App is built on the Angular framework and Bootstrap web front-end toolkit, along with libraries that support specific elements of the user interface. The Cytoscape library provides network visualization utilities for graphical tools such as the Polyglot OMCI Editor. Device statistics charts and graphs are built using the D3 Graphs package and ngx-charts library. The Font Awesome library provides icons and fonts for the user interface. See [PON Manager] for information on using the Web Application.

Web Server

The Cisco Routed PON Manager Web Server is built on the ubiquitous Apache HTTP server, which serves the Web Application graphical user interface, as well as the REST API for the PON Manager. The mod_wsgi module provides an interface between the Apache server and Django REST framework through the Web Server Gateway Interface (WSGI) specification for Python. PON Manager supports use of HTTPS for the REST API only. Non-secure HTTP is not supported.

Django REST Framework

The Cisco Routed PON REST API is built on the open source Django REST framework and toolkit for Python. By default, Django does not support integration with non-relational databases. The djongo library is used in parallel with a custom module to communicate with MongoDB. As HTTP requests are received, they are handled by Django accordingly, but are mapped to the custom module for database operations instead of Django's default ORM (object relational model). API endpoints are implemented using the PyMongo library for interfacing with MongoDB.

The PON Manager utilizes the user, group, permissions, and session management features of the Django REST framework to provide secure access and authorization for the web interface. See the Security section for more information on the security features for PON Manager.

MongoDB

The Mongo database provides the datastore for the Cisco Routed PON, and is used to store PON device provisioning and monitoring information collected by the PON Controller. MongoDB is an open source, secure database (www.mongodb.com) which employs a NoSQL architecture. See section MongoDB Installation for information on installation and configuration for use in Cisco Routed PON management solutions.

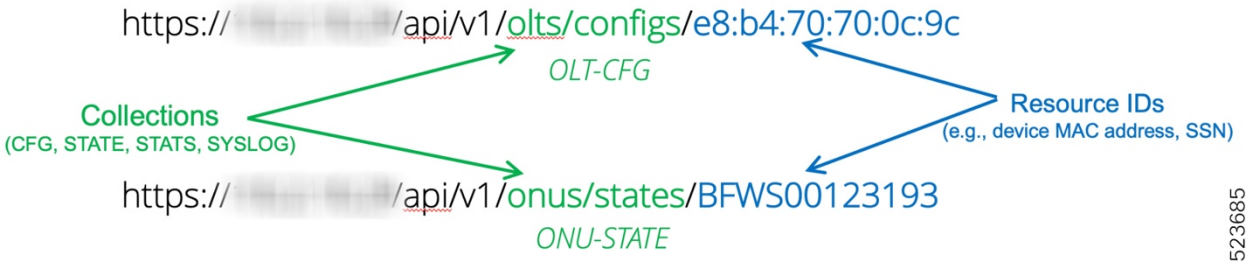
In addition to the PON device configuration and monitoring information, MongoDB is also used to store users, permissions, and session information used by the Django REST framework. The PON Controller defines the data model used to manage the PON network.

Endpoint Design

The Cisco Routed PON REST API URL format follows best practices for a "nouns, not verbs" REST API design, with endpoints designed around resources. An endpoint refers to a URL available for use within the REST API and the HTTP method associated with the request. A resource is defined by the data collection and ID, where specific resources are identified by ID in the URL after the collection mapping.

The API is designed to map to the [PON Controller] database model as closely as possible. Most URL endpoints map directly to a MongoDB collection. In the Cisco

Routed PON API, resources are PON Controllers, OLTs, ONUs Switches, SLAs, and files. Resource IDs are typically MAC Addresses, XGS-PON Serial Numbers, and file names. Examples of URLs identifying a specific OLT configuration and ONU state are shown in the figure below.



Message Formats

Cisco Routed PON REST API request and response messages support JSON formatted payloads only. POST and PUT requests use a common request body format with a “data” field containing the MongoDB document or other payload data required for the API. In addition to the HTTP response status code, API responses contain “status”, “data”, and “details” fields providing detailed response information.

Request Format

```
{
  “data”: {
    <MongoDB doc>
  }
}
```

Success Response Format

```
{
  “status”: “success”
   “data”: {...}
}
```

Error Response Format

```
{
  “status”: “fail”
  “details”: {...}
}
```

Private APIs

The REST API implements a number of ‘private’ endpoints that are intended to be used by the PON Manager Web App only. These ‘private’ APIs are not published in customer documentation and should not be used in customer applications. Private endpoints are not considered stable APIs and are subject to change from release to release.

API Programming Model

The REST API does not support PATCH or partial updating of documents. To update a document, you must upload the entire document to not lose data. The normal sequence of events is to retrieve a document, modify the document, and PUT/POST the document.

API Versioning

The Cisco Routed PON REST API maintains its own versioning scheme independent of the greater Cisco Routed PON release versions. The API version is identified at the beginning of the URL after the host information for every request. The format of the version string is “v1” where the “1” is replaced by the desired version number.

The API will only be versioned in the following cases:

- Any endpoint request message formats change
- Any endpoint response message formats change
- Required query parameters are added or removed

Each endpoint has the ability to be versioned independently of others. If a change is made to one endpoint, after v1 is released, it will now be accessible at v1 and v2. The v1 functionality will be unchanged from its prior implementation. If accessed with v1 in the URL the original, unchanged action will be performed. If accessed with v2, the new version will be used. All endpoints will be accessible via “v1” or “v2”, however the behavior and responses will be identical for all endpoints that were not stated as versioned.

Note that MongoDB documents contained in the payload of the API requests are versioned separately from the API version. See Document Versioning for more information.

Document Versioning

Each document in the database has a version that corresponds to the PON Controller version, which defines the format or schema version for a specific document. This version determines the format of the document and the set of fields in the document. Device configs use [CNTL][CFG Version] to determine their schema version. Note that documents are individually versioned Document version is written when a PON Controller creates new documents. Documents can be upgraded to the latest schema version using PON Manager's database upgrade feature.

MongoDB documents are versioned separately from the API. For example, in the PUT request shown below, the API versioning covers changes to the parts of the message highlighted in blue. This is independent of the format of the contents of the 'data' field in the API message, which is highlighted in red below. The format of the contents of the 'data' field is determined by the MongoDB document version as defined by the CNTL.CFG Version field.

```
PUT: /v2/onus/configs/<ONU ID>/
{
   "data": {
      "_id": "BFWS00123193",
      "CNTL": {
         "CFG Version": "R4.0.0"
      },
            . . .
   }
}
```

Schema Versioning

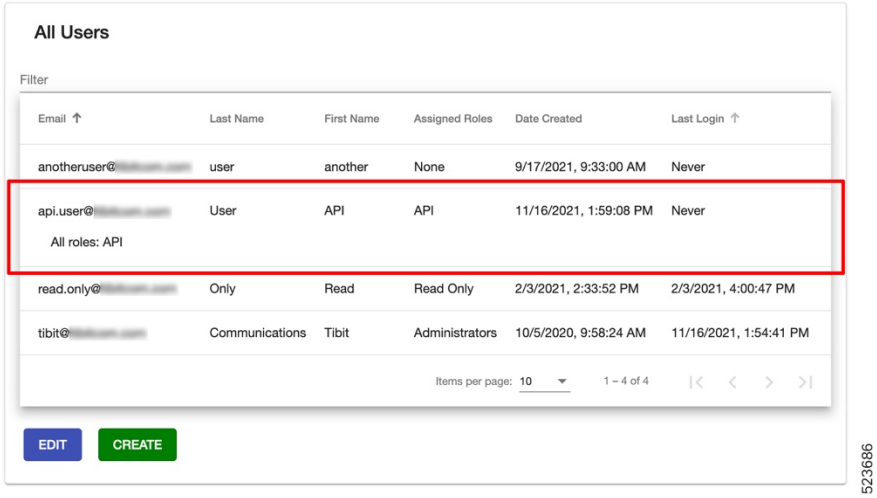
Schema for documents in the database can be found in the /opt/tibit/ponmgr/api/schema_files directory. The API schema files are versioned based on the release. The Schema version to use is based on the PON Controller configuration version of the given document.

The API applies the schema for validating input in API requests. For GET requests, the schema is used to validate query parameters, including the query filter and projection paths. The API returns a 400 error for an invalid query path. For PUT/POST requests, the schema is used to validate the format, data types, and ranges in documents provided with the requests. The API returns a warning if a required field is missing. The API returns a 400 error if a value is of the wrong type or doesn't fit the expected criteria defined by the schema.

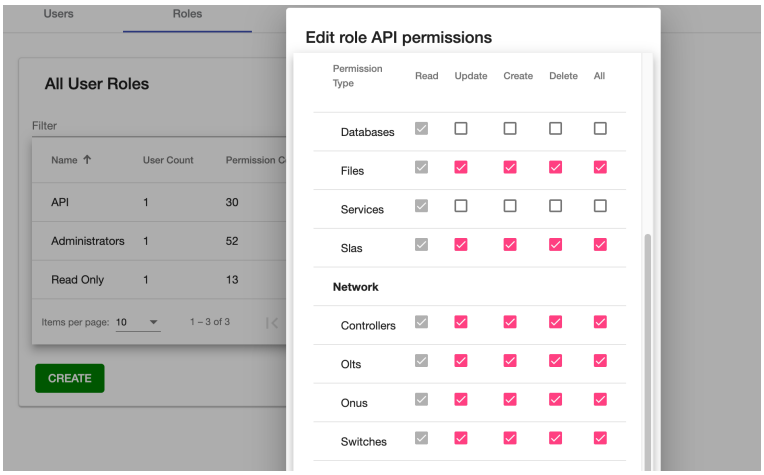
Security

The Cisco Routed PON REST API uses the same secure HTTPS transport, user management, session-based authentication, and role-based access controls utilized by

the PON Manager Web Application. See [PON Manager] for more information on the security features supported by PON Manager and how to configure and manage security for the REST API.



