RBFS API User Guide

Version 22.7.1, 25 July 2022

Registered Address	Support	Sales
26, Kingston Terrace, Princeton, New Jersey 08540, United States		
		+91 80 4850 5445
http://www.rtbrick.com	support@rtbrick.com	sales@rtbrick.com

Table of Contents

1. Overview . . . 4
2. RBFS API Fundamentals . . . 6
  2.1. RBFS in a Nutshell . . . 6
  2.2. Low-level BDS API . . . 6
  2.3. Brick Daemon API . . . 7
  2.4. RESTCONF API . . . 7
  2.5. Operational State API . . . 8
  2.6. CTRLD API . . . 8
  2.7. RADIUS . . . 8
  2.8. Summary . . . 9
3. Configurations . . . 10
  3.1. APIGWD . . . 10
    3.1.1. SSL certificate . . . 10
    3.1.2. JWKS File . . . 10
    3.1.3. Caching . . . 11
    3.1.4. config.json . . . 11
    3.1.5. access_secret_jwks.json . . . 12
    3.1.6. tls.pem . . . 12
  3.2. CTRLD . . . 13
    3.2.1. config.json . . . 13
    3.2.2. policy.json . . . 15
    3.2.3. element.config . . . 16
4. Business Events . . . 17
5. Related Documentation . . . 22

Abstract

This document outlines the basic components of RBFS for managing devices running RBFS and their interaction to other systems. It gives an overview of the different types of APIs and outlines the usage of that APIs.

1. Overview

A device running RBFS uses different protocols to interface with the control and management plane and vice versa. This section gives a brief overview of the communication flow. More detailed information will be provided in the next sections and in dedicated documents.

Zero-Touch Provisioning (ZTP) is driven by DHCP options. First the Open Network Installation Environment (ONIE) that is installed on a bare metal switch by default, discovers the RtBrick Full Stack (RBFS) installer image and installs RBFS /ONIE/. ONIE supports HTTP and TFTP to load the installer image. After the first reboot the switch loads the startup configuration. The base URL is read from DHCP option 210 and defaults to the DHCP server if DHCP option 210 is not set /ZTP/. ONIE has only access to the dedicated management interface. Consequently ZTP requires out-of-band access to the DHCP service and to download the RBFS installer image and the startup configuration.

RADIUS messages are used to provision subscriber sessions. All per-subscriber settings are encoded in RADIUS attributes /RADIUS/. RADIUS messages are sent in-band.

Global RBFS configuration like credentials, BGP peerings or CoS profiles for example, are either part of the startup configuration files or applied through REST API invocations.

The RBFS management high-level communication flow and provided APIs are illustrated below:

