Automation

Software Developers Kit for the FOS RESTConf API

3 April 2023

Contents

PREFACE

This document is not a Brocade controlled document.

This document is intended for programmers who will be writing their own modules to interface with Brocade fibre channel switches.

Topics covered in this document:

- How to access Brocade directors and switches through the REST API using Python
- How to correlate data from different resources and turning it into valuable business data.
- How to use examples and libraries posted to https://github.com/jconsoli

For additional detail, consult the Fabric OS REST API Reference Guide.

Author's Note

This began as examples on how to perform a few basic functions using the FOS API. As I needed something for my own work or to support a customer I added to the scripts maintained on https://github.com/jconsoli. This evolved in two different ways:

Simple Examples	This began as a driver with a few simple examples on how to do stuff using the driver. Some actions in the API require multiple steps so I added library modules to keep programming simple. For example, there are multiple steps involved with creating a logical switch but most programmers just want to call a method with a few basic parameters and a list of ports so I created brcdapi/switch.py. Often, I needed to do something using the API so I added a command line interface. For example, clearing port statistics is a common action so I added an interface to specify a range of ports or '*' for all ports in brocade-rest-api-applications/port_config.py. To support a range of ports or '*' as input, I also needed to read what ports were in the switch
Complex Examples	The first thing I needed was a report. To put a useful report together, I needed a way to

RESOURCES

Broadcom Communities:

Education

All Brocade education is offered at no charge. Recommended courses:

- Introduction to Fabric OS Introduction (API-220) – General overview of the REST API and Ansible automation topics
- Fabric OS REST Implementation (REST-320) – In depth discussion on the Fabric OS REST API
- Introduction to the Brocade RESTCONF API", API 200-WBT

github

https://github.com/brocade	Even if you are not using PyFOS, this is a good place to get the Yang models.
https://github.com/jconsoli	Where the libraries described herein are placed.
https://github.com/chipcopper	A good site for Anisble resources.

ENVIRONMENT SETUP

The discussions in this section are simple and easy for those not familiar with server administrative functions. They are not limitations.

Requirements

- FOS 8.2.1c or higher
  - Several operations are not exposed in FOS 8.2.1c
  - FOS v9.1.1a or higher is recommended
- Python 3.6 or higher

Installing Python

Download the Python interpreter:

https://www.python.org/downloads

Remember to check the box in the installer to set the path to Python.

You are going to need to add to the library, folder "Lib", so you need to find the folder and make note of it or add a short-cut to it on your desk top.

The “AppData” folder is typically hidden. File Explorer is typically defaulted to not show hidden files so you will need to either change this setting or manually enter that address. For example:

C:\Users\your-name\AppData\Local\Programs\Python\Python311\Lib

You can also just search for “Python” from the root directory.

If you are a developer, keep in mind that most IDE such as PyCharm do not analyze files in the Lib folder so you will need to develop or at least check-in scripts for the Lib folder in your user space.

You will also note that there is a "Scripts" folder which will contain pip*.exe files. This is not where you will be adding any of the sample scripts. All sample scripts and any scripts you are developing not intended for the Lib folder should be in your user space. This is the “Documents” folder in Windows. If you are manually updating PATH, in addition to including the path to the Python executable, a path to the scripts folder is also necessary. Specifically, the pip installer is located here which you will use to install additional libraries.

Additional Libraries

The sample scripts require libraries that may not be included with the standard Python installation. The easiest way to install the libraries is to use the pip installer. The URLs for the libraries are predefined in the pip installer so all you need to do is copy and paste the test from the “Install” column in a DOS or shell window. The latest version will be installed automatically. This means you will need to have internet access. The standard set of libraries included with the Python installation may change in the future. At the time this document was written, the following additional libraries were needed:

Library	Install	Additional Information
jdcal	pip install jdcal	https://pypi.org/project/jdcal
et_xmlfile	pip install et_xmlfile	https://pypi.org/project/et-xmlfile
*cryptography	pip install cryptography	https://pypi.org/project/cryptography
urllib3	pip install urllib3	https://pypi.org/project/urllib3

openpyxl	pip install openpyxl	https://pypi.org/project/openpyxl

*Note: The cryptography library is required for scripts that evaluate or modify security certificates: certs_eval.py and certs_get.py. Additional software may be required to install it. If the install fails, the error message will articulate what additional software is required. If you don't plan on working with security certificates you can skip installing this library.

Additional FOS Specific Python Libraries

In addition to the standard Python libraries, there are two libraries specifically for FOS that are required for the samples. A distribution package was not prepared for these libraries so you will need to download them and simply move them to your Lib folder.

Libraries to download:

https://github.com/jconsoli/brcdapi
https://github.com/jconsoli/brcddb

Navigate to the "Code" pull-down menu and select "Download zip". Note that github appends "-master" or "-main" to the folder name. Before adding brcddb or brcdapi to the Lib folder you will need to remove it so that the resulting folder names are "brcdapi" and "brcddb".

For Linux environment you will need to set the executable attribute, `-x`, using the `chmod` command on all of the files in "brcddb" and "brcdapi". Python also needs to create and write to a folder named "__pycache__" so you will also need to make sure the write, `-w`, attribute is set on the folders.

Script Work Space & Samples

You should create a folder in your user workspace where you plan to keep scripts. This is the "Documents" folder in DOS. Whether you create sub-folders for the applications (brocade-rest-api-applications) and examples (brocade-rest-api-examples) or put them all in the same folder is a personal choice. If you have not given any thought as to how to organize your scripts, just create a folder named "scripts". You can always reorganize it later. Open a command shell (referred to as DOS in Windows environments) and cd to the directory you just created.

Samples to download:

https://github.com/jconsoli/brocade-rest-api-examples
https://github.com/jconsoli/brocade-rest-api-applications

Shebang & Encoding

The “shebang” line is the first line in every script. It provides information about where the Python environment. The next line is always the encoding line. The operating system uses this to determine how to interpret the file contents

DOS (Windows)

By default, this is how the shebang line and encoding line in the samples are. If you are working with Windows this is informational only. You do not need to make any modifications to the files. It may be possible to work with a local Python environment in some Windows configurations but if you don’t know the difference is, keep it simple and use the shebang line below.

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
```

Linux, Linux Variants

```
#!/usr/local/bin/python
# -*- coding: utf-8 -*-
```

zOS (Mainframe)

Note that you will need to convert these files to EBCDIC

```
#!/usr/local/bin/python
# -*- coding: ebcdic -*-
```

Security & Certificates

In FOS 8.2.x, the HTTPS certificate is empty by default. This means all access to the API is via HTTP. FOS 9.x defaults to HTTPS on new installations but depending on the version of FOS running on your switch, it may still be HTTP after a code upgrade.

Most environments use a simple self-signed certificate which is all that is discussed in this document. Before making any configuration changes on production switches, you should discuss the security method with your organization’s security team.

See STTP status codes 404 and 301 in HTTP Status for additional information.

To check to see if a certificate exists:

FOS Command:

```
seccertmgmt show -all
```

Sample output:

```
sshprivatekey:
  Does not Exist

ssh public keys available for users:
  None

Certificate Files:
-------------------------------------------------------------------------
Protocol  Client CA   Server CA   SW       CSR      PVT Key  Passphrase
-------------------------------------------------------------------------
FCAP      Empty       NA          Empty    Empty    Empty    Empty
RADIUS    Empty       Empty       Empty    Empty    Empty    NA
LDAP      Empty       Empty       Empty    Empty    Empty    NA
SYSLOG    Empty       Empty       Empty    Empty    Empty    NA
HTTPS     NA          Empty       Exist    Empty    Exist    NA
KAFKA     NA          Empty       NA       NA       NA       NA
ASC       NA          Empty       NA       NA       NA       NA
```

By default, the highlighted text above will be "Empty" which means an HTTP login is required. "Exist", as in the example above, indicates that a security certificate exists.

To auto-create a self-signed certificate via the FOS CLI:

```
seccertmgmt generate -cert https -keysize 2048 -hash sha256
```

Environment Validation

From a shell or DOS prompt:

```
py lib_check.py
```

Since some people keep brocade-rest-api-applications and examples brocade-rest-api-examples separate, for convenience the same lib_check.py script is in both locations.

The library check is performed in reverse hierarchical order meaning that libraries that are dependent on other libraries will fail if the dependent library is not found. Rectify all not found libraries in order before attempting to resolve other libraries that failed.

Login Test

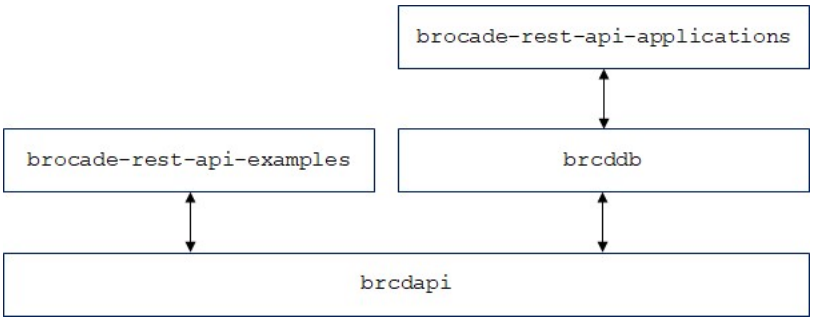
The login test script below is in brocade-rest-api-examples. The only additional library required is brcdapi.

```
python login.py -h
```

You will need to provide login credentials. The `-h` option will tell you how.

DRIVERS AND SAMPLES

Architecture



brcdapi	This is a driver. It is the single interface to the API. It formats data to send and receive from the switch, retries requests if the switch is busy, and simplifies some basic functions. Used by brocade-rest-api-examples and brocade-rest-api-applications.
brocade-rest-api-examples	Initially intended as programming examples only for basic operations. As the need arose to use some of these scripts for useful purposes, a user interface was added. For example, port_config.py has an example on how to clear port statistics. Since periodically clearing port statistics is a common operation, adding a user interface permitted using these scripts to perform useful work. See subsection “brocade-rest-api-examples”
brcddb	This is a hierarchical relational database. It is used by the applications to resolve complex dereferencing. For example, the zone analysis in report.py uses this to determine what zones ports participate in.
brocade-rest-api-applications	Scripts to perform common data center operations. See subsection “brocade-rest-api-applications”

The general approach is to read data, then so something with the data. There are four types of scripts:

- Capture data only
- Operate on captured data only
- Operate on either captured data or by interacting with the switch
- Capture data as need. Operating on previously captured data is not an option.

Common Command Line Options

-h	Prints help information and exits.
-log	Unless –nl is specified, a log file is always created. The name of the file includes a time stamp when the file was created. The operand is the folder where the log files are created. If not specified, log files are created in the same folder where the script was run from.
-sup	All output to the screen is written to the log file with an option to echo what is being written to the log to the screen. Setting this option prevents the echo to the screen. This is useful for batch processing and machine execution where printing to the screen is undesirable.
-nl	A log file is not created when this option is set. Output is still printed to the screen.
-dm	Only available with applications that read Excel Workbooks for input. Allows workbooks to be read into a table and that table can be written to a JSON formatted file. The primary reason for doing this is so that other scripts can launch these scripts by creating the JSON file rather than have to create a workbook. The file name is the same as the Excel file name except the extension is ".json" instead of ".xlsx". By default, if the corresponding .json file exists and it is more recent than the .xlsx file, the JSON file is read instead of the Excel file.
-ip	IP address of switch. Only used in scripts that do not get login credentials from a workbook but do need switch access.
-id	User id. See –ip.
-pw	Password. See –ip.
-s	HTTP access mode. Typically "–s self" for HTTPS with a self signed certificate. See –ip.

-d	Debug mode. Typically only used by script developers. All data sent to or received from the switch is formatted and printed to the log.

brocade-rest-api-examples

api_get_examples	Programmer examples only.
app_config	Enable/disable the HTTPS interface, Enable/disable the Rest interface, set the maximum number of Rest sessions, and set the keep alive time.
certs_eval	Determines the expiration date of all security certificates. Also has options to create a certificate signing request (CSR), add certificates, and delete certificates.
certs_get	Gets and displays certificates without evaluating them.
cli_poll_to_api	Programming examples only.
ge_stats_to_db	Programming examples only. Illustrates how GE port statistics can be polled and written to a database.
login_test	Used to validate network connectivity. Logs into a switch and then immediately logs out.
maps_clear	Clears the MAPS dashboard alerts.
port_config	Performs common port actions such as clearing all stats, enable/disable a group of ports, set ports to their default state, etc.
stats_to_db	Programming examples only. Illustrates how port statistics can be polled and written to a database.

brocade-rest-api-applications

capture	Captures data for a single chassis. Outputs a JSON formatted brcddb database entry for the chassis.
chassis_restore	Makes a best effort attempt to restore a chassis from data collected from a previous capture. Parameters allow the user to select certain attributes to restore.

cli_zone	Intended as zoning examples but customers with existing CLI scripts find it convienent to use this script because it performs checking, has a test mode only, and is much faster than feeding an SSH session with CLI commands.
combine	Combines multiple brcddb database files (output of capture) into a single project.
compare_report	Compares two brcddb database files. May be a project containing output for multiple chassis.
lib_check	Validates that all required libraries are accessible and up to date.
lib_validate	Provides detailed information about library requirements and search paths for a specific script.
login_test	Performs a Rest login/logout. Used to validate access to the Rest interface.
multi_capture	Launches multiple capture sessions to collect data for multiple chassis in parallel. After collection is complete it uses the combine module to combine all data into a single project. Optionally generates a report.
port_count	Generates an Excel report with port count usage for all switches in a brcddb project..
report	Generates an Excel report similar to SAN Health
search	Searches a brcddb project for specific API items. Requires specific knowledge of the API branches and leaves. Search parameters can be simple or complex regex and wild card searching.
stats_c	Captures all port statistics for a specific logical switch. Output is a brcddb database.
stats_g	Converts the output of stats_c to human readable Excel with the option to create graphs based on certain ports and statistical parameters.
switch_config	Creates or modifies logical switches using parameters read from an Excel workbook.
zone_config_x	Although capable of creating aliases, zones, and zone configurations from a workbook, the primary use is to test zoning changes outside of a zone change control window.

zone_merge	Merges and tests zone merge capabilities from multiple fabrics.
zone_restore	Restores the zone configuration from a previous capture. The target switch does not have to be the same switch the capture is from.

FOR PROGRAMMERS

Converting CLI and Perl to Python

CLI & Perl

There is not a direct correlation between the API and the CLI. An additional document published by Broadcom is expected in 2022 with a list of CLI commands and the corresponding URI(s). In the meantime, the module below includes a partial list of URIs and the corresponding CLI command so that it can be searched for either a URI leaf or FOS command.

List Comprehension

A list comprehension is a short hand way of writing a loop that builds a list (array). For example, to extract all the numbers from a list of strings and convert them to a list of integers:

```
input_str_list = ['5', 'a', '234', 'dog']

# The long way:

number_list = list()
for buf in input_str_list:
  if buf.isnumeric():
    number_list.append(int(buf))

# The short way (using list comprehension syntax):

number_list = [int(buf) for buf in input_str_list if \
  buf.isnumeric()]

# In either case, number_list = [5, 234]
```

Table Driven Software

Calling a method (subroutine) from a table is a very common practice in C++ programming. Although Python and other scripting languages support doing this, it's not common. It is

especially useful in Python because Python does not have a case statement so this approach saves a lot of `if, elif, ...` Keep in mind that when you do this, the same code is calling different methods so each method must have the same input parameters.

Note that in the table below the method name is not followed by parenthesis. A method name followed by parenthesis tells the Python interpreter to call the method and load the return value into the table. To load the pointer to the method in the table, enter the method name without parenthesis. For example, these methods convert a number, float or int, with a K, M, or G multiplier to their full value as a string:

```
def _action_k(value):
  return str(value * 1000)

def _action_m(value):
  return str(value * 1000000)

def _action_g(value):
  return str(value * 1000000000)

# Set up the table

_action_table = dict(K=_action_k, M=_action_m, G=_action_g)

# This code:

value = 12
for buf in ['K', 'M', 'G']:
  str_value = _action_table[buf](value)
  print(str(value) + buf + ' = ' + str_value

# Outputs:

12K = 12000
12M = 12000000
12G = 12000000000
```

try/except

Python has a feature that allows you to try something and if it doesn't work, do something else. This saves considerable time and code validating data structures.

running vs operations

There are three basic branches:

null	These are login, logout, and module version branches that are at the root level.
running	Anything in the running branch is executed immediately. The response is returned upon completion of the operation.
operations	Anything in the operations branch is passed to another process in FOS which executes the request. In some instances the process handling the front end API request waits for the secondary process to complete such that there appears to be no difference between a running response and an operations response. The primary value with requests in the operations branch is that instead of waiting for the secondary process to complete execution of the request, it can return good status with a response body indicating that the request has not completed. The response body contains a token that is used to poll the switch for status. This feature is useful for long running requests such as a firmware download because it avoids an HTTP connection timeout while waiting for status. **Important Notes**: • As of FOS v9.1.1, not all requests that require a long time to execute were implemented. • As of FOS v9.1.1, requests in the operations branch support POST and OPTIONS only. This means a GET request in the running branch is required to read the values. • The body (contents) of requests do not include the final leaf as is the case with requests in the running branch.

Connection Headers

See the first 4 lines of `brcdapi_rest._api_request()` for an example on how to build JSON connection headers

Throttling

Although there is no need to be concerned with throttling requests, carefully consider the work load. Work load does not necessarily translate to CPU utilization; however, it can effect HTTP connection timeouts. In the “Important API Notes” section, read sub-section “Important Work Load Considerations” for additional detail.

Flow control is on a per chassis basis, not a logical switch basis. All parameters discussed in the table below can be set concurrently and take effect immediately but since the parameters are applied to each request at the time the request was made, these changes effectively occur with the next request.

Idle time	This is the length of time to wait (sleep) after receiving a 503 status, service unavailable response. It can be set from 3 - 2147483647 seconds. By default, it is set to 3 seconds. Example – change the idle time to 6 seconds: `mgmtapp --config –idletime 6` Recommendation: There is no need to change the idle time to anything other than the default of 3 seconds; however, your code should add 1 second to this time. It was discovered in testing that with just a 3 second sleep before re-driving the request that FOS nearly always returned another 503 status. The frequency of immediately getting a 503 status after re-driving the I/O diminished as this time was incremented. At 4 seconds, the re-driven request was always accepted. In `brcdapi.brcdapi_rest`, the application sleep time is defined by `_SVC_UNAVAIL_WAIT`.
Maximum number of sessions	The maximum number of RESTConf API sessions can be 3 – 10. By default, it is set to 3. This number is not effected by Network Advisor or SANnav. Example - Change the maximum number of sessions to 10: `mgmtapp --config –maxrestsession 10` Recommendation: Unless there is a need for more than 3 sessions, leave the maximum number of sessions to the default of 3.
Sample time	The FOS throttling algorithm permits a certain number of requests within a time frame window before returning 503 status. The sample time defines this window. The sample window can be set to anything between 30 and 2147483647 seconds. By default, it is set to 30 seconds. Example - set the sample time to 60 seconds. `mgmtapp --config  -sampletime 60`

	Recommendation: There is no known reason at this time to ever change the sample time to anything other than the default. Should throttling be required, the better parameter to adjust is the number of sample requests.
Sample request count	IN FOS 9.1 and above, this parameter is no longer user configurable. This is the number of requests that can be made within the sample time before a 503 status is returned. It can be set from 30 – 2147483647. By default, it is set to 30. Example – set the request count to the maximum: `mgmtapp --config  -samplerequest 2147483647` Recommendation: By default, all of the API throttling setting are set to the lowest setting. This doesn't make sense for the sample request count though as this effectively limits the number of request to just one per second. For several zoning changes, for example, this could result in an excessively long period of time to complete the required changes. Testing has shown there are no CPU load concerns handling multiple requests. The recommendation therefore is to set the sample request count to the maximum of 2147483647. As a practical matter, it's not possible to complete that many requests in a 30 sec window so this effectively disables request throttling.
Determine current settings	There is no ability to read or set these parameters via the API in FOS 8.x. To read them via the CLI: `mgmtapp --show`

HTTP Connection Timeout

The RESTConf model does not have a specified standard as to how long the processing time should be before a response is expected but there is a guideline of 20 seconds. For zoning changes and most GET methods, 20 seconds is adequate but there are requests that will take longer. The brcdapi driver uses a 60 sec connection timeout. Some requests still need to be broken into multiple requests. It is not possible to break up all requests such that a 20 sec timeout will work.

If the timeout is too short, HTTP connect lib raises an exception but the session is not terminated on the FOS switch. This means FOS will continue to process the request. If the request was to make a change, the only way to determine if the request completed successfully when the connection times out is to read the resource and compare it against expected values.

This also means that your script must logout after a timeout. Using the standard Python libraries, an exception is raised when this occurs. The sample applications all use the Python Try/Except feature to ensure a logout for any exception.

Recommendation: Testing revealed that too many requests timed out at 15 and then 20 seconds to articulate them all. The timeout value in `brcdapi.brcdapi_rest` is defined by `_TIMEOUT` and was set to 60 seconds. This was adequate for all requests although in some cases the number of actions to take within a single request had to be reduced.

HTTP Status Codes

HTTP Status	Description
200	As per standards, FOS uses this status to indicate successfully execution of a request. It is also used by brcddb and brcdapi libraries to indicate success when simulating successful request responses.
202	Not used. FOS 8.2.1c processes requests until complete. It does not time the request. Requests with many embedded requests can result in a timeout before status is received by the client so it is up to the programmer to limit the work load within a single request to avoid a timeout. Although this status has not been implemented, it is discussed herein because many programmers are familiar with it and a request to implement this status has been submitted.
204	The brcddb.apps.zone library uses this in test mode or bulk mode in response to a request to save (commit) the zoning transaction buffer when there are no outstanding zoning transactions.
301	Reason: “Moved Permanently” is returned by FOS when an HTTPS certificate is defined by you attempted to login as HTTP.
400	This is used as a general purpose catch all in FOS for just about any error. The “Reason” and “err-msg” may provide additional information. When “The Fabric is busy” is in the reason field, this indicates that the switch is too busy to process the request (typically 502 in most HTTP implementations). This typically occurs when making requests immediately after enabling a switch or after power on.

	The `brcdapi.brcdapi_rest` library sleeps for the time specified in `_FABRIC_BUSY_WAIT` (10 seconds as of this writing). The maximum number of retries is `_MAX_RETRIES` (5 as of this writing). This wait time and maximum number of retries is excessively long but given how infrequent this situation is, no testing was done to find more efficient parameters.
403	This is used as per standards to indicated unauthorized access such as: • Login with invalid credentials • Attempt to read/write URIs the user is not authorized to access
404	Reason: "Not found". This is synthetically generated in brcdapi.fos_auth.login() when the standard Python library conn.request() raises an exception. This happens when the IP address is unreachable or HTTPS was used before a certificate was generated in the switch. If you can ping the address,
408	Not used by FOS. Used by the brcdapi libraries when a requests times out.
409	Not used by FOS. The brcddb.apps.zone library uses this whenever there is a conflict. For example, if you attempt to create a peer zone with the same WWN as a member and a principal member.
500	Used by FOS, brcdapi libraries, and brcddb libraries when a programming error is encountered. When using the brcddb or brcdapi libraries, additional information and a stack trace is written to the log. Search the log for 'Exception call with msg:'
501	"Not Implemented" – Consistent with HTTP standards, this error code is used to indicate when a method or URI is not supported by FOS. The typical error response is not returned. Instead, the error code is in the status of a normal response.
503	When "Service Unavailable" is in the reason field, this indicates that too many requests were received (typically status429). When the recommended throttling settings are set, it's highly unlikely that you will ever see this status but your code should be designed to re-drive the request after waiting the appropriate time. See "Idle time" in the Throttling section.

Request/Response Size

The maximum length of a request passed to the API cannot exceed 10 Mbytes. As a practical matter, there is no useful request or response that takes up this much space.

Empty Response Errors

Some requests return an error when an empty list is expected. Future versions of FOS are expected to change this behavior so that empty lists are always returned when the request completes successfully but has no content to return.

Since applications in most layered architectures will exit with an error code when a protocol layer error is past upstream, it makes sense to convert these errors to empty list responses with good status at the interface layer.

Search for “empty list” in this document for additional detail.

Important Work Load Considerations

Although work load is not CPU intensive enough to warrant concerns over CPU utilization, to perform certain actions the CPU must wait on sub-systems to complete those actions. There is no difference in time to complete actions regardless of where the action is initiated from. If work load is started from multiple sources that need access to the same resources, time to completion will be elongated.

The work load discussion in this section is focused on the timeout considerations relative to work load.

Switch Configuration

With an HTTP Connect timeout of 60 seconds, timeouts usually occurred when creating logical switches with 4 ports from two or more sessions. Timeouts always occurred when trying to configure 3 switches, each with 4 ports, at the same time. Timeouts usually occurred when creating just one switch with 4 ports while there was a `supportshow` running from an SSH session. With no other workload running, logical switch creation takes about 22 seconds. The same was observed for switch deletion. Adding ports takes about 600 msec. per port.

Recommendation: To allow for a little room, the recommended HTTP connection timeout is 60 seconds when creating a logical switch. Logical switches should not be created with any ports. When adding ports to the switch, using the same HTTP connection timeout, limit the number of ports being added in a single request to 35.

Limitations

With FOS 8.2.1c, most data capture and configuration capabilities of FOS required to maintain a typical database of switch information and to perform common provisioning tasks have been exposed through the API. A complete list of equivalent API to CLI commands is unavailable.

Below is a partial list of limitations for common features. Future versions of FOS are expected to address these limitations.

- FICON
  - RNID data and security (SCC) policies are not available.
- Some of the detailed trouble shooting information is not available.
- The ability to enable/disable XISLs is not available but the state can be read
- MAPS rules, configurations, groups, and status is complete but reading the dashboards was not fully implemented.
- RAS Log (errdump)
  - Configuration information about the RAS log is available but only the syslog is present in 'message-text'.
- Audit log
  - Only configuration information is available.

Port Statistics

FOS polls statistics from the ASIC every 5 seconds in Gen5 and every 2 seconds in Gen6 and Gen7. The statistics are stored in a memory cache. A GET of `brocade-interface/fibrechannel-statistics` returns the cached statistics. Keep in mind that due to other network and CPU task scheduling priorities, polling of statistics does not occur at precise time intervals so with a short poll cycle, it's possible to poll for the same statistics twice. The timestamp `brocade-interface/fibrechannel-statistics/time-refreshed` can be used to determine if the statistics are from a different samples. This time stamp was introduced in FOS 9.1. Note that `time-generated` is a timestamp of when the request was made, not a timestamp associated with the port statistics in the response.

Empty Lists

All errors that indicate an empty list are intercepted by the `brcdapi` libraries and returned as an empty list with good status.

There is a plan to return empty lists as empty list with good status in future versions of FOS. This should have no effect on the `brcdapi` library.

Background

Below FOS 8.2.1c, there were several different error messages for an empty list. For example, requesting the ports in a logical switch with no ports would return an error. A development project to clean these up and return an empty list with good status.

As of FOS 8.2.1c, most empty lists were returned as an empty list but some still returned:

```
status=404
reason='Not Found'
```

FDMI list, for either hba or port, will return:

```
Status: 400
Reason: Bad Request
```

HA status (specific to bladed switches) are returned as noted below. This is considered an empty list by `brcdapi` because it is assumed that data gathering scripts are written at a layer before the chassis configuration is understood. This is true of the capture utility in `brcddb` where data is just read and stored in a container.

```
Status: 400
Reason: Bad Request
Error msg: Not supported on this platform
```

When changing a resource with PATCH, no change is either returned as an empty list of changes or the error below. The `brcdapi` library converts all of these errors to an empty list.

```
Status: 400
Reason: Bad Request
Error msg: No Change in Configuration Value
```

List of Length 1

Prior to FOS v8.2.1b, lists of dictionaries of length 1 were often returned as a dictionary (`dict` in Python) instead of a list of dictionaries with one entry. With FOS v8.2.1b and above, all lists of length 1 should be returned as a list.

brcddb List Handling

Rather than write excessive code to discern when a dictionary should be converted to a list, as a work around, the `brcdapi.gen_util.convert_to_list()` is always called by methods within the `brcddb` libraries and sample applications when operating on lists. When a list is passed to `convert_to_list()`, the same list is returned so there is no need to modify

working code. Programmers beginning with FOS 8.2.1b should not have to be concerned with lists of length 1 but do not need to modify existing code that call `convert_to_list()`.

Note that since the call to `convert_to_list()` was never removed from any of the `brcddb` libraries, no test was performed to validate that all lists of 1 are in fact returned as a list of 1.

Zoning

All zoning transactions take place in a zoning transaction buffer on the switch where the zoning changes are being made. The zone transactions are not sent to the fabric until the zoning changes are saved. A fabric is one or more switches connected together. Once zoning transactions are saved, the zoning transaction buffer is cleared. Any time the zoning transaction buffer is not clear, zoning transactions are outstanding.

A checksum is required by the API of the existing zone database to make zone changes. The checksum is validated before pushing any changes to the fabric.

Although it appears to the programmer as a single operation, enabling a zone configuration in FOS is actually a two-step process. It puts the action to enable the configuration in the transaction buffer and then pushes the zoning transaction buffer to the fabric.

You should familiarize yourself with the zoning rules as discussed in the FOS Administrators Guide before attempting to script zoning changes.

Remote Media

Usually, `brocade-media/media-rdp/remote-media-xxx`, is `None` whenever there is no remote media information available. Similarly, the associated thresholds, `remote-media-xxx-alert`, are typically `None`; however, in some instances the thresholds have been observed to be 0. These issues are typical of older optics.

It has been observed in some cases that SFPs return 0 for all remote SFP data. These observations were made when the optic in the attached device was 8G or slower (older optics) and therefore assumed to be an issue limited to older optics. Broadcom restricts support of SFPs to ensure certain quality and standards are met but Broadcom has no control the type of SFPs used in the attached equipment.

The data associated with `remote-media-current` and `remote-media-temperature` doesn't make sense. There is either a defect in the code or documentation. The data is posted as received in the report generated by `report.py` but no best practice checking is done against these two leaves.

The `brcddb/brcddb_bp.py` module ignores remote current, remote temperature, and all remote SFP data if `remote-media-voltage` is 0 values when checking for alarm or warning threshold violations.

Hung Login Sessions

When working with the API, chances are you'll have a bug or two that crashes your script. If this happens before you logout, your login session will remain active. Since there are a limited number of application logins supported, you may need to terminate those sessions.

In the brcddb libraries, try/except (written in Python) is always used between login and logout to ensure a code bug doesn't prevent a logout. This makes it a little more challenging to get exception information though.

From the CLI, there are two ways to terminate login sessions:

Reboot the Switch

`fastboot` – This command reboots the switch. While fast, easy, and convenient, if you are developing on shared switches, execution of this command may make you unpopular with your colleagues.

Manually Disable Application Sessions

Display the sessions Pre-FOS 9.1:

```
apploginhistory --show
```

Display the sessions FOS 9.1 and above:

```
mgmtapp --showsessions
```

The application logins are last but the session keys are very long and not practical to retype. Issue this command with logging enabled so you can copy and paste the session key.

```
mgmtapp --terminate
```

Using json.dumps()

The `json.dumps()` library requires properly formatted JSON dictionaries and lists; however, Booleans require the first letter to be uppercase as is the Python standard. Do not use closing commas in dictionaries or lists passed to `json.dumps()`. Commas should only be used to separate multiple items. It is common for programmers to include a comma at the end of lists because Python interprets a list of dictionaries with just one entry and no comma as a `dict`.

Adding the comma forces the library to interpret the single `dict` in a list case as a list. Regardless of the number of entries, if the last element of a list is followed by a comma, it may cause unintended formatting resulting in errors being returned from the API.

Bad Example

```
content = {'fibrechannel-logical-switch': {
            'fabric-id': fid,
        }
    },
```

Good Example

```
content = {'fibrechannel-logical-switch': {
            'fabric-id': fid
        }
    }
```

Required API Parameters

It is not necessary to include empty lists or defaults when adding or modifying resources. In some cases, empty lists are only permitted in PATCH to clear an existing list so the general advice, except for zoning, is add items to a resource with POST. The highlighted items below are either default or empty and therefore this:

```
content = {
    'fibrechannel-logical-switch': {
        'fabric-id': fid,
        'base-switch-enabled': 0,
        'logical-isl-enabled': 0,
        'ficon-mode-enabled': 0,
        'port-member-list': {
            'port-member': ['0/0', '0/1', '0/2', '0/3']
        },
        'ge-port-member-list': {'port-member': []}
    }
}
```

Is better represented as:

```
content = {
    'fibrechannel-logical-switch': {
        'fabric-id': fid,
        'logical-isl-enabled': 0,
        'port-member-list': {
            'port-member': ['0/0', '0/1', '0/2', '0/3']
        }
```