Users, roles, and permissions can be created and configured specifically to allow access through the REST interface. For example, one or more API-specific users could be configured that restrict access through the REST interface for managing ONUs and subscriber services, without providing access to more system administration functions such as user, file, and database management.



Documentation

In addition to this document, API documentation is provided with the PON Manager package. An OpenAPI schema is provided in both YAML and JSON formats, which describes request and response formats, as well as response status codes and query parameters for a given endpoint. Parsable JSON schema files are also provided for the PON management data structures defined by the PON Controller database. These

JSON schemas define the format for the request and response bodies transmitted in API HTTP messages.

File/Directory	Description
/opt/tibit/ponmgr/	Root directory of the PON Manager application.
doc/R2.3.0/	Directory containing the JSON schema definition files for various document types defined for the Cisco Routed PON Controller database.
doc/R3.0.0/	Directory containing the JSON schema definition files for various document types defined for the Cisco Routed PON Controller database.
doc/openapi.json	OpenAPI schema file describing the PON Manager REST API in a JSON file format.
doc/openapi.yaml	OpenAPI schema file describing the PON Manager REST API in a YAML file format.

Examples

API usage examples for are provided with the PON Manager package. The examples include python scripts, instructions on how to access the API through cURL commands (libcurl), and a Postman environment and collection. The python scripts show how to provision subscriber services and gather monitoring information for OLTs and ONUs on the PON network. A markdown file provides examples of cURL commands for use cases similar to the Python examples, such as provisioning a service on an ONU. Finally, Postman files can be imported into the Postman tool to be used in its graphical interface to visualize requests and responses. All examples may be found within the “examples” directory in the PON Manager installation directory, each in their own designated sub-directory.

File/Directory	Description
/opt/tibit/ponmgr/	Root directory of the PON Manager application.
examples/python/	Directory containing Python example code for configuring and provisioning OLT and ONU devices and subscriber services.
examples/curl/	Directory containing a markdown file with cURL examples for accessing the REST API.

examples/postman/	Directory containing a Postman application environment and collection files. Import these files into Postman to access the REST API through the tool (https://www.postman.com/).

API Reference

This section provides a reference for the API, including a listing and descriptions of API endpoints, message formats, applicable HTTP methods and response codes, and query parameters supported by the API.

Request Sequence

Applications utilizing the API follow the same general sequence described in this section. Before sending requests to retrieve or modify data, applications are required to login and establish a secure authenticated session. After login, all API requests must include the authentication Session ID and CSRF cookies as well as the CSRF Token. When the application is done with the session, the application must logout and terminate the session.

API sessions generally have the following sequence:

1. Log in to authenticate the user with the web server
   POST /v1/users/authenticate/

2. (Optional) Select the database to use for the session. If skipping this step, the user's default database (typically 'Default') is used for the session.
   PUT /v1/databases/selection/

3. Execute one or more request(s) to fetch data and/or configure the database. For example,
   GET /v1/onus/configs/BFWS00123193
   ...
   PUT /v1/onus/configs/BFWS00123193

4. Log out of the web server.
   GET /v1/users/logout/

Request Format

The REST API requires specific HTTP headers and request body format for all API requests it receives. After a successful login and establishing a secure authenticated session, all API requests must contain the following HTTP headers:

```
Cookie: <Session ID Cookie>
Cookie: <CSRF Token Cookie>
X-CSRFToken: <CSRF Token>
Referer: <API Host Address>
```

Additionally, all PUT and POST requests must also contain the content type header:

```
Content-Type: application/json
```

The Cookie and X-CSRFToken headers are used by the API to identify the user's session to verify they are logged in and have permissions to access the requested endpoint. The Referer header lets the API know what client is making the request (e.g. PON Manager web UI). Finally, the Content-Type header is required when the request contains a body. This tells the API that the content is JSON data and some other format. The cookies are returned after a successful login attempt. The CSRF Token cookie is the same value used for the X-CSRFToken header.

The REST API also expects specific formats of any bodies included in a request. PUT and POST requests follow a consistent request body format with an outer "data" tag containing the MongoDB document or other JSON payload as defined by the endpoint. The request body format is a follows:

```
{
        "data": {
                <MongoDB document>
        }
}
```

API request bodies are specific to each endpoint. Refer to the documentation below for details on the specific request body format for a specific endpoint.

An example of a request to select the active database would be as follows:

```
PUT @ /databases/selection/
Body: {
        "data": "DatabaseId"
}
```

Response Format

API responses have a consistent response body format. The body may contain the requested data, a status indicator, warning, or error messages depending on the result. All Cisco Routed PON REST API responses use the following formats.

Success

The API returns a 'success' response payload for requests that PON Manager is able to process and complete successfully. The response payload contains a 'status' field that reports if the request was accepted and processed or encountered an error. GET responses also include the "data" field containing the requested data. An example of a successful response when retrieving a resource is shown below:

```
GET @ /onus/states/<ONU ID>/
{
        "status": "success",
        "data": {
                <ONU-STATE MongoDB document>
```

```
        }
    }
```

A successful PUT or POST request may have the following body with an HTTP status code of 200 or 201:

```
    {
        "status": "success"
    }
```

Data Validation Warning

PUT or POST requests may return a “status” field containing “validation warning”. This means that the provided data in the request body does not match the defined schema. However, it is not a breaking difference and will not affect the Cisco Routed PON functionality, so the request is processed as a ‘success’. The API updates the database and returns an HTTP status code 200. Validation warning responses include an additional ‘details’ field, which describes the warning in detail including the offending attribute name, the value, the schema path to the attribute, and a description of why validation failed. In the example shown below, CNTL.CFG Version is a required field that is missing from the document provided in the PUT request:

```
PUT @ /olts/<OLT ID>/
{
   "status": "validation warning",
   "details": {
      "level": "warning",
      "message": "JSON validation failed: 'CFG Version' is a
                 required property",
      "collection": "OLT-CFG",
      "id": "<OLT ID>",
      "path": "['CNTL']",
      "bad value": {},
      "bad value type": "object",
      "validator": "required",
      "schema": {
         "type": "object",
         "properties": {
            "CFG Version": {
               "type": "string",
               "description": "Capability of configuration
                               file format."
            }
         },
         "required": [
            "CFG Version"
         ]
```

```
        }
    }
}
```

Data Validation Error

A validation error is similar to the validation warning described above. PUT or POST requests may return a “status” field containing “fail” with a JSON validation failure message. This error indicates that data validation failed due to the data provided in the request body not matching the schema defined for this document. Moreover, the request was NOT successful and no update was made to the database. The API returns this response with a 400 HTTP status code, indicating that the data provided is invalid and not usable. In the example below, the request was setting an integer value of ‘0’ for the ‘auth_db’ fields that requires a string value:

```
PUT @ /databases/Default/
{
    "status": "fail",
    "details": {
        "level": "error",
        "message": "JSON validation failed: 0 is not of type 'string'",
        "collection": "DATABASES",
        "id": null,
        "path": "['auth_db']",
        "bad value": 0,
        "bad value type": "integer",
        "validator": "type",
        "schema": {
            "type": "string",
            "pattern": "^([^\/\\. ';$*<>:|?]+)$",
            "maxlength": 63
        }
    }
}
```

Server Error

In general, server errors returned with 400 or 500 level status codes include a “status” field with a value of “fail” in the response body. The Data Validation Error described above is a subset of the possible server error responses from the API. There are many ways an error response may occur, such as malformed request body, missing or invalid query parameters, missing permissions, or an internal server error. An example below shows the 400 status code response body for a missing query parameter that is required:

```
GET @ /olts/stats/<OLT ID>/
```

```
{
    "status": "fail",
    "details": {
        "message": "Parameter 'start-time' is required"
    }
}
```


HTTP Methods

There are four basic HTTP operations available for use in the Cisco Routed PON REST API. Not all HTTP methods are available on all endpoints. Each method maps to a CRUD operation. The equivalent MongoDB operations are also listed below. See the MongoDB documentation for more details on these operations.

HTTP Method	CRUD	MongoDB	Description
GET	Read	find, findOne	Used to retrieve data such as device configuration, state, statistics, and logs.
POST	Create	insertOne	Create a new resource on the server or database.
PUT	Update	replaceOne	Replace an existing or create a new resource configuration. This operation completely replaces the existing document with the new document from the request body.
PATCH (future)	Update	updateOne	Updates a portion of an existing document resource. **Note:** This is not yet supported in the current release.
DELETE	Delete	deleteOne	Deletes the configuration or state for the specified resource.

HTTP Status Codes

Listed below are the most common HTTP response codes that are returned by the Cisco Routed PON REST API and their meaning. Other status codes may also occur, but do not appear often.

Status	Description
200 - OK	The request was completed successfully with no issues
201 - Created	The request was completed successfully and a new resource was created
204 - No Content	The request was completed successfully, but no data is returned
400 - Bad Request	Indicates an error in the request format or request data. Often caused by a request data validation error.
401 - Unauthorized	Indicates the user could not be authenticated. Will be returned after a failed login attempt.
403 - Forbidden	Indicates the request could not be completed because the user is either not authenticated or does not have the required permissions.
404 - Not Found	Indicates that the requested resource was not found. May be returned if the URL does not exist or a file or database entry does not exist.
409 - Conflict	Indicates that the requested resource could not be created because its ID is already in use. May occur when uploading files or creating databases.
500 - Server Error	A generic server error response that contains information about the failure in the response body.

URL Parameters

There are various URL query parameters available within the Cisco Routed PON REST API. Listed below are all possible query parameters. However, every endpoint supports its own subset of these options, if any. Query parameters available for each endpoint can be found in the description of the endpoint. **Note:** some tools, such as cURL, require any space characters in the query parameters to be escaped as “%20”.