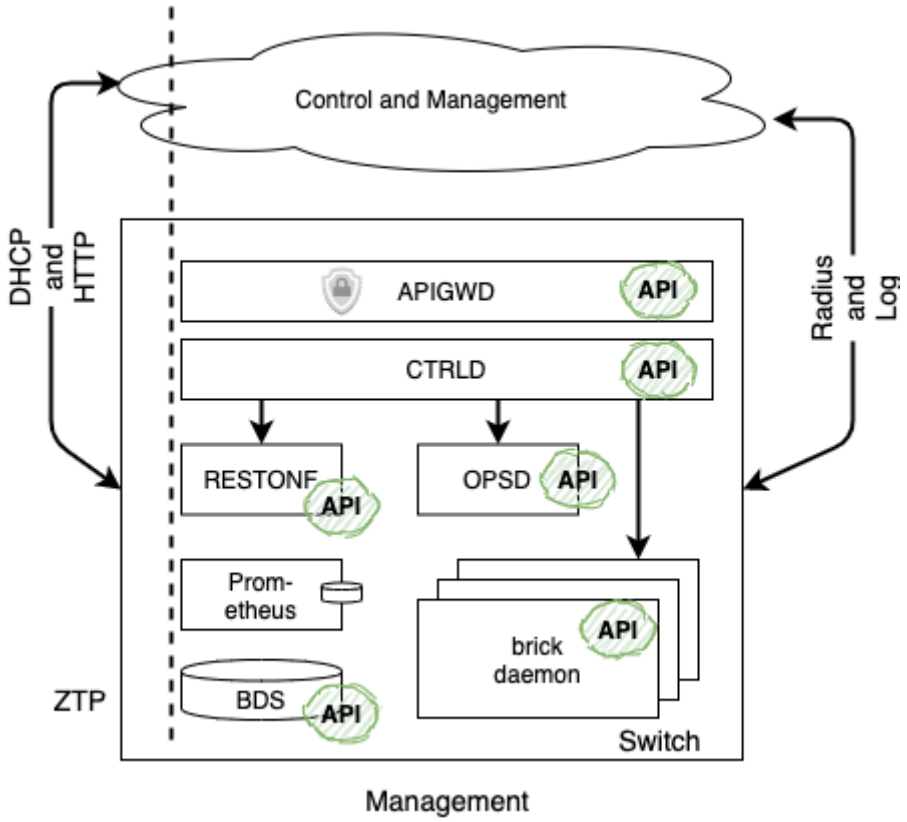

Figure 1. RBFS Management APIs

Most of the RBFS components provide an API.

The API Gateway Daemon (APIGWD) is the TLS endpoint for the switch and converts an external access token into an internal rtbrick token /SEC/. Further it forwards the requests to Control Daemon (CTRLD).

The CTRLD is the entry point to RBFS. In essence it provides a proxy to the other APIs and also higher level APIs for switch configuration. Apart from that it handles the ZTP post process.

The Operational State API allows inspecting the operational state of a switch and running actions to diagnose and troubleshoot problems. The operational state is ephemeral and is lost when the switch reboots, which differentiates it from the persistent switch configuration. The Operational State API is provided by the Operational State Daemon (OPSD).

The RBFS configuration is managed through RESTCONF.

The Brick Data Store (BDS) provides a really low-level API to the BDS tables and objects.

Every Brick Daemon (BD) can provide additional APIs beside the BDS API. Additionally to these REST APIs also parts of the RADIUS protocol are implemented.

2. RBFS API Fundamentals

2.1. RBFS in a Nutshell

RtBrick Full Stack (RBFS) architecture basics and an understanding of the RBFS API characteristics is needed to make use of the RBFS APIs. The Brick Data Store (BDS) is a schema-driven in-memory database optimized for networking use cases. It stores objects, which are described in schemas, and organizes them in different tables. Each brick daemon has a single responsibility and defines the tables and schemas needed for its respective task. Brick daemons interact with each other over a pub-sub facility provided by the BDS.

RBFS differentiates between persistent and non-persistent configuration. Non-persistent configuration is lost after each switch-reboot whereas persistent configuration is written to disk. Service related information is an example for non-persistent configuration. All service information is lost when a switch is rebooted and has to be pushed again on demand by the controller after the reboot. The startup configuration is an example for persistent data. Whenever startup related configuration gets changed, e.g. by adding a new BGP peering, the startup configuration file gets updated to preserve the new settings if the switch reboots.

2.2. Low-level BDS API

The BDS API provides access to the BDS tables and objects. The BDS API relies on RPC over HTTP and defines an envelope protocol to convey BDS objects. The data exchange format is JSON.

The BDS API supports adding objects, removing objects and walking through a table. Working with BDS tables and objects requires awareness of the indices that exist on each table. The add operation adds new or replaces existing objects.

The BDS API only validates syntactic but not semantic integrity.

The delete operation removes objects. A merge operation to add and remove objects in one go does not exist. Walking a table requires to specify the index the walk operation should follow. The sequence index is used by default if no index is specified. The sequence index represents the temporal insertion order of objects into a table. The objects are returned in the index sort order. Searching for objects requires an understanding of key and non-key attributes of each index.

The usage of the BDS API also requires an understanding of the relations between the objects. It can be compared to referential integrity in SQL databases which also forces the client to modify database records in the right order. Using the BDS API for integration is as if you would use SQL (or the database model) rather than an API of a software product relying on a relational database.