Name	Description	Example
start-time	Only consider documents newer than the UTC timestamp	2021-01-01 00:00:00
end-time	Only consider documents older than the UTC timestamp	2021-01-01 00:00:00
query	Performs the custom query on the MongoDB database. Takes a comma separated list of “<key>=<value>” strings.	CNTL.Version=R2.3.0
projection	Limits the database attributes returned in the response. Use 1 to include, 0 to exclude. Takes a comma separated list of “<key>=0|1” strings.	CNTL.Version=1
sort	Sorts the response data on the given attribute in ascending or descending order	_id=1
limit	Limits the number of items to be included in the response data. **Note:** All endpoints that use the limit parameter use a default value of 1,000	50
skip	Number of items to be excluded from the beginning of the results data list	10
next	A document _id that will exclude all items from the results list that have this ID or come before it	70:b3:d5:52:34:1e
distinct	A database attribute to retrieve all unique values for from the endpoint’s collection	CNTL.Version

PON Controllers

States

The /controllers/states/ resource and endpoints are used to monitor PON Controller devices. A controller resource is identified by the device MAC Address and maps to a

CNTL-STATE document in MongoDB. See [PON Controller] for more details on the content of the CNTL-STATE document. See the CNTL-STATE.json schema file for a description of the parameters, data types, and ranges.

HTTP Method	Endpoint
GET	**GET /controllers/states/** Get a list of CNTL-STATE documents. URL Parameters: query (optional), projection (optional), sort (optional), limit (optional), skip (optional), next (optional), distinct (optional)
GET	**GET /controllers/states/<CNTL ID>/** Get a CNTL-STATE document identified by <CNTL ID> MAC address.
DELETE	**DELETE /controllers/states/<CNTL ID>/** Delete a CNTL-STATE document identified by <CNTL ID> MAC address.

Configurations

The /controllers/configs/ resource and endpoints are used to configure PON Controller devices. A controller resource is identified by the device MAC Address and maps to a CNTL-CFG document in MongoDB. See [PON Controller] for more details on the content of the CNTL-CFG document. See the CNTL-CFG.json schema file for a description of the parameters, data types, and ranges.

HTTP Method	Endpoint
GET	**GET /controllers/configs/** Get a list of CNTL-CFG documents. URL Parameters: query (optional), projection (optional), sort (optional), limit (optional), skip (optional), next (optional), distinct (optional)
POST	**POST /controllers/configs/** Create a CNTL-CFG document.
GET	**GET /controllers/configs/<CNTL ID>/** Get a CNTL-CFG document identified by <CNTL ID> MAC address.
PUT	**PUT /controllers/configs/<CNTL ID>/** Create or replace a CNTL-CFG document identified by <CNTL ID> MAC address.
DELETE	**DELETE /controllers/configs/<CNTL ID>/** Delete a CNTL-CFG document identified by <CNTL ID> MAC address.

Authentication State

The /controllers/auth/states/ resource and endpoints are used to monitor the Cisco Routed PON Authenticator, which is packaged with the PON Controller. An authenticator state resource is identified by the device MAC Address and maps to a CNTL-AUTH-STATE document in MongoDB. See [PON Controller] for more details on the content of the CNTL-AUTH-STATE document. See the CNTL-AUTH-STATE.json schema file for a description of the parameters, data types, and ranges.

HTTP Method	Endpoint
GET	**GET /controllers/auth/states/** Get a list of CNTL-AUTH-STATE documents.
GET	**GET /controllers/auth/states/<CNTL ID>/** Get a CNTL-AUTH-STATE document identified by <CNTL ID>.
DELETE	**DELETE /controllers/auth/states/<CNTL ID>/** Delete a CNTL-AUTH-STATE document identified by <CNTL ID>.

Engine State

The /controllers/engine/states/ resource and endpoints are used to monitor the Cisco Routed PON UMT Relay process, which is packaged with the PON Controller. An engine state resource is identified by the device MAC Address and maps to a CNTL-ENGINE-STATE document in MongoDB. See [PON Controller] for more details on the content of the CNTL-ENGINE-STATE document. See the CNTL-ENGINE-STATE.json schema file for a description of the parameters, data types, and ranges.

HTTP Method	Endpoint
GET	**GET /controllers/engine/states/** Get a list of CNTL-ENGINE-STATE documents.
GET	**GET /controllers/engine/states/<CNTL ID>/** Get a CNTL-ENGINE-STATE document identified by <CNTL ID>.
DELETE	**DELETE /controllers/engine/states/<CNTL ID>/** Delete a CNTL-ENGINE-STATE document identified by <CNTL ID>.

Alarm Configurations

The /controllers/alarm-configs/ resource and endpoints are used to configure PON Controller alarm profiles. An alarm configuration resource is identified by the configuration ID and maps to a CNTL-ALARM-CFG document in MongoDB. See [PON Controller] for more details on the content of the CNTL-ALARM-CFG document. See the CNTL-ALARM-CFG.json schema file for a description of the parameters, data types, and ranges.

HTTP Method	Endpoint
GET	**GET /controllers/alarm-configs/** Get a list of CNTL-ALARM-CFG documents. URL Parameters: query (optional), projection (optional), sort (optional), limit (optional), skip (optional), next (optional), distinct (optional)
POST	**POST /controllers/alarm-configs/** Create a CNTL-ALARM-CFG document.
GET	**GET /controllers/alarm-configs/<CFG ID>/** Get a CNTL-ALARM-CFG document identified by <CFG ID>.
PUT	**PUT /controllers/alarm-configs/<CFG ID>/** Create or replace a CNTL-ALARM-CFG document identified by <CFG ID>.
DELETE	**DELETE /controllers/alarm-configs/<CFG ID>/** Delete a CNTL-ALARM-CFG document identified by <CFG ID>.

Statistics

The /controllers/stats/ resource and endpoints are used to monitor statistics for PON Controllers. A PON Controller resource is identified by the device MAC Address and maps to the STATS-CNTL-<CNTL ID> collection in versions R3.0.0 and earlier and STATS-CNTL in versions R3.1.0 and later in MongoDB. See [PON Controller] for more details on the content of the STATS-CNTL collection.

HTTP Method	Endpoint
GET	**GET /controllers/stats/<CNTL ID>/** Get a list of STATS-CNTL documents. URL Parameters: start-time (required), end-time (optional)
DELETE	**DELETE /controllers/stats/<CNTL ID>/** Delete the STATS-CNTL collection.

Logs

The /controllers/logs/ resource and endpoints are used to monitor logs for PON Controllers. A PON Controller resource is identified by the device MAC Address and maps to the SYSLOG-CNTL-<CNTL ID> collection in versions R3.0.0 and earlier and SYSLOG-CNTL in versions R3.1.0 and later in MongoDB. See [PON Controller] for more details on the content of the SYSLOG-CNTL collection.

HTTP Method	Endpoint
GET	**GET /controllers/logs/<CNTL ID>/** Get a list of SYSLOG-CNTL documents. URL Parameters: start-time (required), end-time (optional)
DELETE	**DELETE /controllers/logs/<CNTL ID>/** Delete the SYSLOG-CNTL collection.

Switches

Configurations

The /switches/configs/ resource and endpoints endpoints are used to configure Switch devices. A switch resource is identified by the device MAC Address and maps to a SWI-CFG document in MongoDB. See [PON Controller] for more details on the content of the SWI-CFG document. See the SWI-CFG.json schema file for a description of the parameters, data types, and ranges.

HTTP Method	Endpoint
GET	**GET /switches/configs/** Get a list of SWI-CFG documents. URL Parameters: query (optional), projection (optional), sort (optional), limit (optional), skip (optional), next (optional), distinct (optional)
POST	**POST /switches/configs/** Create a SWI-CFG document.
GET	**GET /switches/configs/<SWI ID>/** Get a SWI-CFG document identified by <SWI ID> MAC address.
PUT	**PUT /switches/configs/<SWI ID>/** Create or replace a SWI-CFG document identified by <SWI ID> MAC address.
DELETE	**DELETE /switches/configs/<SWI ID>/** Delete a SWI-CFG document identified by <SWI ID> MAC address.

OLTs

States

The /olts/states/ resource and endpoints are used to monitor OLT devices. An OLT resource is identified by the device MAC Address and maps to an OLT-STATE document in MongoDB. See [PON Controller] for more details on the content of the OLT-STATE document. See the OLT-STATE.json schema file for a description of the parameters, data types, and ranges.

HTTP Method	Endpoint
GET	**GET /olts/states/** Get a list of OLT-STATE documents. URL Parameters: query (optional), projection (optional), sort (optional), limit (optional), skip (optional), next (optional), distinct (optional)
GET	**GET /olts/states/<OLT ID>/** Get an OLT-STATE document identified by <OLT ID> MAC address.
DELETE	**DELETE /olts/states/<OLT ID>/** Delete an OLT-STATE document identified by <OLT ID> MAC address.

Configurations

The /olts/configs/ resource and endpoints are used to monitor OLT devices. An OLT resource is identified by the device MAC Address and maps to an OLT-CFG document in MongoDB. See [PON Controller] for more details on the content of the OLT-CFG document. See the OLT-CFG.json schema file for a description of the parameters, data types, and ranges.

HTTP Method	Endpoint
GET	**GET /olts/configs/** Get a list of OLT-CFG documents. URL Parameters: query (optional), projection (optional), sort (optional), limit (optional), skip (optional), next (optional), distinct (optional)
POST	**POST /olts/configs/** Create an OLT-CFG document.
GET	**GET /olts/configs/<OLT ID>/** Get an OLT-CFG document identified by <OLT ID> MAC address.
PUT	**PUT /olts/configs/<OLT ID>/** Create or replace an OLT-CFG document identified by <OLT ID> MAC address.
DELETE	**DELETE /olts/configs/<OLT ID>/** Delete an OLT-CFG document identified by <OLT ID> MAC address.