The BDS schema definitions get constantly honed and can become subject of non-backward compatible changes.

For example, BDS schemas definitions can be modified to simplify CLI implementation without changing the data that is being exchanged as such. Even a patch can introduce a non-backward compatible BDS schema definition change.

For every brick daemon there is a dedicated BDS API endpoint.

2.3. Brick Daemon API

Every brick daemon provides additional APIs beside the BDS API. The configuration daemon is a good example. It introduces a transactional configuration API that allows you to incrementally update a candidate configuration that is later applied in one go. The configuration daemon detects deltas between configurations and only the detected deltas trigger subsequent actions. For example, if a new BGP peer has been added to the candidate config, the additional BGP peering gets established without resetting established peerings.

The configuration daemon processes the persistent configuration.

All configurations applied via config daemon will be added to the startup configuration file and applied after every reboot. Not all BDS tables become part of the persistent configuration by default. A brick daemon can declare a BDS table persistent. In this case the ownership of the table is transferred from the respective brick daemon to the configuration daemon. The brick daemon cannot modify the objects in BDS tables when the ownership has been transferred to configuration daemon. Consequently, the BDS low-level API can still be used for reading objects but not for object modifications. All changes are applied by the configuration daemon. The configuration daemon provides a delta processing and the brick daemons get notified about configuration changes via the BDS pub-sub facility mentioned earlier.

2.4. RESTCONF API

> — According to RFC 8040
> RESTCONF is an HTTP-based protocol that provides a programmatic interface for accessing data defined in YANG, using the datastore concepts defined in NETCONF.

The YANG models describe the configuration syntax. The RESTCONF API provides the configuration data only, while the state information is provided by the Operational State API.

2.5. Operational State API

The Operational State API forms a stable contract between network management systems and RBFS. The API is backward compatible to support running multiple versions of RBFS in the network, which is typically the case when a new RBFS release is rolled out in the network. In addition, the formal Operational State API contract simplifies developing management system integrations by using API mock-ups rather than running RBFS instances with the respective services configured. This lowers the entry-barrier for network management integration for both, customers and integration partners. The Operational State API is implemented in Python which allows customers, integration partners, professional services and engineering to collaborate on the API endpoints.

2.6. CTRLD API

The control daemon (CTRLD) is the entry point to RBFS. In addition, CTRLD implements the ZTP startup configuration discovery by means of downloading the startup configuration files from the retrieved base URL and applying it to the switch.

CTRLD exposes a REST-API to program the switch, which is kept backward compatible /CTRLD/. CTRLD itself uses the low-level BDS API as needed. For example, CTRLD API supports to specify which metric shall be sampled and to define alert conditions which are then stored in BDS tables. CTRLD takes care of compatibility issues outlined earlier. RBFS leverages Prometheus for metric sampling. All counters and metrics of interest shall be sampled by Prometheus and accessed through PromQL rather than the BDS API. A local history of all sampled metrics is also very handy for troubleshooting.

The CTRLD API also provides access to the low-level BDS API endpoints. The usage of the BDS API is an indication that a CTRLD API endpoint is missing. It is recommended to define an external representation of state implemented by the CTRLD API rather than extensively working with the BDS objects in order to avoid compatibility issues. Since CTRLD runs on the switch it needs only to support the current BDS schema definition to implicitly provide backward compatibility.

2.7. RADIUS

The subscriber session establishment relies on RADIUS. All per-subscriber settings are exchanged through RADIUS messages whereas information that is shared across multiple subscribers, like QoS profiles, need to be configured through API invocations.

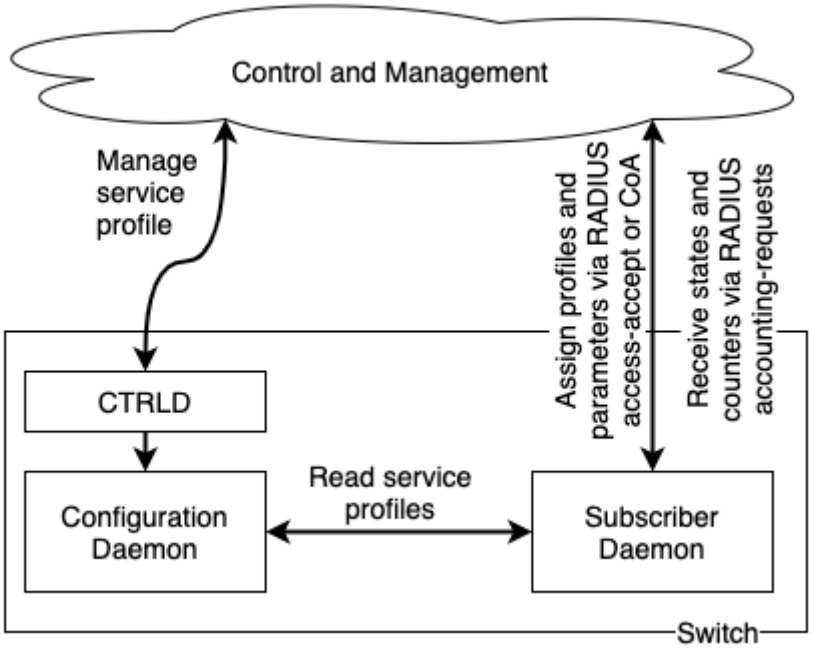

Figure 2. Overview of REST API and RADIUS calls for subscriber management

Subscriber session settings are non-persistent and are lost if a switch reboots or the session ends. The sessions must be established again after each reboot. RADIUS relies on UDP for maximum throughput and minimum latency. Message loss is handled by the application rather than the transport layer. The application asks for the message again if the message is not delivered in time. In addition the application can cope with receiving the same message multiple times.

2.8. Summary

The previous sections provided a quick overview of the existing RBFS APIs and what they are intended for:

- Use the CTRLD API to maintain the persistent configuration. All settings that shall preserve a switch reboot need to become part of the persistent configuration. Running configuration supports a delta processing and only identified deltas trigger an action on the switch. QoS profiles are a good example for persistent configurations.
- Use the CTRLD API to configure sampling of sensors and counters. Use PromQL, the prometheus query language, to access the sampled values. Additionally program alert conditions as needed. All alerts are sent to a GELF log infrastructure /GELF/.
- Use RADIUS messages to exchange per-subscriber settings for session establishment.
- Be aware that BDS schemas can change when using the low-level BDS API.

3. Configurations

This sections describes the configuration files of the APIGWD and CTRLD.

3.1. APIGWD

In a production environment, the APIGWD binary starts with default parameters. This service of APIGWD is called rtbrick-apigwd.

To see the default parameters and therefore also the files where configurations are stored, the easiest way to figure out is to do a apigwd -help.

To see the actual installed APIGWD version we can use apigwd -version.

APIGWD has in essence 3 important configuration files:

- /etc/rtbrick/apigwd/config.json: Configuration for the apigwd
- /etc/rtbrick/apigwd/access_secret_jwks.json: JWKS file for external communication
- /etc/rtbrick/apigwd/tls.pem: X509 public/private key file in pem format

3.1.1. SSL certificate

Every call to the APIGWD is secured by TLS. If there is no TLS certificate provided, it is one generated and signed by a self signed root CA.

To specify a TLS certificate there are the following possibilities to achieve this:

- Provide the TLS file via ZTP
- Provide the TLS remote file URLs via the config.js:
  Therefore you can also specify a reload-time. Every x seconds APIGWD tries to download a new TLS file. But for downloading a RFC 7234 compliant cache is used.

3.1.2. JWKS File

The APIGWD validates the access token against an JSON Web Key Set (JWKS) (https://tools.ietf.org/html/rfc7517).

APIGWD allows to specify 2 source for keyset, and for a validation the sets are consulted in the following order:

- A local file on the filesystem:
  This file can be provided via ZTP. It is recommended to provide one file, even if it is an empty key set file, otherwise there a preconfigured file on the system

that will be used.