Alarm Configurations

The /olts/alarm-configs/ resource and endpoints are used to configure OLT alarm profiles. An alarm configuration resource is identified by the configuration ID and maps to a OLT-ALARM-CFG document in MongoDB. See [PON Controller] for more details on the content of the OLT-ALARM-CFG document. See the OLT-ALARM-CFG.json schema file for a description of the parameters, data types, and ranges.

HTTP Method	Endpoint
GET	**GET /olts/alarm-configs/** Get a list of OLT-ALARM-CFG documents. URL Parameters: query (optional), projection (optional), sort (optional), limit (optional), skip (optional), next (optional), distinct (optional)
POST	**POST /olts/alarm-configs/** Create an OLT-ALARM-CFG document.
GET	**GET /olts/alarm-configs/<CFG ID>/** Get an OLT-ALARM-CFG document identified by <CFG ID>.
PUT	**PUT /olts/alarm-configs/<CFG ID>/** Create or replace an OLT-ALARM-CFG document identified by <CFG ID>.
DELETE	**DELETE /olts/alarm-configs/<CFG ID>/** Delete an OLT-ALARM-CFG document identified by <CFG ID>.

Statistics

The /olts/stats/ resource and endpoints are used to monitor statistics for OLTs. An OLT resource is identified by the device MAC Address and maps to the STATS-OLT-<OLT ID> collection in versions R3.0.0 and earlier and STATS-OLT in versions R3.1.0 and later in MongoDB. See [PON Controller] for more details on the content of the STATS-OLT collection.

HTTP Method	Endpoint
GET	**GET /olts/stats/<OLT ID>/** Get a list of STATS-OLT documents. URL Parameters: start-time (required), end-time (optional)
DELETE	**DELETE /olts/stats/<OLT ID>/** Delete the STATS-OLT collection.

Logs

The /olts/logs/ resource and endpoints are used to monitor logs for OLTs. An OLT resource is identified by the device MAC Address and maps to the SYSLOG-OLT-<OLT ID> collection in versions R3.0.0 and earlier and SYSLOG-OLT in versions R3.1.0 and later in MongoDB. See [PON Controller] for more details on the content of the SYSLOG-OLT collection.

HTTP Method	Endpoint
GET	**GET /olts/logs/<OLT ID>/** Get a list of SYSLOG-OLT documents. URL Parameters: start-time (required), end-time (optional)
DELETE	**DELETE /olts/logs/<OLT ID>/** Delete the SYSLOG-OLT collection.

Debug

The /olts/debug/ resource and endpoints are used to monitor debug information for OLTs. An OLT resource is identified by the device MAC Address and maps to the DEBUG-OLT collection in versions R3.2.0 and later in MongoDB. See [PON Controller] for more details on the content of the DEBUG-OLT collection.

HTTP Method	Endpoint
GET	**GET /olts/debug/<OLT ID>/** Get a list of DEBUG-OLT documents. URL Parameters: start-time (optional), end-time (optional), limit (optional), skip (optional)
DELETE	**DELETE /olts/debug/<OLT ID>/** Delete the documents for the given OLT from the DEBUG-OLT collection.

Actions

The /olts/<OLT ID>/ endpoints provide a way to execute common actions on an OLT device. An OLT resource is identified by the device MAC Address and maps to the OLT-CFG collection in MongoDB. See [PON Controller] for more details on the content of the OLT-CFG collection.

All OLT action GET requests return a document of the following data:
{ "Pending": False|True }

HTTP Method	Endpoint
GET	**GET /olts/<OLT ID>/disable-onu/<ONU ID>/** Gets the pending status of the Disable ONU action identified by <OLT ID> and <ONU ID>.
PUT	**PUT /olts/<OLT ID>/disable-onu/<ONU ID>/** Send the Disable ONU signal for the ONU identified by <ONU ID> managed by the OLT <OLT ID>. **Request Data:** { "disable": True|False }
GET	**GET /olts/<OLT ID>/broadcast-enable-onus/** Gets the pending status of the Broadcast Enable ONUs action for the OLT identified by <OLT ID>
PUT	**PUT /olts/<OLT ID>/broadcast-enable-onus/** Send the Broadcast Enable ONU signal for the OLT identified by <OLT ID>.
GET	**GET /olts/<OLT ID>/allow-registration/** Gets the pending status of the Allow ONU Registration action for the OLT identified by <OLT ID>.
PUT	**PUT /olts/<OLT ID>/allow-registration/** Send the Allow ONU Registration signal for the OLT identified by <OLT ID>. **Request Data:** { "onu-id": <ONU MAC Address/SSN>|"ALL" }
GET	**GET /olts/<OLT ID>/reset/** Gets the pending status of the Reset action for the OLT identified by <OLT ID>.
PUT	**PUT /olts/<OLT ID>/reset/** Send the Reset signal for the OLT identified by <OLT ID>.

ONUs

States

The /onus/states/ resource and endpoints are used to monitor ONU devices. An ONU resource is identified by the device Serial Number (when using XGS-PON) or MAC Address (when using 10G EPON) and maps to an ONU-STATE document in MongoDB. See [PON Controller] for more details on the content of the ONU-STATE document. See the ONU-STATE.json schema file for a description of the parameters, data types, and ranges.

HTTP Method	Endpoint
GET	**GET /onus/states/** Get a list of ONU-STATE documents. URL Parameters: query (optional), projection (optional), sort (optional), limit (optional), skip (optional), next (optional), distinct (optional)
GET	**GET /onus/states/<ONU ID>/** Get an ONU-STATE document identified by <ONU ID> MAC address.
DELETE	**DELETE /onus/states/<ONU ID>/** Delete an ONU-STATE document identified by <ONU ID> MAC address.

Configurations

The /onus/configs/ resource and endpoints are used to monitor ONU devices. An ONU resource is identified by the device Serial Number (when using XGS-PON) or MAC Address (when using 10G EPON) and maps to an ONU-CFG document in MongoDB. See [PON Controller] for more details on the content of the ONU-CFG document. See the ONU-CFG.json schema file for a description of the parameters, data types, and ranges.

HTTP Method	Endpoint
GET	**GET /onus/configs/** Get a list of ONU-CFG documents. URL Parameters: query (optional), projection (optional), sort (optional), limit (optional), skip (optional), next (optional), distinct (optional)
POST	**POST /onus/configs/** Create an ONU-CFG document.
GET	**GET /onus/configs/<ONU ID>/** Get an ONU-CFG document identified by <ONU ID> MAC address/Serial Number.
PUT	**PUT /onus/configs/<ONU ID>/** Create or replace an ONU-CFG document identified by <ONU ID> MAC address/Serial Number.
DELETE	**DELETE /onus/configs/<ONU ID>/** Delete an ONU-CFG document identified by <ONU ID> MAC address/Serial Number.

Alarm Configurations

The /onus/alarm-configs/ resource and endpoints are used to configure ONU alarm profiles. An alarm configuration resource is identified by the configuration ID and maps to a ONU-ALARM-CFG document in MongoDB. See [PON Controller] for more details on the content of the ONU-ALARM-CFG document. See the ONU-ALARM-CFG.json schema file for a description of the parameters, data types, and ranges.

HTTP Method	Endpoint
GET	**GET /onus/alarm-configs/** Get a list of ONU-ALARM-CFG documents. URL Parameters: query (optional), projection (optional), sort (optional), limit (optional), skip (optional), next (optional), distinct (optional)
POST	**POST /onus/alarm-configs/** Create an ONU-ALARM-CFG document.
GET	**GET /onus/alarm-configs/<CFG ID>/** Get an ONU-ALARM-CFG document identified by <CFG ID>.
PUT	**PUT /onus/alarm-configs/<CFG ID>/** Create or replace an ONU-ALARM-CFG document identified by <CFG ID>.
DELETE	**DELETE /onus/alarm-configs/<CFG ID>/** Delete an ONU-ALARM-CFG document identified by <CFG ID>.

Statistics

The /onus/stats/ resource and endpoints are used to monitor statistics for ONUs. An ONU resource is identified by the device Serial Number (when using XGS-PON) or MAC Address (when using 10G EPON) and maps to the STATS-ONU-<ONU ID> collection in versions R3.0.0 and earlier and STATS-ONU in versions R3.1.0 and later in MongoDB. See [PON Controller] for more details on the content of the STATS-ONU collection.

HTTP Method	Endpoint
GET	**GET /onus/stats/<ONU ID>/** Get a list of STATS-ONU documents. URL Parameters: start-time (required), end-time (optional)
DELETE	**DELETE /onus/stats/<ONU ID>/** Delete the STATS-ONU collection.

Logs

The /onus/logs/ resource and endpoints are used to monitor logs for ONUs. An ONU resource is identified by the device Serial Number (when using XGS-PON) or MAC Address (when using 10G EPON) and maps to the SYSLOG-ONU-<ONU ID> collection in versions R3.0.0 and earlier and SYSLOG-ONU in versions R3.1.0 and later in MongoDB. See [PON Controller] for more details on the content of the SYSLOG-ONU collection.

HTTP Method	Endpoint
GET	**GET /onus/logs/<ONU ID>/** Get a list of SYSLOG-ONU documents. URL Parameters: start-time (required), end-time (optional)
DELETE	**DELETE /onus/logs/<ONU ID>/** Delete the SYSLOG-ONU collection.

CPEs

The /onus/cpe-states/ resource and endpoints are used to monitor CPEs for ONUs. An ONU resource is identified by the device Serial Number (when using XGS-PON) or MAC Address (when using 10G EPON) and maps to the ONU-CPE-STATE collection in MongoDB. See [PON Controller] for more details on the content of the ONU-CPE-STATE collection. See the ONU-CPE-STATE.json schema file for a description of the parameters, data types, and ranges.