- Provide the JWKS remote file URLs via the config.js:
  Everytime a token has to be verified the JWKS file will be downloaded, but for the download an RFC 7234 compliant cache is used.
- Provide the OpenIDConnect configuration via the config.js:
  Everytime a token has to be verified the Issuer is consulted to get the link to the JWKS file, and the file will be downloaded, but for the download a RFC 7234 compliant cache is used.

3.1.3. Caching

As stated above, for configuration file downloads a RFC 7234 compliant cache is used. The cache directives should be used wise, otherwise a lot of traffic could be generated.

3.1.4. config.json

This section describes the main configuration file of APIGWD. This file can be changed on the file system, APIGWD has a file watcher on the file, and does a reload when the file changes.

/etc/rtbrick/apigwd/config.json example

```
{
  "access_token_jwks_urls": [
    "http://192.168.202.56:8080/primaryJWKS",
    "http://192.168.202.56:8080/secondaryJWKS"
  ],

  "request_rate": 5,
  "request_burst": 10,
  "report_rejects_every": 10
}
```

Table 1. /etc/rtbrick/apigwd/config.json format

Name	Type	Description
access_token_jwks_urls	[]string	Allows to specify multipe jwks remote urls.
Remote PEM file		
pem_urls	[]string	Allows to specify multipe pem remote urls. Empty list disables the download.
pem_reload_time	int	Allows to specify the time after a new reload is triggerd. 0 disables the download.
Request rate limit		

request_rate	float	The allowed requests per second per client.
request_burst	int	Is the maximum number of tokens that can be consumed at once, without respect the rate.
report_rejects_every	int	Report rejects only every x seconds, to avoid massive logging to a gelf endpoint.

3.1.5. access_secret_jwks.json

This section describes the access_secret_jwks.json file. This file can be changed on the file system, APIGWD has a file watcher on the file, and does a reload when the file changes.

JSON Web Key Set (JKWS) is described in the RFC 7517.

/etc/rtbrick/apigwd/access_secret_jwks.json example

```
{
  "keys": [
    {
      "kty": "RSA",
      "e": "AQAB",
      "use": "sig",
      "kid": "access",
      "alg": "RS256",
      "n": "NOT A REAL KEY"
    }
  ]
}
```

This keys are for authentication of external call towards to the APIGWD.

The right key is selected by the kid (key id). With this key the access tokens are verified and converted to a rtbrick token.

3.1.6. tls.pem

This section describes the tls.pem file. This file contains the TLS certificate (public and private key) used to serve the TLS endpoint.

If the file is not specified a new self signed certificate is created.

This file can be changed on the file system, APIGWD has a file watcher on the file, and does a reload when the file changes.

This file is an X509 public/private key file in PEM format specified in the RFC7468 .

/etc/rtbrick/apigwd/tls.pem example

```
-----BEGIN CERTIFICATE-----
NOT A REAL KEY
-----END CERTIFICATE-----
-----BEGIN RSA PRIVATE KEY-----
NOT A REAL KEY
-----END RSA PRIVATE KEY-----
```

3.2. CTRLD

In a production environment, the CTRLD binary starts with default parameters. This service of CTRLD is called rtbrick-ctrld.

To see the default parameters and therefore also the files where configurations are stored, the easiest way to figure out is to do a ctrld -help.

To see the actual installed CTRLD version we can use ctrld -version.

CRTLD has in essence 3 important configuration files:

- /etc/rtbrick/ctrld/config.json: Configuration for the ctrld
- /etc/rtbrick/ctrld/policy.json: Role Based Access Control policy file.
- /var/lib/lxc/<container-name>/element.config: Element configuration file per container.

3.2.1. config.json

This section describes the main configuration file of CTRLD. This file can be changed via API, if it is changed on the file system CTRLD has to be restarted.