HTTP Method	Endpoint
GET	**GET /onus/cpe-states/** Get a list of ONU-CPE-STATE documents. URL Parameters: query (optional), projection (optional), sort (optional), limit (optional), skip (optional), next (optional), distinct (optional)
GET	**GET /onus/cpe-states/<ONU ID>/** Get an ONU-CPE-STATE document identified by <ONU ID> MAC address/Serial Number.
DELETE	**DELETE /onus/cpe-states/<ONU ID>/** Delete the ONU-CPE-STATE document identified by <ONU ID> MAC address/Serial Number.

MIB Reset States

The /onus/mib-rst/ resource and endpoints are used to monitor MIB Reset States for ONUs. An ONU resource is identified by the device Serial Number (when using XGS-PON) or MAC Address (when using 10G EPON) and maps to the ONU-MIB-RST-STATE collection in MongoDB. See [PON Controller] for more details on the content of the ONU-MIB-RST-STATE collection. See the ONU-MIB-RST-STATE.json schema file for a description of the parameters, data types, and ranges.

HTTP Method	Endpoint
GET	**GET /onus/mib-rst/** Get a list of ONU-MIB-RST-STATE documents. URL Parameters: query (optional), projection (optional), sort (optional), limit (optional), skip (optional), next (optional), distinct (optional)
GET	**GET /onus/mib-rst/<ONU ID>/** Get an ONU-MIB-RST-STATE document identified by <ONU ID> MAC address/Serial Number.
DELETE	**DELETE /onus/mib-rst/<ONU ID>/** Delete the ONU-MIB-RST-STATE document identified by <ONU ID> MAC address/Serial Number.

MIB Current States

The /onus/mib-cur/ resource and endpoints are used to monitor MIB Current States for ONUs. An ONU resource is identified by the device Serial Number (when using XGS-PON) or MAC Address (when using 10G EPON) and maps to the ONU-MIB-CUR-STATE collection in MongoDB. See [PON Controller] for more details on the content of the ONU-MIB-CUR-STATE collection. See the ONU-MIB-CUR-STATE.json schema file for a description of the parameters, data types, and ranges.

HTTP Method	Endpoint
GET	**GET /onus/mib-cur/** Get a list of ONU-MIB-CUR-STATEdocuments. URL Parameters: query (optional), projection (optional), sort (optional), limit (optional), skip (optional), next (optional), distinct (optional)
GET	**GET /onus/mib-cur/<ONU ID>/** Get an ONU-MIB-CUR-STATEdocument identified by <ONU ID> MAC address/Serial Number.
DELETE	**DELETE /onus/mib-cur/<ONU ID>/** Delete the ONU-MIB-CUR-STATE document identified by <ONU ID> MAC address/Serial Number.

Actions

The /onus/<ONU ID>/ endpoints provide a way to execute common actions on an ONU device. An ONU resource is identified by the device Serial Number (when using XGS-PON) or MAC Address (when using 10G EPON) and maps to the ONU-CFG collection in MongoDB. See [PON Controller] for more details on the content of the ONU-CFG collection.

All ONU action GET requests return a document of the following data:
{ "Pending": False|True }

HTTP Method	Endpoint
GET	**GET /onus/<ONU ID>/reset/** Gets the pending status of the Reset action for the ONU identified by <ONU ID>.
PUT	**PUT /onus/<ONU ID>/reset/** Send the Reset signal for the ONU identified by <ONU ID>.

Files

OLT Firmware

The /files/olt-firmware/ resource and endpoints are used to manage OLT Firmware. An OLT Firmware resource is identified by the filename and maps to the OLT-FIRMWARE bucket in MongoDB. See [PON Controller] for more details on the content of the OLT-FIRMWARE bucket.

HTTP Method	Endpoint
GET	**GET /files/olt-firmware/** Get a list of OLT Firmware.
GET	**GET /files/olt-firmware/<FILENAME>/** Get an OLT Firmware's metadata identified by <FILENAME>.
PUT	**PUT /files/olt-firmware/<FILENAME>/** Replace an OLT Firmware's metadata identified by <FILENAME>. See [OLT Firmware PUT Request Data] for the request body example.
POST	**POST /files/olt-firmware/<FILENAME>/** Upload an OLT Firmware identified by <FILENAME>. See [OLT Firmware POST Request Data] for the request body example.
DELETE	**DELETE /files/olt-firmware/<FILENAME>/** Delete an OLT Firmware identified by <FILENAME>.

OLT Firmware PUT Request Data

```
{ "metadata": { "Compatible Manufacturer": "TIBITCOM",
                "Compatible Model": [ "180713" ],
                "Version": "R3.2.0" } }
```

OLT Firmware POST Request Data

```
{ "file": <base64 file data>, "metadata": {
        "Compatible Manufacturer": "TIBITCOM",
        "Compatible Model": [ "180713" ],
        "Version": "R3.2.0" } }
```

ONU Firmware

The /files/onu-firmware/ resource and endpoints are used to manage ONU Firmware. An ONU Firmware resource is identified by the filename and maps to the ONU-FIRMWARE bucket in MongoDB. See [PON Controller] for more details on the content of the ONU-FIRMWARE bucket.

HTTP Method	Endpoint
GET	**GET /files/onu-firmware/** Get a list of ONU Firmware.
GET	**GET /files/onu-firmware/<FILENAME>/** Get an ONU Firmware's metadata identified by <FILENAME>.
PUT	**PUT /files/onu-firmware/<FILENAME>/** Replace an ONU Firmware's metadata identified by <FILENAME>. See [ONU Firmware PUT Request Data] for the request body example.
POST	**POST /files/onu-firmware/<FILENAME>/** Upload an ONU Firmware identified by <FILENAME>. See [ONU Firmware POST Request Data] for the request body example.
DELETE	**DELETE /files/onu-firmware/<FILENAME>/** Delete an ONU Firmware identified by <FILENAME>.

ONU Firmware PUT Request Data

```
{ "metadata": { "Compatible Manufacturer": "CISCO",
                "Compatible Model": [ "Cisco PON ONU" ],
                "Version": "R3.2.0" } }
```

ONU Firmware POST Request Data

```
{ "file": <base64 file data>, "metadata": {
        "Compatible Manufacturer": "CISCO",
        "Compatible Model": [ " Cisco PON ONU" ],
        "Version": "R3.2.0" } }
```

Pictures

The /files/pictures/ resource and endpoints are used to manage device pictures. A picture resource is identified by the filename and maps to the PICTURES bucket in MongoDB. See [PON Controller] for more details on the content of the PICTURES bucket.

HTTP Method	Endpoint
GET	**GET /files/pictures/** Get a list of device pictures.
GET	**GET /files/pictures/<FILENAME>/** Get a device picture identified by <FILENAME>.
PUT	**PUT /files/pictures/<FILENAME>/** Replace a device picture's metadata identified by <FILENAME>. See [Picture PUT Request Data] for request body example.
POST	**POST /files/pictures/<FILENAME>/** Upload a device picture identified by <FILENAME>. See [Picture POST Request Data] for request body example.
DELETE	**DELETE /files/pictures/<FILENAME>/** Delete a device picture identified by <FILENAME>.

Picture PUT Request Data

```
{ "metadata": { "Device Type": "ONU" } }
```

Picture POST Request Data

```
{ "file": <base64 file data>,
  "metadata": { "Device Type": "ONU" } }
```

SLAs

The /slas/ resource and endpoints are used to manage Service Level Agreements. An SLA resource is identified by the configuration ID and maps to a SLA-CFG document in MongoDB. See [PON Controller] for more details on the content of the SLA-CFG document. See the SLA-CFG.json schema file for a description of the parameters, data types, and ranges.

HTTP Method	Endpoint
GET	**GET /slas/** Get a list of SLA-CFG documents. URL Parameters: query (optional), projection (optional), sort (optional), limit (optional), skip (optional), next (optional), distinct (optional)
POST	**POST /slas/** Create an SLA-CFG document.
GET	**GET /slas/<CFG ID>/** Get an SLA-CFG document identified by <CFG ID>.
PUT	**PUT /slas/<CFG ID>/** Create or replace an SLA-CFG document identified by <CFG ID>.
DELETE	**DELETE /slas/<CFG ID>/** Delete an SLA-CFG document identified by <CFG ID>.

Downstream QoS Maps

The /downstream-maps/ resource and endpoints are used to manage downstream QoS mapping configuration profiles. A Downstream QoS Map resource is identified by the configuration ID and maps to a DS-MAP-CFG document in MongoDB. See [PON Controller] for more details on the content of the DS-MAP-CFG document. See the DS-MAP-CFG.json schema file for a description of the parameters, data types, and ranges.

HTTP Method	Endpoint
GET	**GET /downstream-maps/** Get a list of DS-MAP-CFG documents. URL Parameters: query (optional), projection (optional), sort (optional), limit (optional), skip (optional), next (optional), distinct (optional)
POST	**POST /downstream-maps/** Create a DS-MAP-CFG document.
GET	**GET /downstream-maps/<CFG ID>/** Get a DS-MAP-CFG document identified by <CFG ID>.
PUT	**PUT /downstream-maps/<CFG ID>/** Create or replace a DS-MAP-CFG document identified by <CFG ID>.
DELETE	**DELETE /downstream-maps/<CFG ID>/** Delete an DS-MAP-CFG document identified by <CFG ID>.

ONU Service Configurations

The /service-configs/ resource and endpoints are used to manage Service Configurations. A SRV-CFG resource is identified by the configuration ID and maps to a SRV-CFG document in MongoDB. See [PON Controller] for more details on the content of the SRV-CFG document. See the SRV-CFG.json schema file for a description of the parameters, data types, and ranges.