/etc/rtbrick/ctrld/config.json example

```
{
  "rbms_enable": true,
  "rbms_host": "http://10.200.32.48",
  "rbms_authorization_header": "Bearer THIS IS NOT A REAL KEY",
  "rbms_heart_beat_interval": 10,
  "graylog_enable": true,
  "graylog_url": "http://10.200.32.49:12201/gelf",
  "graylog_heart_beat_interval": 10,
  "graylog_endpoints": [
    {
      "name": "ztp",
      "url": "http://192.168.202.46:12201/gelf"
    }
  ],
  "auth_enabled": false
}
```

Table 2. /etc/rtbrick/ctrld/config.json format

Name	Type	Description
rbms_enable	bool	To enable all RBMS outgoing messages rbms_host
rbms_host	string	RBMS base url e.g.: http://192.168.202.44:9009
rbms_authorization_header	string	RBMS Authorization Header is set to all calls which are outgoing to RBMS
rbms_heart_beat_interval	int	RBMS heartbeat Interval in seconds (0 means deactivated)
graylog_enable	bool	To Enable all Graylog outgoing messages
graylog_url	string	Graylog url e.g. http://127.0.0.1:12201/gelf
graylog_heart_beat_interval	string	Graylog heartbeat Interval in seconds (0 means deactivated)
graylog_severity_level	string	Graylog min LogLevel that will be forwarded (default Notice: 5)
graylog_endpoints	GraylogEndpoints allows to specify multiple graylog endpoints by name. If a log to a specific endpoint is requested and the endpoint is not available, the log is send to the default graylog endpoint.	
auth_enabled	bool	To Enable the authorization and authentication

Name	Type	Description
name	string	Logical name of the endpoint e.g.: ztp for ztp messages.
url	string	Graylog url e.g. http://127.0.0.1:12201/gelf
disable	string	Disables this endpoint.
severity_level	string	Graylog min LogLevel that will be forwarded (default Notice: 5)

If the default endpoint is disabled, but the specific one is enabled than the message to the specific endpoint will be sent.

If the default endpoint is enabled, but the specific one is disabled than the message to the specific endpoint will not be sent.

3.2.2. policy.json

This section shows the **r**ole **b**ased **a**ccess **c**ontrol (RBAC) configuration for CTRLD. This file can be changed via API, if it is changed on the file system CTRLD has to be restarted.

/etc/rtbrick/ctrld/policy.json example

```
{
  "permissions": [
    {"sub": "system", "obj": "/*", "act": ".*" },
    {"sub": "supervisor", "obj": "/*", "act": ".*" },
    {"sub": "operator", "obj": "/*", "act": ".*"},
    {"sub": "reader", "obj": "/*", "act": "GET"},
    {"sub": "reader", "obj":
"/api/v1/rbfs/elements/{element_name}/services/{service_name}/proxy/bds/table
/walk", "act": ".*"},
    {"sub": "reader", "obj":
"/api/v1/rbfs/elements/{element_name}/services/{service_name}/proxy/bds/objec
t/get", "act": ".*"}
  ]
}
```

Table 3. /etc/rtbrick/ctrld/policy.json format

Name	Type	Description
sub	string	Subjects means the role which has the permission. Here RegexMatch Function is used: a regular expression pattern matcher.
obj	string	Object is the REST endpoint. Here KeyMatch4 Function is used: KeyMatch4 determines whether key1 matches the pattern of key2 (similar to RESTful path), key2 can contain a * and other patterns: • "/foo/bar" matches "/foo/*" • "/resource1" matches "/{resource}" • "/parent/123/child/123" matches "/parent/{id}/child/{id}" • "/parent/123/child/456" does not match "/parent/{id}/child/{id}"
act	string	And Action is the HTTP Method. Here RegexMatch Function is used: a regular expression pattern matcher.

So the example rules mean:

- The user with the role system is allowed to access all rest endpoints, and act on them with all HTTP methods.
- The user with the role reader is allowed to access all rest endpoints, but can only call the HTTP GET method.
- All authenticated users are allowed to access the proxy endpoint with all HTTP methods.The user with the role system is allowed to access all rest endpoints, and act on them with all HTTP methods.