HTTP Method	Endpoint
GET	**GET /service-configs/** Get a list of SRV-CFG documents. URL Parameters: query (optional), projection (optional), sort (optional), limit (optional), skip (optional), next (optional), distinct (optional)
POST	**POST /service-configs/** Create a SRV-CFG document.
GET	**GET /service-configs/<CFG ID>/** Get a SRV-CFG document identified by <CFG ID>.
PUT	**PUT /service-configs/<CFG ID>/** Create or replace a SRV-CFG document identified by <CFG ID>.
DELETE	**DELETE /service-configs/<CFG ID>/** Delete a SRV-CFG document identified by <CFG ID>.

CPEs

The /cpes/ resource and endpoints are used to monitor CPEs. A CPE resource is identified by the CPE MAC Address and maps to the CPE-STATE collection in MongoDB. See [PON Controller] for more details on the content of the CPE-STATE collection. See the CPE-STATE.json schema file for a description of the parameters, data types, and ranges.

HTTP Method	Endpoint
GET	**GET /cpes/states/** Get a list of CPE-STATE documents. URL Parameters: query (optional), projection (optional), sort (optional), limit (optional), skip (optional), next (optional), distinct (optional)
GET	**GET /cpes/states/<CPE ID>/** Get an CPE-STATE document identified by <CPE ID> MAC address.
DELETE	**DELETE /cpes/states/<CPE ID>/** Delete the CPE-STATE document identified by <CPE ID> MAC address.

Databases

The /databases/ resource and endpoints are used to manage database connections. A database resource is identified by the Database ID.

HTTP Method	Endpoint
GET	**GET /databases/** Get a set of all databases with their connection information.
GET	**GET /databases/<DATABASE ID>/** Get the database connection information identified by <DATABASE ID>.
PUT	**PUT /databases/<DATABASE ID>/** Update the database connection information identified by <DATABASE ID>. See [Database Create and Update Request Data] for the request body example.
POST	**POST /databases/<DATABASE ID>/** Create the database connection identified by <DATABASE ID>. See [Database Create and Update Request Data].
DELETE	**DELETE /databases/<DATABASE ID>/** Delete the database connection identified by <DATABASE ID>.
GET	**GET /databases/selection/** Get the database connection currently selected to be used for this user.
PUT	**PUT /databases/selection/** Update the database connection currently selected to be used for this user.
GET	**GET /databases/session/** Get the database connection currently selected to be used for this session only.
PUT	**PUT /databases/session/** Update the database connection currently selected to be used for this session only.
GET	**GET /databases/status/** **v1:** Get the connection status of all database connections. **v2:** Get the connection status and details of all database connections.

GET	**GET /databases/status/<DATABASE ID>/** **v1:** Get the connection status of the database connection identified by <DATABASE ID>. **v2:** Get the connection status and details of the database connection identified by <DATABASE ID>.

Database Create and Update Request Data


```
{
        "auth_db": "tibit_users",
        "auth_enable": false,
        "ca_cert_path": "",
        "db_uri": "",
        "dns_srv": false,
        "host": "127.0.0.1",
        "name": "tibit_pon_controller",
        "password": "pdmPass",
        "port": "27017",
        "replica_set_enable": false,
        "replica_set_hosts": [
                "localhost",
        "127.0.0.1"
        ],
        "replica_set_name": "rs0",
        "tls_enable": false,
        "username": "pdmPonManager"
}
```

Users

The /users/ resource and endpoints are used to authenticate users. See [PON Manager] for more details on user accounts and authentication.

HTTP Method	Endpoint
GET	**GET /users/logout/** Log out the user and invalidate their session.
POST	**POST /users/authenticate/** Login the user identified by the POST data. See [User Login Request Data] for the request body example.

User Login Request Data

```
{
        "email": "api.user@gmail.com",
        "password": "apiUserPassword"
}
```

PON Manager

The /ponmgr/ resource and endpoints are used to access PON Manager installation information.

HTTP Method	Endpoint
GET	**GET /ponmgr/version/** Get a set of attributes describing the installed version of PON Manager.

Use Cases

Common use cases and examples using the Cisco Routed PON REST API are outlined in this section.

Examples

The PON Manager installation package contains several examples using the Cisco Routed PON REST API using Curl, Postman, and Python. The examples are located under the /opt/tibit/ponmgr/examples directory.

Curl

The Curl examples are located under the /opt/tibit/ponmgr/examples/curl directory. To install cURL on Linux run the following commands.

```
sudo apt update && sudo apt upgrade
sudo apt install curl
```

The following is an example of a GET request which retrieves an OLT configuration document from the database. The GET request must include the cookies provided from the login response. In the example below, the response is saved to the file OLT-CFG.json.

```
curl -X GET https://10.1.20.102/api/v1/olts/configs/70:b3:d5:52:35:ac/ \
        -b cookies.txt -k \
        > OLT-CFG.json
```

The following is an example of a PUT request which writes/updates an OLT configuration document to MongoDB. The PUT request must include the payload, content type header, CSRF token, cookies, and referrer URL. In the example below, the request paylod is contained in the file OLT-CFG.json.

```
curl -X PUT https://10.1.20.102/api/v1/olts/configs/70:b3:d5:52:35:ac/ \
        -d @OLT-CFG.json \
        -H "Content-Type: application/json" \
        -H "X-CSRFToken:$(grep csrftoken cookies.txt|sed 's/^.*csrftoken\s*//')"\
        -b cookies.txt -k \
        -e https://10.1.20.102/api
```

More examples can be found in *api/examples/curl/curlExamples.md*.

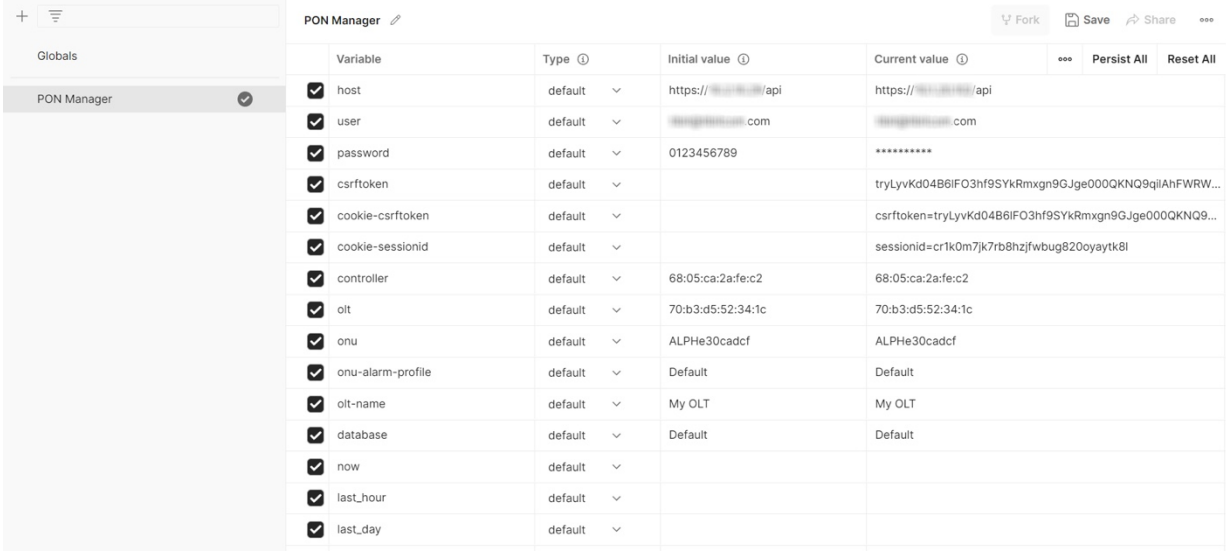
Postman

The Postman examples are located under the /opt/tibit/ponmgr/examples/postman directory. See Postman for information on installing and using Postman.

Getting started with the Postman examples.

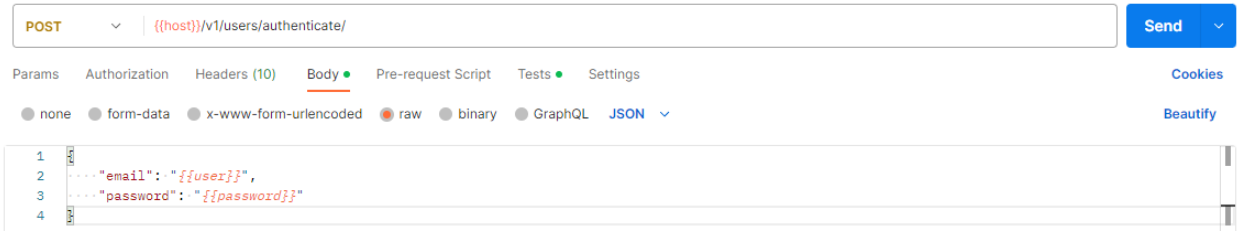
1. Import the postman_mcms_restapi_R2.3.0 folder into postman.
   Note: Newer versions of Postman may require an account

2. Go to the Environments tab and click on PON Manager. Host, User, and Password need to be set. Other variables can be set if desired.



523687

3. Go to the Collections tab and click on the ‘Login’ POST in the tree. The ‘host’, ‘user’, and ‘password’ variables will use the values from the Environment. Click Send and this will authenticate you and set your session variables.



4. Navigate to a GET request and query parameters can be added to directly the url or in the Params tab.

Python

The Python examples are located under the /opt/tibit/ponmgr/examples/python directory. The Python examples require Python version 3 and the 'requests' Python Package form pypi.org.
Use the pip3 command to install the 'requests', enter the following into the command line.

```
pip3 install requests
```

The following example configures service for a subscriber, where the OLT pushes/pops S-Tag 200 and the ONU pushes/pops C-Tag 25.

```
python3 config_addctag_service.py --url https://10.2.10.29/api \
        --user <email> --password <password> \
        --olt e8:b4:70:70:0c:9c --olt_tag 200 \
        --onu ALPHe30cadcf --onu_tag 25
```

The following example queries the registration status for all ONUs configured under the specified OLT.

```
python3 get_onu_registration_status.py --url https://10.2.10.29/ \
        --user <email> --password <password> \
        --olt e8:b4:70:70:0c:9c
```

The following example resets an ONU.

```
python3 reset_onu.py --url http://10.2.10.29/api \
        --user <email> --password <password> \
        --onu ALPHe30cadcf
```

More examples can be found in *api/examples/python/*

Programming Model

This section describes the common programming model for the Cisco Routed PON REST API and this general sequence is required for each or use cases presented in sections that follow. See API Reference for more details on the header requirements, request/response message format, and programming model.

After a successful login and establishing a secure authenticated session, all API requests must contain the following HTTP headers (see Request Format for more details):

- Session ID Cookie
- CSRF Token Cookie

Additionally, all PUT and POST requests must also contain the following HTTP headers:

- Content-Type: application/json
- X-CSRF Token

- Referrer

API sessions generally have the following sequence:

1. Log in to authenticate the user with the web server
   POST /v1/users/authenticate/

2. (Optional) Select the database to use for the session. If skipping this step, the user's default database (typically 'Default') is used for the session.
   PUT /v1/databases/selection/

3. Execute one or more request(s) to fetch data and/or configure the database. For example,
   GET /v1/onus/configs/BFWS00123193
   ...
   PUT /v1/onus/configs/BFWS00123193

4. Log out of the web server.
   GET /v1/users/logout/

Use Case: ONU Registration Status

Two methods can be used to determine the registration status of an ONU: 1) querying PON Controller state or 2) querying the OLT state.

ONU Registration Status using CNTL-STATE

To determine the status of an ONU through the PON Controller state use the steps outlined below.

1. Query ONU-STATE of the ONU by GPON Serial Number or EPON MAC Address.

   GET <URL>/v1/onus/states/<ONU ID>/

   Look up the PON Controller MAC Address and OLT MAC Address that this ONU is attached to.
   - ONU-STATE.CNTL.MAC Address (blue highlight in the document below)
   - ONU-STATE.OLT.MAC Address (red highlight in the document below)

```
ONU-STATE = {
  "_id" : "BFWS00123193",
  "CNTL" : {
    "MAC Address" : "68:05:ca:2a:fe:c2",
    "Version" : "R4.0.0-rc99"
  },
  "OLT" : {
    "MAC Address" : "e8:b4:70:70:0c:9c",
    "Reg Disallowed" : false
```

```
    },
    ...
    "Time" : "2023-07-19 20:40:08.730244"
}
```

2. Query CNTL-STATE of the PON Controller by MAC Address

   GET <URL>/v1/controllers/states/<CNTL ID>/

   Note: The timestamp in the CNTL-STATE file can be checked to determine the current status of the PON Controller. If the “Time” is no older than 2 minutes from the current time, then the controller is assumed to be online. If the controller is online, it is possible that the ONU is also online.

3. Look up the ONU Registration status from CNTL-STATE.

   The ONU Status is reported under the following path in the CNTL-STATE document.
   System Status[<OLT ID>].ONUs[<ONU ID>]

```
CNTL-STATE = {
   "_id" : "68:05:ca:2a:fe:c2",
   ...
   "System Status" : {
      "e8:b4:70:70:0c:9c" : {
         "OLT State" : "Primary",
         "ONUs" : {
            "ISKT71e81e78" : "Registered",
            "BFWS00123193" : "Registered",
            "ALPHe30cadcf" : "Registered",
            "CIEN00099729" : "Registered",
            "TBITc84c0083" : "Registered"
         }
      },
   },
   "Time" : "2023-07-19 20:49:22.596464"
}
```

ONU Registration Status using OLT-STATE

To determine the status of an ONU through the OLT state use the steps outlined below.

1. Query ONU-STATE of the ONU by GPON Serial Number or EPON MAC Address.

   GET <URL>/v1/onus/states/<ONU ID>/

   Look up the OLT MAC Address that this ONU is attached to.
   - ONU-STATE.OLT.MAC Address (red highlight in the document below)

```
ONU-STATE = {
   "_id" : "BFWS00123193",
   "CNTL" : {...},
   "OLT" : {
      "MAC Address" : "e8:b4:70:70:0c:9c",
      "Reg Disallowed" : false
   },
   ...
   "Time" : "2023-07-19 20:40:08.730244"
}
```

2. Query OLT-STATE of the OLT by MAC Address

```
GET <URL>/v1/olts/states/<OLT ID>/
```

Look up the Unspecified ONUs and Registered ONUs from the ONU States field in the OLT-STATE document.

- OLT-STATE.ONU States.Registered (green highlight in the doc below)

or

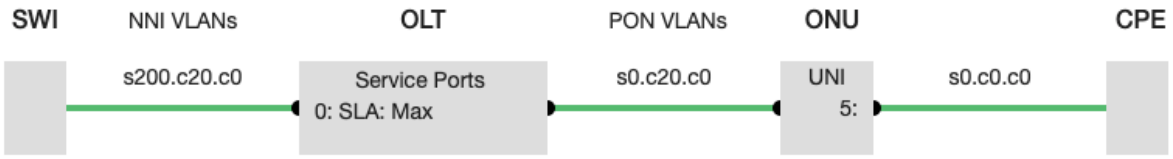
- OLT-STATE.ONU States.Unspecified

```
OLT-STATE = {
   "_id" : "e8:b4:70:70:0c:9c",
   ...
   "ONU States" : {
         “Disabled” : [],
         “Deregistered” : [],
         “Disallowed Admin” : [],
         “Disallowed Error” : [],
         “Disallowed Reg ID” : [],
         “Dying Gasp” : [],
         “Registered” : [
               0 : "BFWS00123193"
         ],
         “Unspecified” : [],
      “Unprovisioned” : []
   },
   ...
}
```

If the ONU is in either the “Registered” or “Unspecified” state, then it is online.

Use Case: Provision service for an ONU

This use case provides an overview of what is typically required to provision service for a subscriber on an ONU. This use case configures the OLT to push/pop an outer S-Tag identifying PON port and configures the ONU to push/pop and inner C-Tag identifying the ONU. This configuration shown in the diagram below, where frames on the NNI are double tagged with S-Tag + C-Tag, frames on the PON have single tagged with C-Tag only, and frames on the UNI are untagged or priority tagged.



Inputs required for this use case are as follows:

- ONU ID – ONU ID used to identify the ONU for the XGS-PON OMCC channel.
- ALLOC ID(s) – TCONT ALLOC ID and XGEM port values assigned for this service.
- OLT S-Tag – SVLAN ID identifying the PON Port.
- ONU C-Tag – CVLAN ID identifying the ONU.

Provisioning service requires the following steps:

1. Inventory the ONU to configure the GPON ONU ID and ALLOC IDs required for the service.
2. Inventory the NNI Network to configure the VLAN Tagging required for the service.
3. Set the ONU Service configuration file and related service attributes.
4. Enable the OLT Service port(s) and configure SLA(s) and VLAN(s).

The detailed API sequence for this use case is presented below:

1. Get the OLT configuration that this ONU is connected to.

```
GET <URL>/v1/olts/configs/<OLT ID>/
```

2. Inventory the ONU. Configure the GPON ONU ID and ALLOC IDs for each OLT Service port required for this configuration. The number of OLT Service ports configured depends on the number ALLOC IDs required for the service. If this is a new entry, include the ‘Disable’, ‘Enable Count’ and ‘Disable Count’ in the new ONU sub-document. The following fields need to be configured:
   - OLT-CFG.ONUs.<ONU ID>. ALLOC ID (OMCC)
   - OLT-CFG.ONUs.<ONU ID>. OLT-Service 0
   - OLT-CFG.ONUs.<ONU ID>. OLT-Service *n*

```
OLT-CFG = {
   "_id" : "<OLT ID>",
   ...
   "ONUs" : {
      "<ONU ID>" : {
            “ALLOC ID (OMCC)” : <ONU ID>,
                  “Disable” : False,
            “Enable Count” : 0,
                  “Disable Count” : 0,
```

```
                    “OLT-Service 0” : <ALLOC ID #1>,
                    “OLT-Service n” : <ALLOC ID #n>
    }
        },
        ...
    }
```

3. Inventory the NNI Network. Configure the VLAN Tags for the NNI Network. The following fields need to be configured:
    - OLT-CFG.NNI Networks.TAG MATCH

```
OLT-CFG = {
   "_id" : "<OLT ID>",
   ...
   "NNI Networks" : [
                {
                "TAG MATCH" : "s<OLT S-Tag>.c<ONU C-Tag>.c0",
                "SLA-CFG" : {
                   "Source" : "N/A"
                },
                "Filter" : {
                   "DHCPv4" : "pass",
                   "DHCPv6" : "pass",
                   "EAPOL" : "pass",
                   "PPPoE" : "pass"
                },
                "Learning Limit" : 2046
                 },
   ...

     ]
  ...
}
```

4. Write the modified OLT configuration to MongoDB.

```
PUT <URL>/v1/olts/configs/<OLT ID>/
```

5. Get the ONU configuration document from MongoDB.

```
GET <URL>/v1/onus/configs/<ONU ID>/
```

6. Configure the ONU Service Configuration file and related attributes. The following fields need to be configured:
    - ONU-CFG.ONU.SRV-CFG
    - ONU-CFG.ONU.CVID
    - ONU-CFG.SRV-CFG Values

```
ONU-CFG = {
   "_id" : "<ONU ID>",
     ...
```

```
  "ONU" : {
     "Enable" : true,
        ...
        “SRV-CFG” : “Add CTag”,
        “CVID” : <ONU C-Tag>,
        ...
  },
     ...
}
```

7. Enable and configure the OLT Service port(s) and related attributes. The following fields need to be configured:
   - ONU-CFG.OLT-Service.*n*.Enable
   - ONU-CFG.OLT-Service.*n*.NNI Networks
   - ONU-CFG.OLT-Service.*n*.PON Networks
   - ONU-CFG.OLT-Service.*n*.SLA

```
ONU-CFG = {
  "_id" : "<ONU ID>",
     ...
  "OLT-Service 0" : {
    "Enable" : true,
        ...
        “NNI Networks” : [“s<OLT S-Tag>.c<ONU C-Tag>.c0”],
        “PON Networks” : [“s0.c<ONU C-Tag>.c0”],
        ...
        "SLA-CFG" : "Max",
        ...
  },
     ...
}
```

8. Write the modified ONU configuration to MongoDB.

```
PUT <URL>/v1/onus/configs/<ONU ID>/
```

Use Case: Disable service for an ONU

This use case disables and removes the service configuration for an ONU. Essentially, disabling and deleting service is the reverse of Use Case: Provision service for an ONU. Some steps in this use case are optional. For example, if simply disabling the service without removing all configuration, the only step required is to set ONU-CFG.OLT-Service 0.Enable to a value of ‘false’.