3.2.3. element.config

This section shows the element.config which can be created per container. This files allows to redefine the element name, so the name can defer from the container name.

/var/lib/lxc/<container-name>/element.config example

```
{
  "element_name": "l1.pod1.blr",
  "pod_name": "blr",
  "ztp_enabled": false
}
```

Table 4. /var/lib/lxc/<container-name>/element.config format

Name	Type	Description
element_name	string	Name of the Element (Default container name)
pod_name	string	Name of the POD
ztp_enabled	bool	If enabled the ztp post process starts after the switch changes to operational state up. It's recommended to set this to false. In that case only the initial installation or reinstallation will trigger that process.

4. Business Events

APIGWD and CTRLD send different GELF messages about status changes or progress of processes to a GELF endpoint.

Table 5. GELF message format

Name	Type	Mandatory	Description
Default Message Fields			
version	String	Yes	The GELF message format version. Default value: 1.1
host	String	Yes	The hostname assigned via DHCP to the management interface. Defaults to the management IP address if no hostname is assigned.
level	int	Yes	Message Severity. See Table-1.
timestamp	float	Yes	Unix epoch time in second with optional fraction of milliseconds.
short_message	String	Yes	Problem message.
full_message	String	No	Detailed problem description.
_daemon	String	Yes	Name of the daemon.
_log_module	String	Yes	The module name identifies the component that created the log record. It allows segregating log records into different streams. Each stream can apply different processing rules and also be processed by different organizational units of the network operator.
_log_event	String	Yes	The log event identifies the log message template in the log configuration. The log event simplifies to find where in the system the log record was created. The log event should be succinct and typically conveys a unique reason code. In addition, the log event should be a reference that can be looked up in the product troubleshooting guide.

Name	Type	Mandatory	Description
_serial_number	String	Yes	The serial number of the switch. This allows tracking hardware replacements, even if the element name remains the same. Empty if not available.
_rtb_image_version	String	No	ONL Image Version that is installed on the switch that reports this message.
ZTP Message Fields			
_config_name	String	No	Exposes the loaded configuration name. Only set when a configuration file was processed or an attempt to process the file failed (e.g. 404 Not Found response from the HTTP server while attempting to load the configuration)
_config_sha1	String	No	Exposes the SHA1 checksum of the loaded configuration. Only set when the HTTP server returned a configuration.
_operational_state	String	No	Exposes the operational state of the element.
Request Message Fields			
_rid	String	No	Request ID, either X-Request-ID or new generated
_user_name	String	No	User name out of the access token
_user_subject	String	No	User subject out of the access token
_received_time	String	No	Time when the requested arrived
_method	String	No	HTTP method
_url	String	No	HTTP url
proto	String	No	HTTP protocol
_remote_ip	String	No	HTTP remote ip address
Service State Message Fields			
_service_name	String	No	Service name
_service_operational_state	String	No	Operational Service

Name	Type	Mandatory	Description
_service_startup_time	Number	No	Service startup time in unix epoch time, the number of seconds elapsed since January 1, 1970 UTC.
_service_down_flap_time	Number	No	Last down flap time in unix epoch time, the number of seconds elapsed since January 1, 1970 UTC.
_service_down_flap_counter	Number	No	Last down flap time in unix epoch time, the number of seconds elapsed since January 1, 1970 UTC.
_service_restarted	String	No	Restart is set to true if service_startup_time was changed.

Table 6. Level Descriptions as in RFC 5424

Level	Name	Comment
0	**Emergency**	System is unusable
1	**Alert**	Action must be taken immediately
2	**Critical**	Critical conditions
3	**Error**	Error conditions
4	**Warning**	Warning conditions
5	**Notice**	Normal but significant condition
6	**Informational**	Informational messages
7	**Debug**	Debug-level messages

GELF sample message

```
{
  "_config_name": "ctrld",
  "_config_sha1": "f1e06ef1e53becde6f8baf2b2fafe7dc9c36f6f0",
  "_daemon": "ctrld",
  "_element_name": "leaf01",
  "_log_event": "ZTP0011I",
  "_log_module": "ztp",
  "_serial_number": "591654XK1902037",
  "host": "leaf01",
  "level": 6,
  "short_message": "ztp ctrld config set",
  "timestamp": 1588382356.000511,
  "version": "1.1"
}
```

Table 7. Event Types

Graylog Instance	severity	log_module	log_event	description
ztp	Notice	ztp	ZTP0011I	ztp ctrld config set
ztp	Warn	ztp	ZTP0012W	ztp ctrld config not provided
ztp	Alert	ztp	ZTP0013E	ztp ctrld config not set
ztp	Notice	ztp	ZTP0021I	ztp startup config set
ztp	Warn	ztp	ZTP0022W	ztp startup config not provided
ztp	Alert	ztp	ZTP0023E	ztp startup config not set
ztp	Notice	ztp	ZTP0031I	ztp element config set
ztp	Warn	ztp	ZTP0032W	ztp element config not provided
ztp	Alert	ztp	ZTP0033E	ztp element config not set
ztp	Notice	ztp	ZTP0041I	ztp ctrld rbac config set
ztp	Warn	ztp	ZTP0042W	ztp ctrld rbac config not provided
ztp	Alert	ztp	ZTP0043E	ztp ctrld rbac config not set
ztp	Notice	ztp	ZTP0051I	ztp tls config set
ztp	Warn	ztp	ZTP0052W	ztp tls config not provided
ztp	Alert	ztp	ZTP0053E	ztp tls config not set
ztp	Notice	ztp	ZTP0061I	ztp accessjwks config set
ztp	Warn	ztp	ZTP0062W	ztp accessjwks config not provided
ztp	Alert	ztp	ZTP0063E	ztp accessjwks config not set
ztp	Notice	ztp	ZTP0071I	ztp apigwd config set
ztp	Warn	ztp	ZTP0072W	ztp apigwd config not provided
ztp	Alert	ztp	ZTP0073E	ztp apigwd config not set
ztp	Notice	ztp	ZTP1000I	ztp process finished
security	Warn	security	SEC0001W	access forbidden
security	Warn	security	SEC0002W	access invalid rtb token
security	Warn	security	SEC0003W	access invalid access token
security	Warn	security	SEC0004W	not able to download remote keys
security	Warn	security	SEC0005W	not able to download remote pem
security	Warn	security	SEC0006W	request rate limited (this message is also rate limited, and can be controlled in the apiwd config)
element	Notice	element	HTB0001	heartbeat with the operational_state

Graylog Instance	severity	log_module	log_event	description
element	Notice	element	STA0001	element state change
element	Notice	element	STA0021	service up
element	Error	element	STA0022	service unexpected down
element	Notice	element	STA0023	service expected down
element	Notice	element	STA0003	ready for service

5. Related Documentation

/ONIE/	The ONIE documentation outlines the DHCP options supported for image discovery. https://opencomputeproject.github.io/onie/design-spec/discovery.html
/ZTP/	The Zero-Touch Provisioning Guide outlines the current configuration discovery process. https://documents.rtbrick.com/21_11_1/ztp/ztp_guide_online.html
/SEC/	The Secure the Management Plane guide gives a detailed insight on this topic. https://documents.rtbrick.com/21_11_1/secmgmt/secmgmt_guide_online.html
/RADIUS/	The RADIUS attribute matrix provides an overview of the supported RADIUS attributes including a reference to the RFC that defines the message attribute. To obtain this document, contact your customer support team.
/CTRLD/	The CTRLD API describes all CTRLD REST API endpoints in detail. https://documents.rtbrick.com/index_api.html
/RESTCONF/	The RESTCONF API reference describes all configuration API endpoints. https://documents.rtbrick.com/index_api.html
/GELF/	The Graylog Extended Log Format (GELF) is a log format, this document outlines the fundamentals. To obtain this document, contact your customer support team.