Disabling service requires the following steps:

1. Disable the OLT Service port(s) and remove the VLAN configuration.
2. Set the ONU Service configuration to ‘Disabled’ and related service attributes.
3. Delete the OLT NNI Network.
4. Remove the device from ONU Inventory.

The detailed API sequence for this use case is presented below:

1. Get the ONU configuration document from MongoDB.

   GET <URL>/v1/onus/configs/<ONU ID>/

2. Set the ONU Service Configuration to ‘Disabled’ and clear related attributes to default values. The following fields need to be configured:
   - ONU-CFG.ONU.SRV-CFG
   - ONU-CFG.ONU.CVID
   - ONU-CFG.SRV-CFG Values

```
ONU-CFG = {
   "_id" : "<ONU ID>",
      ...
   "ONU" : {
      "Enable" : true,
         ...
         “SRV-CFG” : “Disabled”,
         “CVID” : 0,
         ...
   },
      ...
}
```

3. Disable the OLT Service port(s) and clear related attributes to default values. The following fields need to be configured:
   - ONU-CFG.OLT-Service.*n*.Enable
   - ONU-CFG.OLT-Service.*n*.NNI Networks
   - ONU-CFG.OLT-Service.*n*.PON Networks
   - ONU-CFG.OLT-Service.*n*.SLA

```
ONU-CFG = {
   "_id" : "<ONU ID>",
      ...
   "OLT-Service 0" : {
      "Enable" : false,
            ...
            “NNI Networks” : [“s<OLT S-Tag>.c<ONU C-Tag>.c0”],
            “PON Networks” : [“s0.c<ONU C-Tag>.c0”],
            ...
            "SLA-CFG" : "Max",
            ...
   },
      ...
}
```

4. Write the modified ONU configuration to MongoDB.

   PUT <URL>/v1/onus/configs/<ONU ID>/

5. Get the OLT configuration that this ONU is connected to.

   GET <URL>/v1/olts/configs/<OLT ID>/

6. Delete the NNI Network. The NNI Network entry needs to be removed from the OLT configuration document:
   - OLT-CFG.NNI Networks

```
OLT-CFG = {
   "_id" : "<OLT ID>",
   ...
   "NNI Networks" : [
            {
      "TAG MATCH" : "s<OLT S-Tag>.c<ONU C-Tag>.c0",
      "SLA-CFG" : {
         "Source" : "N/A"
      },
      "Filter" : {
         "DHCPv4" : "pass",
         "DHCPv6" : "pass",
         "EAPOL" : "pass",
         "PPPoE" : "pass"
      },
      "Learning Limit" : 2046
            },
   ...

     ]
  ...
}
```

7. Remove the ONU from the OLT's ONU Inventory. The ONU entry needs to be removed from the OLT configuration document:
   - OLT-CFG.ONUs.<ONU ID>.

```
OLT-CFG = {
   "_id" : "<OLT ID>",
   ...
   "ONUs" : {
    "<ONU ID>" : {
        "ALLOC ID (OMCC)" : <ONU ID>,
            "Disable" : False,
      "Enable Count" : 0,
            "Disable Count" : 0,
            "OLT-Service 0" : <ALLOC ID #1>,
            "OLT-Service n" : <ALLOC ID #n>
}
   },
   ...
}
```

8. Write the modified OLT configuration to MongoDB.

```
PUT <URL>/v1/olts/configs/<OLT ID>/
```

Use Case: Firmware upgrade for an ONU

ONU firmware download is initiated through the ONU configuration document in MongoDB. Modify the ONU-CFG with the desired firmware bank, files, and versions as listed below to initiate an ONU firmware download.

1. Get the configuration document for the ONU to be upgraded.

```
GET <URL>/v1/onus/configs/<ONU ID>/
```

2. Set the firmware bank versions, firmware bank files, and firmware bank pointer to the desired firmware settings. Update the following fields:
    - ONU-CFG.ONU.FW Bank Ptr – Select bank ‘0’ or bank ‘1’
    - ONU-CFG.ONU.FW Bank Files[0] – Firmware filename for bank 0
    - ONU-CFG.ONU.FW Bank Versions[0] – Firmware version string for bank 0
    - ONU-CFG.ONU.FW Bank Files[1] – Firmware filename for bank 1
    - ONU-CFG.ONU.FW Bank Versions[1] – Firmware version string for bank 1

```
ONU-CFG = {
   "_id" : "<ONU ID>",
           ...
   “ONU” : {
            “FW Bank Files”: [
         “0” : “FW-GPON-CIEN-3802_91x-V2.3.10.bin”
         “1” : “FW-GPON-CIEN-3802_91x-V2.4.3.bin”
           ],
           "FW Bank Versions" : [
              "0" : “V2.3.1”
              "1" : “V2.4.3”
           ],
           “FW Bank Ptr”: 1,
           ...
   }
}
```

3. Write the modified ONU configuration to MongoDB.

```
PUT <URL>/v1/onus/configs/<ONU ID>/
```

Use Case: Reset an ONU

ONU reset is implemented as an action using the Cisco Routed PON REST API. The following PUT command with the ONU reset endpoint triggers a reset of the ONU.

PUT <URL>/v1/onus/${ONU}/reset/

Debug and Troubleshooting

If the REST API is not working as intended, it is important to review log messages and gather any information related to the problem that could help with debugging.

Logs

Checking the logs can often provide important details as to what is going wrong. Logs are populated and stored on the machine where PON Manager was installed.

PON Manager logs

PON Manager logs are located at “/var/log/tibit/ponMgr.log”. Log messages including “ERROR” can provide helpful information as to why an operation has been unsuccessful. It is also important to check the status code associated with a request. Reference common HTTP status codes for more information.

For example, the following error message appears in the PON Manager logs if a user attempts to get ONU configurations but is not logged in:

```
2023-06-28 18:02:15.942 ERROR 10.3.8.104 unauthenticated GET /v2/onus/configs/ status: 403
```

The user is not authorized to view the ONU configurations and must first use the REST API to log in as a valid user with the correct permissions.

Apache Logs

error.log

Apache error log is located here: /var/log/apache2/error.log, and is most useful when debugging PON Manager configuration issues which can prevent the REST API from working properly.

other_vhosts_access.log

Apache other vhost log is located here: /var/log/apache2/ other_vhosts_access.log. The vhost access logs provide more detailed information about HTTP requests between the web server and virtual host serving the REST API. These logs capture data such as: time, HTTP version, response code, and browser details associated with each request.

Errors

Listed below are some common API errors relating to MongoDB or syntax issues.

MongoDB Connection

The following is a common MongoDB connection error.

```
ERROR MongoDB server at 10.3.8.180:27017 has gone offline due to Heartbeat failure: hello cancelled
```

The log states that the MongoDB server has gone offline due to Heartbeat Failure. One possible cause for this error is that MongoDB is no longer running on the system where the database is located. Another possibility is that the connection information is wrong. The IP address or database name could have a typo or could be pointing to a database that does not exist.

In the situation where the selected database has gone offline, the API will return status code 500 (server error) for most requests. This is because no connection to the database can be established and therefore the API can not retrieve the necessary data.

```
2023-07-28 17:29:04.424 ERROR 192.168.33.1 tibit@tibitcom.com GET /v2/olts/alarm-configs/Default/ status: 500
"GET /v2/olts/alarm-configs/Default/ HTTP/1.1" 500 180
2023-07-28 17:29:04.474 ERROR 192.168.33.1 tibit@tibitcom.com GET /olt/alarm/configuration/identifiers/ status: 500
```

Validation

A request to the API may be failing due to validation errors. The following error occurred when attempting to update an ONU-CFG without the required ‘CNTL’ property. The server will reject a configuration that does not contain all required fields, resulting in a 400 level status code warning, and an INFO level log stating that the JSON validation failed due to the missing required property.

```
WARNING 10.3.8.104 tibit@tibitcom.com PUT /v1/onus/configs/CIEN00000000/ status: 400
```

Bad Request

When the client sends a request that the server does not understand, a 400 level status code is returned. This error can occur when a user makes a request through any means other than the browser. One example of this occurs when the URL is improperly formatted. The URL below includes a random “%” character which causes the error.

The following “invalid URI path” error will appear in the Apache error log.

```
AH10244: invalid URI path (/api/v1/olts/Default/%disable-onu/CIEN00099629/)
```

Americas Headquarters
Cisco Systems, Inc.
San Jose, CA

Asia Pacific Headquarters
Cisco Systems (USA) Pte. Ltd.
Singapore

Europe Headquarters
Cisco Systems International BV Amsterdam,
The Netherlands

Cisco has more than 200 offices worldwide. Addresses, phone numbers, and fax numbers are listed on the Cisco Website at **https://www.cisco.com/go/offices**.