```
        }
    }
```

Note that setting `base-switch-enabled` and `ficon-mode-enabled` to the default on a newly created switch, albeit superfluous, is supported but an empty port list is not supported. In the example above, `ge-port-member-list` is empty. FOS would return an error if this was sent to a switch.

Also worth noting in this example is that `logical-isl-enabled` should not be confused with allow XISL enable. The parameter for setting XISL usage was not yet implemented in FOS 8.2.1c.

When disabled, the `logical-isl-enabled` parameter causes the switch to not permit any E-Port connections. It can be enabled with POST or PATCH but disabling it is only supported using POST during initial logical switch creation. It is useful because switches are often cabled and configured prior to being ready to be put into production. This features allows all ports to be enabled without having to filter out E-Ports from the list of ports to enable or having to worry about enabling an E-Port accidentally due to a cable miss-plug.

FICON

RNID data and some of the security settings are not supported in FOS 8.x. FOS v9.0 exposed all information and control required for FICON except the ability to set a port address. Support for setting a port address, equivalent to `portaddress --bind`, was introduced in FOS 9.1.

RESTConf Request Methods

Overview

In most cases, it is an array (Python `list`) element that will be modified, added, or deleted. In most cases, the resource being modified is an array. Each array element is usually a dictionary (Python `dict`). All arrays of dictionaries have one key/value pair that is used as an identifier for the element. Typically this key is ‘name’.

Determining if a request method was applicable to the URI proper or a dictionary element in an array identified by the aforementioned key was not fully tested. It appears that `PUT` operates on the URI proper while `PATCH` operates on the identified element.

Use the examples in `api_direct` to determine how to build URIs, content, and determine the best method to use.

DELETE

Since `PATCH` cannot be used to add to a resource, no testing was done with `DELETE`. The rationale for not testing `DELETE` was that methods designed to modify a resource need to do both add and delete and since adding to a resource can only be done by interrogating the resource

and then replacing it, the modify methods may as well use the same logic for deleting elements of a resource.

GET

Standard `GET` behavior. Use `GET` for all read operations.

PATCH

In all cases tested, when the resource is an array of dictionaries, `PATCH` operates on just the identified element. Only the identified dictionary is modified by `PATCH`, not the entire array. Partial updates on the individual element are not possible. The entire element must be replaced. In this regard, `PATCH` behaves as PUT as far as that specific element is concerned. All `brcddb.apps` and `api_direct` examples use `PATCH` to replace resources.

In some cases, `PATCH` will return status = 400, reason = 'Bad Request', with a message 'No Change in Configuration Value' when there is nothing to change. In other cases, a good status, 200 – 299, is returned with an empty response.

To ensure consistent behavior and alleviate the need for application layer code to process protocol layer errors, `brcdapi_rest.api_request()` checks for this error and returns an object with an empty response. Although the actual status is not modified, `brcdapi_rest.is_error()` returns `False`, effectively making it look like the request succeeded..

POST

Use `POST` to add new resources. In some cases, `POST` can be used to replace a resource as described with `PATCH`. In all of the libraries described herein, `PATCH` was used exclusively to replace a resource.

PUT

Unlike `PATCH`, `PUT` does not operate on just the identified array element. Limited testing was done with `PUT` but what testing was done indicates that `PUT` operates on the URI only. When attempting to use `PUT` to operate on an element the same way testing with `PATCH` was done, an error was returned. No additional investigation was done to determine what appeared to be a discrepancy between how `PUT` and `PATCH` behave.

The only dynamic resources controlled external to the fabric is zoning. There is a special URI to clear the zone database so there was never a need to use `PUT` to completely replace a resource for any of the libraries or applications discussed herein.

Methods With json.loads()

Requests other than `GET` do not always return something to read when there is good status. When there is good status, 200 – 299, but nothing to read, reason = 'No Content', `json.loads()` will raise an exception for some resources. `brcdapi.brcdapi_rest.api_request()` executes requests inside try/except so as to clear the exception and return an object with the status so that application layer code does not need to address protocol layer errors.

Logical Switches

There must always be a default switch. If virtual fabrics is not enabled, the chassis is effectively the default switch. The default switch cannot be deleted from a chassis with virtual fabrics enabled; however, the default switch FID can be changed.

Ports are not deleted from logical switches, they are moved to another logical switch. Ports can be moved implicitly with PATCH and DELETE or explicitly with POST. When using DELETE or omitting a port in a port list using PATCH, ports are moved to the default switch. An error is returned if you attempt to delete a port from the default switch. From the discussion in 'PATCH and POST', note that PATCH can only be used when ports already exist in a switch so PATCH is not useful in code that is port count agnostic.

All ports must be moved to another logical switch, typically the default switch, prior to deleting a logical switch. Special configurations must be removed from a port before it can be moved to another logical switch.

Ports do not have to be disabled before moving them. They are automatically disabled when moving from one logical switch to another. If the intent is to have ports enabled after moving them, they must be explicitly enabled.

Creating & Changing The Zone Database

Although any portion of the zone database can be modified, it's easier to simply clear out the entire database and replace with a consolidated list of what the final zoning configuration should look like. It is also much faster when making several zoning changes due to request throttling. Examples of how to do this are in `brcddb.api.zone`.

Don't forget that although FOS allows the defined zone configuration of the effective zone configuration to be modified, it cannot be deleted.

Remember that all zoning transactions take place in a separate transaction buffer and are not implemented or distributed to the fabric until activated.

Zoning on Newly Created Fabrics or Enabled Switches

When a logical switch is initially created, it is not ready for zoning configuration changes. Attempting to make zoning changes before a switch is ready will return an error 400 with message 'The fabric is busy, try again later'.

There is no simple way through the API to determine if a switch is ready for zoning changes. The `brcdapi` library sleeps for 10 seconds before re-driving the request whenever a fabric busy message is received.

XISLs

XISLs have a lower fibre channel cost and therefore direct ISLs will not be used if a base switch is present. The default is to allow XISL use; however, if there is no base switch defined in the chassis the XISL does not exist and therefore this setting is moot.

FOS Rules

All FOS rules apply to requests sent to the API. The FOS Admin Guide and FOS Command Reference Guide are useful resources for determining FOS rules.

Ordered Actions

Don't forget that the parameters in a Python `dict` are organized by how Python decides to create the hash, which may not be in the order you added key/value pairs to the dictionary. If, for example, you want to change a switch parameter that requires the switch to be disabled, `'enabled-state=3'` must appear first in the content. Use the Python library `ordereddict` to ensure items are in the order intended. Many of these instances were considered defects and may have been addresses since they were first discovered in FOS 8.2.1c.

Tip: *Little testing was done using an ordered dictionary. To avoid any potential conflicts, make separate requests when the order of actions is important.*

Lists

This problem was discovered in FOS 8.2.1c and may no longer be an issue.

Number of list elements	Return

0	This is rectified with the brcdapi library.
	
	In 8.2.x, empty lists are returned as an error:
	
	`status=404`
	`reason='Not Found'.`
	
	Since most programmers handle errors of this type at a lower layer and want the ability to loop on each element of a `list` regardless of how many

	elements are in the `list`, `brcdapi.brcdapi_rest.py` converts these errors to empty lists. In FOS v8.2.1a and below, several empty lists returned different error codes. Most of the empty lists cases in v8.2.1b have consistent behavior in that they return a status of 404, but there were still some returning a status of 400. Note that when modifying resources that are not changing, this is essentially the same empty list issue. When a resource is changed, most responses include a summary of changes in a list format. If a resource didn't actually change, that list is empty. Some requests return an error instead of an empty list.
1	Lists of dictionaries of length 1 are always returned as a `dict` in FOS 8.2.1a and below. This should be fixed in FOS 8.2.1b. Since `brcdapi` and `brcddb` are FOS version agnostic and the only way to tell if something in a request should be a `list` or `dict` is by context, they are not converted in `brcdapi.brcdapi_rest.py`. ***Note:*** *Given the challenges of determining when a `dict` should really be a `list`, all of the libraries and sample applications always call `brcddb.util.util.convert_to_list()` as simple means to perform the convert. Although these should all be fixed in 8.2.1b, the code to convert the lists was left in place.*
> 1	Lists of 2 or more are always returned as lists.

Unique Keys

When creating a key-value database to store data returned from the switch in objects it will be necessary to create unique keys.

Since the API references certain resources based on unique URIs and leaves, using the URI and leaves for database keys is an ideal way to create unique database keys. Furthermore, the URIs and leaves are well documented in the Rest API guide so it is easy to correlate the keys to the documentation.

Another approach is to parse the data from the API into individual objects. The brcddb libraries use a hybrid approach. The database contains objects for major items (such as switches, chassis, fabrics, zoning, etc.) parsed from requests and therefore needed a unique key not present in the URI and leaves. Data associated with those objects use the URI and leaf names as unique keys within the objects. For example, a switch object is referenced using the switch WWN as the key but all information associated with a switches is stored using the leaves as the keys within the switch object.

Table of Unique Keys Used in brcddb

Object	Comments
Project	The project object is a collection of switch, chassis, and fabric objects. Key: User defined Object where stored: Application code
Chassis	All chassis also have a unique serial number and license-id. As of FOS v8.2.1c, the license-id was not available in the API. Key: Chassis WWN Object where stored: Project
Switch	Although switch WWNs are unique, as discussed in the SAN tips section, a switch WWN can change if the corresponding switch fabric ID (FID) was deleted and re-created. Switches are also assigned an address which is subject to the same potential to be changed but otherwise unique. Since a disabled switch is not in a fabric, the brcddb libraries store the switch object in the project object. Only a list of the switch WWNs (key) are stored in the fabric object. Key: Switch WWN Object where stored: Project
Fabric	If the principal switch is not specifically configured in a fabric, it may change during a fabric rebuild. FOS, and therefore API requests that associate a resource with a fabric, uses the principal switch WWN as the unique fabric identifier. Since other parameters can change as a result of a fabric rebuild, it is recommended to delete all database entries from a fabric and rebuild the fabric. Following this recommendation means the principal switch WWN can be used as the unique fabric identifier. Key: Principal switch WWN. Object where stored: Project
Port	Although physical ports have a unique WWN, virtual ports do not have a WWN. All ports, including virtual ports have a unique index but the physical GE ports do not have an index. All ports have a unique slot/port. Note that a slot number is always returned, even on fixed port switches, except as noted below. The slot is always 0 on a fixed port switch. All ports within a switch, and chassis, have a unique port number. Exceptions:

	• Port names returned with `brocade-media/media-rdp` also have a media type as well. Something like: m/s/p so m/ is removed to associate the media with a port in the brcddb libraries • Ports in MAPS messages for fixed port switches are returned as the port number only. Prepend '0/' to MAPS messages when no slot is given in the response. Key: slot/port Object where stored: Switch
login	Includes name server registration, FDMI HBA registration, and FDMI port registration. FOS does not permit duplicate WWNs in a fabric. Theoretically, all WWNs should be unique but as virtualized servers and storage proliferate, duplicate WWNs are becoming a problem. Login WWNs therefore can only be considered unique within a fabric. Note that FICON devices typically do not register with the name server or FDMI database. Instead, RNID data is used. When RNID data becomes available, it likely will be stored in the login object. Key: Login WWN Object where stored: Fabric
Zoning	Includes aliases, zones, and zone configurations. FOS does not allow duplicate zone configuration, zone, or alias names so they are always unique within a fabric. The brcddb libraries store all zoning objects with the fabric object. Key: Alias, zone, or zone configuration name Object where stored: Fabric

Resource-ID inconsistencies

There are only a few inconsistencies in the API. As long as you are aware of them in advance, they should not cause any programming problems.

wwn & user-friendly-name

'wwn' and 'user-friendly-name' are context sensitive. For example, 'wwn' returned with a switch resource is the WWN of the switch while 'wwn' for a login is the login WWN. The URI is always unique. In some cases, usually when the context is not clear, the resource ID may be prefixed with an indicator. For example, the user friendly name for a switch can be 'user-friendly-name' or 'switch-user-friendly-name'.

Although the brcddb database stores data as it was received, methods are provided in the brcddb libraries to resolve them.

Time Stamps

Timestamps have various formats. Use brcdapi.gen_util.date_to_epoch() to convert to a common timestamp.

Port References

Ports can be referenced by several means. Too many to discuss herein. The brcddb libraries resolves all port references.

MAPS

Thresholds

Thresholds must be integers. All thresholds, including numeric thresholds, must be sent as a string.

Quarantine Actions

A GET request for brocade-maps/rule will always return a value for un-quarantine-timeout and un-quarantine-clear whether a quarantine action is supported for the monitoring system or not. The monitoring system leaf is monitoring-system.

When sending a POST or PATCH request for a monitoring system that does not support a quarantine action, an error will be returned if un-quarantine-timeout or un-quarantine-clear is included in the request even when SDDQ is not included in the actions.

Actions for temperature thresholds, SFP_TEMP, do not support a quarantine action. As a practical matter, there is never a reason to quarantine an SFP due to physical media issues.

Zoning Changes

Just as with the CLI, zoning changes are made in a temporary workspace and not distributed to the fabric until saved. Making complex zoning changes can require several API requests to complete. Except in `api_config_examples.py`, the `brcddb` libraries and applications completely clear out the zone database and replace it with a PATCH of the complete final zoning configuration. This approach is not only faster because it reduces the number of requests, it eliminates the need to understand all the nuances of zoning changes. Zoning changes in `brcddb.api.zone.replace_zoning()` is always these 4 requests:

- Get a check sum
- Clear the database (which is aborted if the next step fails)
- Send a whole new zone configuration (inclusive of aliases, zones, and zone configurations)
- Save the changes.

A transaction is completed (contents of the zoning transaction buffer pushed to the fabric) whenever the transaction is saved. Zoning transactions can be saved either explicitly or implicitly.

Explicit zone configuration save:

```
content = {'effective-configuration': {
        'cfg-action': 1,
        'checksum': checksum
    }
}
```

Implicit (via a zone configuration enable) zone configuration save:

```
content = {
    'effective-configuration': {
        'cfg-name': zonecfg,
        'checksum': checksum
    }
}
```

ISL Trunks

ISL trunk information can be obtained from:

```
brocade-fibrechannel-trunk/trunk?vf-id=xx

{'trunk': [{'deskew': 5,
            'destination-port': 384,
            'group': 1,
            'master': True,
            'neighbor-domain-id': 2,
            'neighbor-switch-name': 'X6_Core',
            'neighbor-wwn': '10:00:c4:f5:7c:2d:a6:20',
            'source-port': 384},
           {'deskew': 5,
            'destination-port': 388,
            'group': 1,
            'master': False,
            'neighbor-domain-id': 2,
            'neighbor-switch-name': 'X6_Core',
            'neighbor-wwn': '10:00:c4:f5:7c:2d:a6:20',
            'source-port': 388},
```

Notable leafs:

`destination-port`	This is the destination port index. It corresponds to `default-index` in `brocade-interface/fibrechannel?vf-id=xx`.
`group`	Each trunk group is assigned a unique group number. Use this to associate trunk members.
`neighbor-wwn`	This is the WWN of the destination switch. Note that `neighbor-wwn` has a different meaning in this context than with brocade-`interface/fibrechannel?vf-id=xx` .
`source-port`	This is the source port index. See `destination-port`.

For those familiar with the CLI, this returns the same information that the `trunkshow` command returns.

Health Checking

There are 3 different types of health checking:

Basic H/W Checking

Health status of every FRU is available. With the exception of SFPs, all FRUs are chassis requests:

```
brocade-chassis/ha-status
brocade-fru/blade
brocade-fru/fan
brocade-fru/power-supply
```

In most cases, use `operational-state` to check for the health. Specific values are dependent on FRU associated with the resource. Check the Rest API Guide for details.

SFP status is a logical switch level resource. The health is in `brocade-interface/fibrechannel/physical-state`. Note that if the port is disabled, power is cut from the SFP so the `physical-state` may not be accurate. Data retrieved from `brocade-media/media-rdp` may not be accurate either. Although you can check to determine if the port is enabled, there are some circumstances whereby a port may be configured enabled but not yet actually enabled. As a practical matter, it's highly unlikely that you'll poll a port before it's ready but a good practice is to check. The status is in the response to `brocade-interface/fibrechannel/operational-status` (2 = enabled). Note that `enabled-state` is deprecated in v8.2.1b and replaced with `operational-status`.

To check the health of a port:

A simple fault/no-fault check can be performed by checking `brocade-interface/fibrechannel /operational-status`. If the value of `operational-status` is 5, the port is faulted. For a more specific reason for the fault, check `brocade-interface/fibrechannel/physical-state` (new in v8.2.1b). Values that indicate a port fault are:

- `faulty`
- `no_module`
  - Not an error if no SFP is plugged but typically customers disable ports that have no SFP plugged so this may actually be a fault.
  - `operational-status` will be 3 (offline), not 5 (faulty).
- `laser_flt`
- `port_flt`
- `hard_flt`
- `diag_flt`
- `mod_inv`
- `mod_val`
- `no_sigdet` (see notes with `no_module`)

Search the FOS Command Reference Guide on the above faults for a detailed explanation.

Logical Checking (MAPS)

Although all MAPS information, including the dashboard, is available through the API, as of this writing, adequate documentation to decipher the MAPS data was unavailable.

Best Practice Checking

Additional checking, such as balanced ISLs and enabled ports with no login, are common best practice checks but there isn't a single URI that returns this type of checking.

Methods for common best practice checking to be updated in this document at a future date.

Matching Switches to Physical Chassis

The response to `'brocade-switch/fibrechannel-switch'` does not return `'chassis-wwn'` but `'brocade-fabric/fabric-switch'` does contain `'chassis-wwn'`. You will need to match the WWN of the switch in question to the switch WWN in the fabric data to find the chassis WWN. There is an example of this in `brcddb.api.interface.get_chassis()`

Matching Name Server and FDMI to Physical Switch Ports

Although logins are to a fabric, only the routing information is shared throughout the fabric. Name server and FDMI is not disturbed throughout the fabric. It is stored on the individual switches where the login occurred.

The name server information includes a reference back to the physical switch port by the WWN for the switch port but the FDMI information does not. The `name`, which is effectively the key for the resource in the FOS API, is the login WWN. Except for AMP trunk ports and SIM ports (ports in debug mode), this WWN appears in the neighbor, see "Port Configuration" sub-section later in this section.

The `brcddb` libraries, specifically `brcddb.util.util.build_login_port_map()`, builds a map of logins to physical port spinning through the `neighbor` data for each port and looking those WWNs up in the fabric.

Remember that NPIV enabled devices will have a base login plus one or more logical WWNs logged in. In the list of port neighbor WWNs and the HBA port list in the HDMI HBA, the base WWN plus all logical logins are present. Each logical login will have its own entry in the name server and HDMI for the port.

Note: *'name-server-device-type' will be 'Physical Unknown(initiator/target)' for the base login and 'NPIV Initiator' or 'NPIV Target' for the logical logins; however, if there are no logical logins, there is no way to tell a base NPIV port from another port who's device type is unknown. As an expedient, brcddb.util.util.login_to_port_map() assumes the first login for the port is the base login if there are more than one login to a port and assumes all remaining logins are the logical logins. In other places, it is assumed that the first entry in fibrechannel/neighbor is the base login.*

Alternatively, the fibre channel address can be used. The fibre channel address is `fcid-hex` in the port data and `port-id` in the name server data. Using the fibre channel address gets around the SIM port issue but not NPIV logins. If you use the FC address, you will have to mask off the portion of the ALPA used for NPIV logins to match it to a port. The portion of the ALPA used for NPIV logins depends on the addressing mode.

Special Ports (AMP & SIM)

SIM Ports

The `neighbor` list of the port configuration data is empty when a port is in simulation mode (SIM-Port). The simulation port does register with the name server, `brocade-name-server/fibrechannel-name-server`. The `'port-properties'` member is `'SIM Port'`.

Ports Connected to AMP

AMP units are special switches whose connections are similar to ISLs. Just as with Remote E-Ports, the `neighbor` list of the port configuration data will contain the WWN of the AMP port and they do not present FDMI data. The trunk master registers with the name server, `brocade-name-server/fibrechannel-name-server`. The `'port-properties'` member is `'I/O Analytics Port'`. The other members of the trunk register with the name server but `'port-properties'` is not present. Also, the trunk master is AE-Port in `fibrechannel/port-type` but the other members are U-Port.

Note: *Since multiple uplinks to an AMP unit from the same switch and SIM ports are unusual occurrences, the brcddb libraries do not account for them. The report generated from report.py will not distinguish these logins from any other and generate an error message.*

Remote E-Port

The `neighbor` list of an E-Port contains the WWN of the remote switch E-Port, see `wwn` in the "Port Configuration" subsection. Remember that switches do not register with the name server or provide any FDMI data.

Working With Fabrics

Not all fabric related data is shared with all individual switches in a fabric. Some data, such as firmware level, switch name, and switch domain ID is returned with certain fabric URIs but name server and FDMI data is not. Since resources are gathered from specific switches, programs combining data into a single fabric view will need to accommodate duplicate information for some data and be able to combine data for other resources.

Except for the firmware level in v8.2.1a and below, all duplicate fabric data is returned with the same key exactly the same regardless of the switch the data was obtained from. In v8.2.1b and above, the representation of the firmware level is consistent.

Routing tables are shared fabric wide but individual login data registered with the name server and FDMI database is only available in the switch where the login occurred. Programmers will have to combine this data to form a fabric wide perspective.

Name Server

Response for `'/rest/running/brocade-name-server/fibrechannel-name-server'`, sub-key `'fibrechannel-name-server'`:

`port-name`	This is the WWN of the attached device port. Matches `'PortName'` in in the FOS command `'nsshow -r'` output. This is the WWN used for zoning. This is also the WWN in `neighbor`. Note that `port-name` is not returned for a port in SIM mode or connected to AMP and the corresponding port data will not have any neighbor data.
`fabric-port-name`	This is the WWN of the physical switch port.

FDMI HBA

Response for `'/rest/running/brocade-fdmi/hba'`:

`hba-port-list`	Look in sub-key `'wwn'`. This is a list of all the port WWNs on the HBA. Matches `'port-name'` in the name server and `'port-name'` in the FDMI port.

FDMI Port

Response for `'/rest/running/brocade-fdmi/port'`:

fabric-name	WWN of the principal fabric switch.
port-id	Fibre channel address for the login.
port-name	The login WWN. This matches port-name in the name server which will match one of the WWNs in the FDMI HBA port list as well as one of the neighbor WWNs in the fabric switch port if the port is not in SIM mode. Note that port-name is not returned for a port in SIM mode or connected to AMP and the corresponding port data will not have any neighbor data.

Port Configuration

Response for '/rest/running/brocade-interface/fibrechannel':

fcid-hex	Fibre channel address of the port. Matches the base portion of the port-id in the FDMI.
neighbor	Sub key is 'wwn' which contains a list of all the WWNs logged into this port. These WWNs match 'port-name' in the name server.
wwn	WWN of the physical switch port. Matches 'fabric-port-name' in the name server

General SAN Tips

Disabled Switches

A disabled switch is not in a fabric and therefore will not appear in any fabric requests. This doesn't happen often after a switch is put into production but it's not unusual for customers to disable a switch prior to being deployed.

The Default Switch

From the factory, the default switch is FID 128 and is always the same as the chassis WWN. Although not often, customers do on occasion create a new logical switch, make it the default switch, and then delete the FID 128 logical switch. This means the WWN of the default switch may not be the same as the chassis WWN and the default FID may not be 128. Furthermore, the customer could have changed the default FID back to 128 in which case, the WWN of the default switch will not be the same as the chassis WWN.

An OEM may re-purposes a chassis in inventory as new if it was never shipped to a customer. Although rare, don't assume that what shipped from an OEM is at the Brocade default factory settings.

Fabric IDs

FID checking can be disabled. When FID checking is enabled, only switches with the same FID are allowed to join in a fabric together. The default FID checking state is enabled (FID must match in all switches in the fabric) and likely never changed in production environments. Since all logical switch partitions in a chassis must have a unique FID, turning off FID checking can be useful in lab environments where a single chassis can be carved up into multiple logical switches and then connected together to form a fabric of multiple switches within the same chassis.

It's highly unlikely that scripts need to be concerned with multiple FIDs in the same fabric; however, since most scripting will be developed in lab environments the intent of this note is to make sure programmers are aware.

License ID and Chassis ID

Prior to Gen6, the chassis WWN and license ID were always the same. The chassis WWN and license ID may not be the same with Gen6 switches. This license ID is not available through the API with FOS v8.2.1b and below.

Aliases

Alias rules:

- Names are case sensitive
- Names must begin with a letter
    - Can be followed by any combination of letters, numbers, an underscore, and a dash.
- Cannot be more than 64 characters..
- Associated with the fabric, not a specific zone configuration
- Adding, deleting, and modifying zone aliases are part of a zoning transaction
- Changing an alias that is already used in a defined zone is permitted.
    - Keep in mind that all aliases were resolved to WWNs when the zone configuration was activated. Changing a WWN associated with an alias of a defined zone that is active will not change the effective zone until that zone configuration is re-enabled.

Typically, most organizations use one alias per WWN, do not allow multiple aliases for the same WWN, and always zone by alias. These are common organizational rules. They are not Fibre Channel restrictions and therefore not FOS restrictions. An alias can be used for a group of WWNs, a domain, index, or a combination of WWNs and domain, index pairs. A WWN can have any number of aliases.

Since multiple members of an alias are permitted, they are returned from the API as a list.

The brcddb libraries handle the situation whereby an alias can have multiple members. The method, `brcddb.brcddb_fabric.zone_analysis()`; however, will only recommend an alias with a single WWN member if a WWN was used in the zone definition when an alias for the WWN is available.

Zones

The same naming rules for aliases apply to zone rules.

Although permitted, zones containing a mix of d,i and WWN members is a bad practice. This is because session based enforcement is used. Session based zoning requires the CPU to resolve the zone and therefore adds considerable time to frame handling within the fabric. When generating zone reports, this should be a warning. All other zoning enforcement is performed in hardware with programmable arrays.

Domain, index (d,i) zones are typically used for FICON (mainframe). Although rare in distributed environments, they are sometimes used to group disk mirroring ports in a single zone.

Zoning Sources & Effect on Zoning Changes

Zoning changes are made in a switches transaction buffer and are not pushed to other switches in the fabric until the changes are saved. Zoning changes not yet pushed to a fabric are considered an outstanding zoning transaction. Although only one zoning transaction can be outstanding per switch, zoning transactions can be outstanding on multiple switches in the same fabric. Therefore, there is a potential for zone changes to be overwritten when they are pushed to the fabric.

The FCS feature was designed to avoid the potential to overwrite zoning changes by designating one switch in the fabric for zoning changes. An error is returned any time zoning changes are attempting on a switch not designated for zoning changes. Although typically the principal switch, it can be any switch in the fabric. By default, FCS is not enforced.

As of FOS v8.2.1a, setting the FCS through the API was not available; however, all FOS rules are relevant to the API as well. If FCS is defined in the fabric, the API will return an error if zoning changes are attempted on a switch not designated for zoning changes.

Sources of Zone Changes

Command Line Interface (CLI)	A CLI session can be established from: • The maintenance port ○ Typically, only used for service which does not include zoning operations • A terminal telnet session • A script with a telnet session

Network Advisor and SANnav	A management system with a proprietary API
Target Driven	Target driven zoning is a fibre channel feature that allows targets to send zoning requests in-band. Target driven zones are peer zones. They cannot be modified via any other means and therefore not relevant to any zoning changes. Target driven zones are reported in API requests for zone information.
Rest API	Keep in mind that your script may not be the only script attempting zoning changes.

Port State, SFPs & QSFPs

When a port is enabled, all information about the SFP can be found in `brocade-media/media-rdp`.

In FOS 8.2.1b, the leaf `physical-state` was added to `brocade-interface/fibrechannel` which returns the port state as it is returned with the `portshow` command in FOS. Any physical state other than `No_Module` indicates that an SFP is present.

When an SFP is disabled, power to the SFP is turned off. A bus in the SFP allows FOS to determine if the SFP is present but no other information from an SFP can be obtained. When the port is disabled, the physical state is '`No_SigDet`'

The same is true for QSFPs; however, each individual port on a QSFP is reported in FOS as its own SFP. All information passed through the API is presented in the same manner. Based on physical form factor, it's easy for a SAN administrator to determine what four groups of SFPs belong to a single QSFP. Since the QSFP has a single serial number, programmatically using the API the easiest way to associate SFPs with a QSFP is by the serial number which is leaf `serial-number` in `brocade-media/media-rdp`

Know Your Organizations Rules & Conventions

Make sure you understand your organizations policies, especially for naming conventions. Do not assume they are always followed.

Creating Reports With Excel

This section has nothing to do with the API; however, a few notes about Excel are included since generating reports in Excel is very common.

The openpyxl library can be found here:

https://pypi.org/project/openpyxl/

Sheet Names & Links

As Workbooks become large, it's common to add a table of contents with links to other sheets. Since links are to sheet names with a cell reference, not the sheet index, it's important to know the sheet name rules and some not so well known Excel rules about creating hyper-links to sheets within the workbook.

Sheet Name & Link Rules:

- No more than 31 characters
- Can't contain ':' (so no WWNs in standard format). Most other special characters are not permitted either.
- The hyperlink formula doesn't work if there are spaces or dashes
    - A simple solution is to convert all non-alpha numeric characters and spaces to an underscore, '_'.
- Must be unique

WORKING WITH THE FOS REST API

Skip this section if you are using the `brcddb` libraries.

It is not always obvious how resources are related. For example, it's often useful to know the port number where a login occurred. The primary purpose of the `brcddb` libraries is to resolve these relationships so that it is easy and transparent to application programming.

PROGRAMMERS REFERENCE: SUPPLIED LIBRARIES & SAMPLE SCRIPTS

Each module has a header with a description. The library modules in brcdapi and brcddb also have a list of public methods in the header. Most of the library methods are used in examples or described in the "How To Do Stuff" section. Only a few methods with unique features are discussed in this section.

Overview

All libraries and sample scripts have a summary of public methods in the header at the beginning of the file.

From:

https://github.com/jconsoli

brcdapi	Typical driver added to the standard Python lib folder. Low level interface to the API. Handles basic session, connection, throttling, and basic protocol (such as retries). Also contains libraries to simplify common tasks that are not specific to the FOS API.
brcddb	Typically added to the standard Python lib folder. Intended as the next layer above `brcdapi`. This is a hierarchical relational database built entirely out of standard Python data structures. The database alleviates the need for programmers to determine how individual KPIs are related. For example, the login object has attributes so that the port, zones, switch, fabric, chassis can be determined from a single line of code. It also includes some useful utilities for searching and report generation.
brocade-rest-api-examples	Specific purpose sample scripts illustrating how to do something with minimal dependencies such as disabling a port using the `brcdapi` library.
brocade-rest-api-applications	Examples illustrating how to perform more complex tasks such as disabling all ports with no logins using the `brcddb` libraries to perform common functions. Scripts in here are often referenced in the "How To Do Stuff" section.
Documentation	This document is placed here. Check periodically for updates.

brcdapi

This is the driver layer that provides a single interface to the FOS API. It is intended to sit between any library or application communicating with the FOS API. Features of the library include:

- Login and logout
- Build connection headers
- Build complete URIs
- Process low level protocol status such as retries

- Clean up some defects in responses.
- Logging
- Two debug modes
- Utilities for common FOS specific operations
- Non FOS API Utilities
    - Read and write Excel Workbook
    - Common file operations
    - Generic utilitarian methods

All of the examples in `brocade-rest-api-examples` and `brocade-rest-api-applications` use this library.

log.py: Logging & Printing to STD_OUT

All of the sample scripts using the logging module log.py in brcdapi instead of using print().All data written to the log can optionally be echoed to STD_IO. There is also a global override so that all printing to STD_IO can be disabled.

The log is used as follows:

```
import brcdapi.log as brcdapi_log

brcdapi_log.open_log(log_file)
brcdapi_log.log(msg, echo_flag, force_flag)
brcdapi_log.exception(msg, echo_flag, force_flag)
```

`log_file`	The name of the log file. If None, a unique log file is created using the current time and date. If `open_log` is never called, then all calls to log or exception are echoed to STD_OUT if `echo_flag` is true but nothing is logged. If the log file was created, it is flushed from cache anytime `exception()` or `flush()` is called.
`msg`	Message to print to the log and/or STD_OUT. When calling exception, this message is preceded with a stack trace dump.
`echo_flag`	If True, print the `msg` to STD_IO. The default is False.
`force_flag`	If True, ignore the global suppress printing to STD_IO and, if `echo_flag` is also True, print `msg` to STD_IO. This is useful for when scripts are launched from other scripts and only the final message.

To set or clear the global suppress printing to STD_IO flag:

```
brcdapi_log.set_suppress_all()
brcdapi_log.clear_suppress_all()
```

brcdapi_rest.py: Single API Interface & Debug Mode

In addition to providing a single interface to the API, there are two debug modes:

Log activity detail	This mode can be programmatically enabled with: `brcdapi_rest.verbose_debug = True` It is set to False by default. When True, all data sent to the API and received from the API is printed to the log and STD_OUT using Python's `pprint()`.
Offline debugging	A write mode allows you to export GET response to a file. In read mode, subsequent GET requests are read from the file instead of attempting to GET the response from a switch. In read mode, a positive acknowledgment is return for login and logout requests without taking any action on the switch. Search for _DEBUG and _DEBUG_MODE in the module for details.

zone.py, port.py, switch.py: Zoning, Port, & Switch Libraries

Simplifies common actions. For example, to avoid HTTP connection timeouts ports must be added to a logical switch in multiple smaller groups of ports. The method `add_ports` () accepts a list of ports, breaks it up into smaller groups of ports, builds the content, and sends it to the API with a single method call.

brcddb

Any time the API returns an error that cannot be resolved or invalid parameters were passed to a built in application an alert is added to the project object. Only the alerts listed in the code sample below indicate a processing error occurred:

```
import brcddb.app_data.alert_tables as al

error_list = (
  al.ALERT_NUM.PROJ_FAILED_LOGIN,
  al.ALERT_NUM.PROJ_CHASSIS_API_ERROR,
  al.ALERT_NUM.PROJ_SWITCH_API_ERROR,
  al.ALERT_NUM.PROJ_PROGRAM_ERROR,
  al.ALERT_NUM.PROJ_USER_ERROR,
)
ml = [obj.fmt_msg() in proj_obj.r_alert_objects() if \
  obj.alert_num() in error_list]
if len(ml) > 0:
```

```
error_msg = '\n'.join(ml)
```

brcdapi

This is the driver layer that provides a single interface to the FOS API. It is intended to sit between any library or application communicating with the FOS API. Features of the library include:

- Login and logout
- Build connection headers
- Build complete URIs
- Process low level protocol status such as retries
- Clean up some defects in responses.
- Logging
- Two debug modes (GET only)
    - Store all responses to a file (JSON dump)
        - Uses a simple hash of the URI so reading the same data twice overwrites the previous file and it is not updated with requests that update a resource
    - Read all responses from previously stored files (JSON load)

brocade-rest-api-examples

Most of these are strictly samples intended to illustrate how to build the content for requests and how data returned from the API is formatted. The assumption is that programmers will copy and paste pieces of the code they need and modify to suite their own needs or execute the code in a development environment using breakpoints to examine data structures. They also illustrate how to interface to the `brcdapi` driver libraries

Since these are intended as programming examples, there are numerous comments throughout the code and the code is written in a more verbose style.

This library also includes some generic utilities not specific to FOS.

brcddb

This library is a simple hierarchical relational database with powerful built in utilities. It is a container for all collected data. It alleviates the need for programmers to filter and determine how to make relationships between various resources. This is useful because the API returns chunks of data that are not organized the way higher level applications normally use data.

Typically, this library doesn't do much for write operations so in most cases, scripts use brcdapi directly to make switch changes. Most scripts need to read and organize data before making

decisions on what changes to make so this library may be useful even if only writing scripts that make changes. Most automation tasks also read data back to validate the changes.

Important Note: *It is not immediately obvious how data is related as it is read from switches. Upon completion of data collection, typically obtained by calling brcddb.apps.capture.capture_all(), your applications must call:*

brcddb.util.util.build_all_cross_ref() – matches name server database and FDMI database with physical ports.

brcddb brcddb_fabric.zone_analysis() – Correlates the name server database with the zoning database.

Primary features are:

- Single interface to brcdapi
- Relational database. Objects return
    - The parent object
    - The port object associated with name server or FDMI data and vice versa
    - All zones effected by a port or login
    - And more
- Search utility
    - ReGex matching, ReGex searching, '*' and '?' wild card matching, numerical comparisons
    - Perform complex searches with simple tables. For example, return all ports with a login matching a certain name server parameter that logged in at 8G.
- Zone analysis. Checks for
    - Mixed server login speeds (capped at storage speed) in the same zone
    - Mixed storage login speeds in the same zone
    - Zone has less than 2 members
    - Peer zone doesn't have at least one principal and one regular member
    - Mixed d,i and WWN zones
    - WWN used in zone when there is an alias for it
    - Zone member in another fabric
    - Zone is not used in any zone configuration
    - Effective zone doesn't match defined zone
    - Zone member not found
    - Base port (HBA) of NPIV logins is in a zone
    - Maximum number of devices zoned to a device. See brcddb.app_data.bp_tables.MAX_ZONE_PARTICIPATION
    - Mix of WWN and alias in the same zone
    - Duplicate aliases
    - Alias is not used
    - Alias contains no members
    - All aliases used in zones actually exist
- Processing batch requests

    - Just pass it a list of URIs and the responses, if applicable, are added to `brcddb` database.
- Built in Excel reporting
    - Makes it easy to customize reports
    - Typically used during development to validate programming
    - Includes a table of best practice checking such as online ports with no connections, unbalanced ISLs, and much more
    - Built in `get_all()` method captures all data and adds it to the `brcddb` database
- Data Containers
    - Capture project data into a data container (JSON dump) and written out to a file
    - Use the previously captured data from the data container rather than capture new data real time.
        - Useful for working with read only data offline.
        - Make changes from previously captured data from container rather than real time
            - Copy zone data from one fabric to another
            - Use as a backup

There is an object for all major SAN areas. A detailed map of which resources is associated with which object is provided in Appendix A.

There is one project object which contains all objects for all major areas. For example: fabric, chassis, and switches are stored in the project object. Major objects contain objects of things specific to the object. For example, all zoning objects are attached to the fabric object.

The project object pointer is stored in all objects. All other object pointers are stored by the object owner only. Applicable keys are stored with each object. The library includes utility modules for complex database correlations and actions but most common correlations are performed with methods associated with the objects. This provides access from any object to other related objects. For example, `port_objects()` returns a list of all port objects in all switches from the project object but only the port objects with a switch from a switch object.

Understanding Parsing & Data Collection

Using the built application `brcddb.apps.capture.py` is a convenient way to capture all data. Due to throttling and time to collect data, the burden of capturing all data can add considerable unnecessary delay to specific purpose applications.

The easiest way to make a customized list of requests is to put them in a table and pass them to `brcddb.apps.interface.get_batch()`.See application `stats_and_login.py` for an example.

This method collects all chassis requests first, then FID (logical switch). Due to the way data is compartmentalized in different requests from FOS, in most cases, you will need the logical switch information. Your list of chassis URIs should include the following:

```
chassis_uris = (
    'brocade-fibrechannel-logical-switch/fibrechannel-logical-switch',
)
```

The first time `get_batch()` is called, if a chassis object does not already exist it will issue a `brocade-chassis/chassis` request and create the chassis object so there is never a need to add this to the list of chassis URIs. All responses for chassis requests are added to the chassis object. All switch level requests are added to objects as specified in the appendix: "BRCDDB URI TO OBJECT MAP". Any logical switch request not listed in the appendix is added to the switch object.

When using `get_batch()`, the preamble is prepended to the passed URI. For a logical switch URI, it also appends `'?vf_id=x'`. FOS doesn't look at any parameters on chassis level URIs so it does not return an error if appended with parameters. This means that if you accidentally add a chassis request to the list of logical switch requests, the response will be added to the switch object. If you pass a logical switch level URI without the `vf_id=` parameter (this would happen if you accidentally put a logical switch level request in the list of chassis request), FOS will return an error.

app_data

The intent of the `brcddb` libraries was to make them FOS version independent and not need to be edited for custom applications. The app_data folder can be placed anywhere your scripts have access to, but usually just the Python Lib folder.

alert_tables.py	Contains alert definitions. Tables can be referenced or copied and modified in your applications. Care needs to be taken when removing alerts. The recommended best practice is to add alerts but not remove any. Currently used by: report.py when calling `brcddb.brcddb_bp.best_practice()` `conv_to_peer_zone.py` to add zoning change information.
bp_tables.py	Any element of any resource can be tested and have an alert added regarding best practice violations. The tables can be referenced or copied and modified in your applications. Currently used by: `report.py`

report_tables.py	Contains several tables to define how reports are displayed. All known elements of URIs with FOS 8.2.1b were included. The tables can be referenced or copied to your applications. Currently used by:
	`report.py`
	`search.py`

Conversion Tables

Numbers are often used as indicators of status, hardware types, and operational states. Even if you plan on writing all of your own code, it may be useful to copy some of these conversion tables. Conversion tables can be found in:

- `brcddb.brcddb_chassis.py`
- `brcddb.brcddb_common.py`
- `brcddb.brcddb_switch.py`

Built in Applications (sub directory apps)

capture	Automatically determines logical switches defined in a chassis and captures all data that was exposed in the API with FOS 8.2.1c. and builds the container maintained by `brcddb`
report	Excel report generator
zone	Send zoning database in brcddb fabric object to a switch.

Applications

General

All applications, including built in applications in `brcddb.apps`, always use the content to define changes. This is because throttling is based on the number of requests, not the amount of work to do for each request.

For example, if you disable 180 ports, it will take over 3 minutes doing them one at a time. If you disable 180 ports in a single request, then all it takes is the networking time and processing time by FOS. Disabling ports doesn't require much processing time so disabling 180 ports typically takes under 2 seconds when sent in a single request.

capture.py

User interface to the built in application capture utility to capture the data necessary for the `brcddb` container from a single chassis. Automatically determines defined logical switches defined in a chassis and captures all logical switch data for all defined logical switches.

Captures all data that was exposed in the API with FOS 8.2.1c.

See also: `multi_capture.py` and `combine.py`

cli_zone.py

Reads a file of CLI zoning commands can converts to API calls or a `brcddb` object file.

Using the API as a replacement for an SSH CLI session isn't useful; however, since there are numerous CLI zoning scripts, this was a useful tool to test `brcddb.apps.zone.py`. It's also a useful example for those familiar with the CLI.

See also: `create_bulk_zone_json.py, patch_zoning.py`

combine.py

Combines multiple projects, typically the output of capture.py, into a single project object.

See also: `multi_capture.py` and `capture.py`

conv_to_peer_zone

Although this could be run stand-alone to convert zones to peer zones, it is intended as a practical example of how to write an application. This example illustrates how to interrogate fabric zoning, make zoning changes based on the current zoning, and SAN related things to think about beyond just coding to the API.

create_bulk_zone_json.py

Sample data structure for programs such as Ansible for use with `brcddb.apps.zone.py`.

disable_unused.py

Disables all ports with nothing connected.

multi_capture.py

Same as capture.py but accepts a list of switches to capture data from instead on an individual chassis.

See also: `capture.py`

patch_zoning.py

Stand-alone module to replace the zoning database in a fabric from a `brcddb` project object. This completely replaces the zone database so it's useful for bulk zoning changes such as restoring a zone database or copying a zone database from one fabric to another. Good example of how to use `brcddb.api_zone.replace_zoning()`.

Use `api_direct.zone_config` for examples on how to make incremental zoning changes.

report.py

Generates a SAN Health like report using the built in application report.

search.py

Examples only. Contains several code samples for generating a wide variety of searches commonly used by operation people. Includes Regex matching, Regex searching, wild card matching, and threshold comparisons.

stats_and_login.py

Collects port statistics and associates the port with name server data. Returns data in JSON format. Good example of how these libraries can be used with automation engines such as Ansible.

stats_c.py

Collects multiple samples of port statistics on a switch. For cumulative counters, which are most of the counters, the differences are calculated and stored so the changes can be evaluated. The output is a JSON dump of a Python dictionary stored in a plain text file.

stats_g.py

Converts the output of stats_c.py to and Excel Workbook

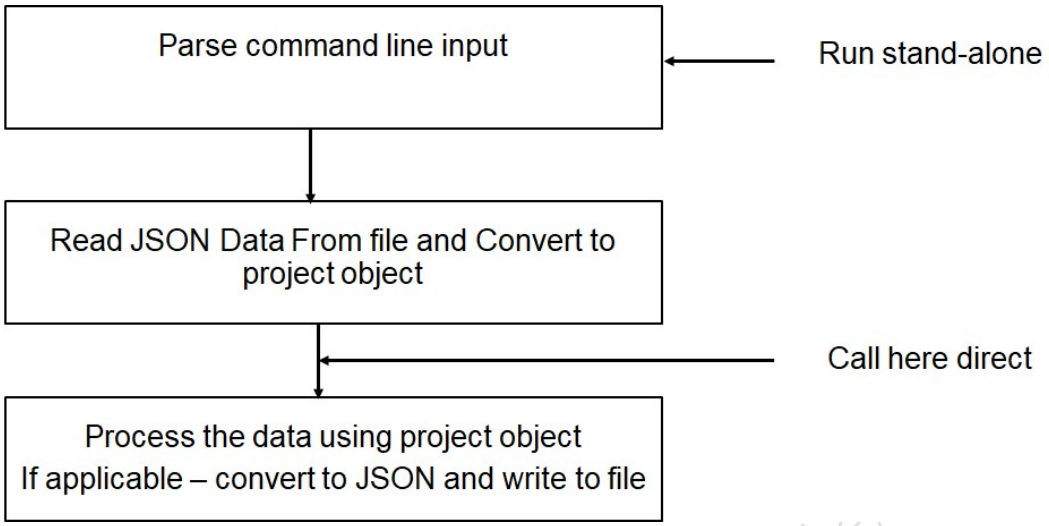
stats_clear.py

Clear all statistical counters

tips.py

Examples only. Miscellaneous tips on how to add data to the objects, write to the log, etc.

A Closer Look at the Interfaces



Converting, writing to a file, reading back, and converting again is time consuming but useful for script development. All of the sample applications use the standard `brcddb` objects and have two entry points. One to read in the JSON object and convert to the brcddb database. The other entry point is for other scripts to import this one and call the main entry point with the project object directly.

Generate A Report

Even if you don't plan on using the `brcddb` library or any of the pre-written applications, everyone needs a report, even if for no other reason than to validate your own scripts. A DOS batch file, jt.bat, is included in the application folder which you can modify to meet your own needs.

The capture application automatically determine the virtual switches in each chassis and poll all switches.

Due to some of the defects with how empty lists are returned, capture.py may display some errors. Usually Status=400, "Bad Request". It will also complete with a non-zero exit code, usually 2. As long as the exit code is 3 or less, you can ignore them.

How To Do Stuff

Common Notes

None vs. not present (null)

Python dictionaries are converted to JSON and sent to the switch. The `None` type in Python is not a common programming language type and more importantly, not a type understood or

defined in JSON. The Python `json` library will convert any value of type `None` to the text string "`null`". When FOS receives the request, it will assign a value of "`null`" to the key value pair. In places where "null" is acceptable input, such as free form text fields, FOS will accept the value; however, this is probably not what was intended. In most cases, FOS returns an error. If an element is not used, it must not be present in the content.

Using The Standard Python json Library With Trailing Commas

When hard coding dictionaries or lists for content to send to the switch do not include a comma after the last element. It has been observed that doing so will cause errors.

Understanding The Examples

In all examples throughout the remainder of this section, to keep URLs to a single line the common preamble has been removed:

```
https://10.10.10.10/rest/
```

For example, the full URL to get the security certificates is:

```
https://10.10.10.10/rest/running/brocade-security/security-
certificate
```

Throughout this document, it is referred to as just:

```
running/brocade-security/security-certificate
```

`brcdapi` is a driver typically installed in the Python lib folder. `brocade-rest-api-examples` is a folder containing examples. In some cases, the drivers perform multiple requests so as to simplify tasks for scripts so the example references may be in a driver. Details and where to find these folders are described elsewhere in this document.

Supplemental Documentation

Consult the "Brocade Fabric OS REST API Reference Manual" for leaf definition details. In the course of explaining how to perform certain operations it is necessary to elaborate on the leaf definitions in which case a subsection "Data:" is provided. The "Data:" subsection is not always included or may only contain information to augment the API Reference Manual.

Often, the supplemental documentation is provided simply to provide search terms such as an equivalent FOS command for URI association.

Security Certificates

`brocade-rest-api-examples/certs_get.py`

What You Need To Know

From a project time budgeting perspective to automate security certificate maintenance, programming to the FOS API is relatively minimal. Most development time will be spent on the interface external to FOS (what certificates to check, what values to use with a CSR, etc.). If you are not already familiar with security certificates, the most significant time factor will be in learning OpenSSL, x509 cryptography, and your organizations policies and procedure. Even if you are familiar with OpenSSL, taking the Brocade SSL 220 training course is highly recommended as it covers some nuances specific to FOS.

It's important to emphasize the need for you to understand your organization's security policies and methodologies for obtaining certificates. The method for obtaining certificates particularly will have a significant effect on how you will design code. Generating security certificates is external to FOS.

Typically, the security teams for organizations have a window, such as 30 days, prior to the expiration date of a certificate when the certificate is updated. The process to automate this is:

1. Read the certificates
2. Filter out the certificates you won't be updating
3. Check the certificate expiration dates
4. If the certificate does not exist, is expired, or approaching the expiration date, get a new certificate. Certificates come from a CA.
5. Apply any new certificates.

It is common to refer to replacing a certificate that is about to expire as updating the certificate. Procedurally, and therefore programmatically, certificates are not updated. They are replaced by loading a new certificate. When keeping the same public key and only applying a certificate with private key who's only change is the expiration date, there will be a few seconds where the interface is unavailable (switch returns a status of 503 indicating its busy) but HTTPS if never removed from the interface.

FOS uses a standard openSSL library. Often, when an error is returned it is the error from the openSSL library, not FOS. In these instances, additional information can be found from a simple web search using "openssl" and the detailed error message as search terms. For example, if the root CA is missing in the certificate chain, "unable to get local issuer certificate" is returned because that is what is returned by the openSSL library.

Common OpenSSL acronyms used in this section:

CA	Certificate Authority. This refers to the system that generates a certificate. This is organization dependent. It is not specific to FOS.
CSR	Certificate Signing Request. This is the first step in setting up a secure interface.

Some FOS commands can act on multiple objects but the API is limited to one operation per request. Writing code to loop multiple operations is easy but this becomes significant when deleting certificates because once the HTTPS cert is removed, the login session is no longer valid. You must re-login as HTTP to delete any additional certificates.

All certificate import and export files have Unix style new line characters, just "\n", not "\r\n". When working in a Windows environment, the "\r" must be stripped from certificates before sending them to the switch.

Test were performed with HTTPs certificates only. The time to read certificates took 35-40 sec. It is unknown if additional certificates will need additional time. You may need to adjust the HTTP connect timeout.

Reading Certificates and Exporting Certificates and CSRs

API request:

```
GET running/brocade-security/security-certificate
```

Note that the response contains the base64 encoded certificates and CSRs. This eliminates the need for an external FTP or SCP server.

Relevant FOS Command(s)

```
seccertmgmt show
seccertmgmt export
```

Example(s):

```
brocade-rest-api-examples/certs_get.py
brocade-rest-api-examples/certs_eval._get_certs()
brocade-rest-api-examples/certs_eval._extract_cert_or_csr()
```

Returned Data:

All certificates and signing certificates supported by FOS are returned. The response is a list of dictionaries, one entry per certificate, as follows:

Key	Type	Value
`certificate-entity`	str	This is the certificate entity. Possible values are: '`ca-server`', '`ca-client`', '`csr`', and '`cert`'.
`certificate-type`	str	The certificate type. Possible values are: '`commoncert`', '`radius`', '`ldap`', '`syslog`', '`fcap`', '`https`'
`certificate`	str	Only included if the certificate exists. Typically not used. Contains information about the certificate in plain text.
`certificate-hexdump`	str	Only included if the certificate or CSR exists. This is a PEM standard formatted block of base64 encoded text, typically returned as a .pem file from a certificate authority. This is what is used when exchanging certificates within FOS as well as with an external CA. Use x509 from cryptography (a standard Python library) to interrogate this data.
`certificate-verbose`	str	Same comments as with "certificate" apply but in addition to the encoded certificated, it includes plain text for human readability. This is what would be returned from `openssl -verbose`

Certificate Signing Request (CSR)

As soon as a CSR is requested, the HTTPS interface is disabled. As a result, your HTTPS connection is dropped and you will need to log back in using HTTP. Typically, this is only done once either at initial switch configuration time or when initiating a new methodology for security certificates. Usually, new certificates with updated expiration dates only are applied in production environments which avoids the need for an exposed HTTP interface.

If certificates already exist when generating a CSR, the recommended best practice is to delete the certificates first beginning with the certificate (private key, referred to as "cert" in FOS) and then the CA (public keys, referred to as "ca-server" and "ca-client" in FOS). This is a general recommendation and not specific to the FOS API.

The CSR is not returned with the API request used to generate the CSR. It must be read back using the method described in the previous sub-section, "Reading Certificates". The content of `certificate-hexdump` is what is sent to the CA.

API request:

```
POST running/brocade-security/security-certificate-generate
```

Relevant FOS Command(s)

```
seccertmgmt generate -csr
seccertmgmt export -csr
```

Example(s):

```
brocade-rest-api-examples/cert_eval._generate_csr()
```

Review the content in the table below with your security team to determine specific values. Since some of the fields are required by FOS but not by OpenSSL, "Required" and "Optional" in the table below refer to the FOS requirement. As of FOS 9.1, these are the only parameters supported. Additional parameters may be available in future versions of FOS.

Content sent to the switch:

Key	**Type**	**OpenSSL Configuration File Equivalent**
`certificate-entity`	str	Required. Same as in the previous "Reading Certificates" subsection.
`certificate-type`	str	Required. Same as in the previous "Reading Certificates" subsection.
`algorithm-type`	str	Required. See notes with `key-size`
`key-size`	str	Required. May be in the `[ req ]` section of an `openssl` definition file but this is typically specified as an option with in the `openssl` shell command with `-newkey`. For example `-newkey rsa:2048` indicates an RSA algorithm using a 2048 bit key. Keep in mind that FOS supports p384 so this value must be text, not a number. For a list of FOS supported key sizes, check the FOS Command Reference Guide for `-keysize` in the `seccertmgmt` command.
`hash-type`	str	Required. May be in the `[ req ]` section but this is typically specified as a dash option when executing the `openssl` command. For example, `-sha256`. For a list of FOS supported hash algorithms, check the FOS

		Command Reference Guide for `-hash` in the `seccertmgmt` command.
`years`	int	Required. It's not clear as to why this is a parameter for a CSR. The recommendation is to set it to 1 to satisfy FOS. Although required input, it has no bearing on the life span of the subsequent certificates you will create from a CA. Typically, the expiration of a certificate is defined with the `-enddate` or `-days` option with `openssl` command.
`keypair-tag`	str	Optional. Only used for FCIP extension. This field should be omitted for switch certificates.
`country-name`	str	Required. `countryName`
`state-name`	str	Required. `stateOrProvinceName`
`locality-name`	str	Required. `localityName`
`organization-name`	str	Required. `organizationName`
`unit-name`	str	Required. `organizationalUnitName`.
`domain-name`	str	Required. `commonName`
`ip-address-in-subject-alternative-name`	bool	Obsolete. Do not use.
`subject-alternative-name-dns-names`	dict	Optional. `subjectAltName`. Note that this field is for DNS names only. Subject alternative names with an IP address, normally preceded with "`IP:`" should be put in `subject-alternative-name-ip-addresses`. DNS names are added to the list of alternate names first followed by IP addresses. WARNING: This is a dictionary. The list of DNS names must be in a sub-leaf "
`subject-alternative-name-ip-addresses`	list	Optional. `subjectAltName`. Only IP addresses should be in the list. The preceding "`IP:`" typically used with openSSL must be omitted.

Applying New Certificates

API request:

```
POST operations/security-certificate/security-certificate-
parameters
```

Note that this URI is new in FOS 9.1. The certificate is base64 encoded and sent to the switch as part of the content. This eliminates the need for an external SCP server. The GET method is not supported on this URI. See "Reading Certificates".

Relevant FOS Command(s)

```
seccertmgmt import
```

Example(s):

```
brocade-rest-api-examples/certs_eval._add_cert()
```

Content sent to the switch:

Key	Type	Value
`certificate-entity`	str	See "Reading Certificates". This is the certificate entity to be updates.
`certificate-type`	str	See "Reading Certificates". This is the certificate type to be update.
`certificate-name`	str	This is used for FCIP extension only. This should not be included in the content for the request when updating switch certs.
`action`	str	When applying certificates, this will always be 'import'
`certificate-hexdump`	str	See "Reading Certificates". This the content of the PEM file returned from the CA.

Deleting Certificates

The equivalent CLI command allows you to delete all certificates but with the API, certificates must be deleted one at a time. Coding a loop to do this should be easy but keep in mind that if you delete the HTTPS certificate (so you must have logged in with a HTTPS), the connection is immediately dropped. The interface needs a few seconds to be reconfigured and then you will need to log back in as HTTP.

API request:

```
DELETE running/brocade-security/security-certificate-action
```

Relevant FOS Command(s)

```
seccertmgmt delete
```

Example(s):

```
brocade-rest-api-examples/certs_eval._del_cert()
```

Content sent to the switch:

Key	Type	Value
`certificate-entity`	str	This is the certificate entity to delete.
`certificate-type`	str	See “Reading Certificates”. This is the certificate type to be update.
`certificate-name`	str	This is used for FCIP extension only. This should not be included in the content for the request when updating switch certs.
`action`	str	When applying certificates, this will always be ‘import’
`certificate-hexdump`	str	See “Reading Certificates”. This the content of the PEM file returned from the CA.

HTTPS Keep Alive

Disabling the HTTPS interface and setting the keep alive is done with the chassis configuration URI. See sub-section “Chassis Configuration: HTTPS Interface”.

Chassis Configuration

What You Need To Know

Programming to the FOS API is relatively minimal. Response times for all URIs are within a few seconds so no special timeout considerations are required.

General Configuration

API request:

```
GET running/brocade-chassis/chassis
PATCH running/brocade-chassis/chassis
```

Relevant FOS Command(s)

```
chassisshow
chassisname
```

Example(s):

```
brocade-rest-api-examples/api_get_examples.py
```

Data:

Most Brocade manufactured products are rebranded and sold through a storage partner. The storage partner may have a different naming convention than that used by Brocade (manufacturer). “vendor” in the leaf name refers to naming or numbering conventions used by the OEM (storage partner).

FRU Status and Temperature Sensors

API request:

```
GET running/brocade-fru/blade
GET running/brocade-fru/fan
GET running/brocade-fru/history-log
GET running/brocade-fru/power-supply
GET running/brocade-fru/sensor
GET running/brocade-fru/wwn
```

Relevant FOS Command(s)

```
chassisshow
slotshow
tempshow
```

Example(s):

```
brocade-rest-api-examples/api_get_examples.py
```

High Availability (HA) Status

API request:

```
GET running/brocade-chassis/ha-status
```

Relevant FOS Command(s)

```
hashow
```

Example(s):

```
brocade-rest-api-examples/api_get_examples.py
```

Configuring REST HTTPS: Keep Alive, Enabling/Disabling,

Use this URI to configure:

- HTTPS keep alive
- Enable or disable the HTTPS at a chassis level
- Set the maximum number of REST sessions

API request:

```
GET running/brocade-chassis/management-interface-
configuration

PATCH running/brocade-chassis/management-interface-
configuration
```

Relevant FOS Command(s)

```
mgmtapp
```

Example(s):

```
brocade-rest-api-examples/app_config.py
```

Data:

The keep alive time, `https-keep-alive-timeout`, is 15 sec when keep alive is enabled, `https-keep-alive-enabled`. As of FOS 9.1, there was no means to alter this time.

As of FOS 9.1 the maximum number of Rest sessions, `max-rest-sessions`, is 10.

Logical Switches

What You Need To Know

- Fabric IDs (FID)
    - The FID number will always be 0 if virtual fabrics is disabled

    - FID 128 is the default FID number for the default logical switch; however, it is possible to change the default logical fabric ID. The FID for the default logical switch should be checked.
- Timing Considerations
    - It takes about 20 sec to create or delete a logical switch
    - It takes about 400 msec to move each port
    - To avoid HTTP timeouts, the `brcdapi.switch` methods create or delete switches as a single request and add/remove ports as separate requests of no more than 32 ports per PATCH request.
        - The interface to the `brcdapi.switch` module simply requires switch configuration information and a list of ports. This module breaks down the switch configuration process into multiple requests to avoid HTTP timeouts but the response to the calling method can and likely will exceed the HTTP timeout
    - A 60 second HTTP connection timeout is recommended
- Although setting an insistent domain ID and setting the domain ID number are within the same URI, testing has shown that the domain ID may not get set so these should be done as two separate requests.
    - Setting insistent domain ID will return an error on logical switches defined as ficon. This is because insistent domain ID is inherent in FICON switch types and the logical in FOS simply tests to see if any requests are made insistent domain ID regardless of the setting.
- Port addressing (port address binding)
    - Not exposed through the API until FOS 9.1.
    - Uses the 'operations', not 'running' URI to set.
- Even if writing your own driver to handle logical switch configuration, it is highly recommended to read through the comments in `brcdapi.switch` for timing considerations and other nuances.
- When adding ports to a logical switch, the WWN of the switch is required.
    - To get the WWN, you must read back the switch information.
    - `?vfid=xx` is still required
- A disabled switch is not part of a fabric.
- Enabling a switch can take up to 10 seconds to complete.
    - An HTTP status code of 400 with "The Fabric is busy" in the error message is a typical API response when immediately attempting to do other work with the switch.
    - When using the driver, brcdapi, the brcdapi_rest module sleeps for 10 seconds and re-drives the request up to 5 times before returning the error.

API request:

```
GET or PATCH
running/brocade-fibrechannel-switch/fibrechannel-switch

PATCH running/brocade-fibrechannel-configuration/fabric

POST
running/brocade-fibrechannel-logical-switch/fibrechannel-
logical-switch

POST operations/port
```

Relevant FOS Command(s)

```
configure
lscfg
switchshow
switchenable
switchdisable
switchpersistentenable
switchpersistentdisable
```

Example(s):

```
brocade-rest-api-examples/switch_create.py
brocade-rest-api-examples/switch_delete.py
```

MAPS

What You Need To Know

When set to 60 seconds, HTTP timeouts were occurring in development test when executing `applications/maps_config.py`. This module creates all the SFP rules defined in sfp_rules_rx.xlsx. When reduced to creating just 20 rules per request, the longest request took 15 seconds.

Recommendation: Since configuring MAPS policies is rare, rather than perform extensive testing to determine under what conditions the configuration of MAPS rules were taking in excess of a few seconds, the recommendation is to limit the number of rules to 20. Additional testing should be done if you are configuring MAPS policies for something other than SFPs.

Reading and Configuring MAPS

API request:

```
running/brocade-maps/rule
running/brocade-maps/maps-policy
running/brocade-maps/group
running/brocade-maps/maps-config
running/brocade-maps/dashboard-rule
running/brocade-maps/dashboard-history
running/brocade-maps/dashboard-misc
running/brocade-maps/system-resources
running/brocade-maps/paused-cfg
running/brocade-maps/monitoring-system-matrix
running/brocade-maps/switch-status-policy-report
running/brocade-maps/fpi-profile
```

Relevant FOS Command(s)

```
mapsconfig
mapsdb
mapspolicy
mapsrule
mapssam
```

Example(s):

```
brocade-rest-api-applications/maps_config.py
brocade-rest-api-examples/maps_clear.py
```

Zoning

What You Need To Know

Zoning typically requires reading some basic fabric information in addition to zoning information. For example, reading the name server so as to determine physical port locations of where certain logins within a zone occurred is common in zone reports.

Even if you do not intend to use the `brcdapi.zone` driver or `brcddb.app.zone application`, reading the comments in that code, especially `brcdapi.zone`, is highly recommended as there are nuances documented in the code comments that are not documented herein.

Timing Considerations:

To add a server to a SAN you will want to give the HBA an alias, create a zone, add the zone to a zone configuration, and activate the zone configuration. To add an alias the `brcddb.apps.zone.send_zoning()` executes the following requests:

1. Login
2. Read basic chassis data
3. Read basic fabric data
4. Read the current zoning database
    a. At this point, the library does some basic checking. In the case of an alias, the alias name must be in a valid format and there must be a valid WWN or domain, index.
5. Send the alias updates
6. Send the zone updates
7. Send the zone configuration update.
8. Active the zone configuration (sometimes, activating the new zone configuration is a step saved for a change control window and not part of this process)
9. Logout

To estimate how long individual changes will take, multiply the number of changes by 0.4 seconds and add another 2 seconds for the basic read of data and the save. For example:

10	Alias creations
+ 4	Zone creations
+ 1	Zone configuration change (add 4 zones in one step)
= 15	Total zone changes
x 0.4	Estimated time in seconds per change
6.0	Processing time in seconds for all changes
+ 2.0	Time in seconds for other processing
= 8.0	Total time in seconds to make all changes

There is a bulk zoning library that accepts an unlimited number of changes and bundles all the changes into a single request such that the entire zoning database is rebuilt and replaced in just 6 requests. Bulk zoning is typically reserved for restoring a zone database or repurposing a switch but if the timing considerations of making individual zone changes is excessive, bulk zoning might be a better alternative. Extremely large zone databases can be built in 2-3 seconds this way.

Testing a large number of zoning changes was too limited to quote a time frame here but although they took longer than bulk changes and since most zoning changes are not vast, testing effort was stopped. The recommendation is to use whatever approach you prefer. Most customers prefer to make individual zoning changes.

Recommendation: Although an HTTP connection timeout of 15 seconds was adequate in limited test, most testing was done with a timeout of 60 seconds. Unless time is of the essence, the recommendation is to leave the timeout at 60 seconds.

Reading and Modifying the Zone Database

API request:

```
running/brocade-zone/defined-configuration
running/brocade-zone/effective-configuration
running/brocade-zone/fabric-lock
```

Relevant FOS Command(s)

```
aliadd
alicreate
alidelete
aliremove
cfgadd
cfgclear
cfgcreate
cfgdelete
cfgdisable (no equivalent in the API)
cfgenable
cfgremove
cfgsave
defzone
zoneadd
zonecleanup
zonecreate
zonedelete
zoneobjectcopy
zoneobjectexpunge
zoneobjectrenamed
zoneobjectreplace
zoneremove
```

Example(s):

```
brocade-rest-api-applications/cli_zone.py
brocade-rest-api-examples/zone_config.py
```

Port Statistics and Configuration

What You Need To Know

Port configuration changes are relatively fast and therefore do not need any special HTTP connect timeout considerations; however, keep in mind that if all E-Ports connected to a fabric are disabled a fabric rebuild occurs. Fabric rebuilds are time consuming. Similarly, enabling E-Ports have the same fabric rebuild consideration.

Keep in mind that FOS rules require that all special port configurations must be removed before they can be moved to another logical switch.

Even if writing your own driver to handle logical switch configuration, it is highly recommended to read through the comments in `brcdapi.port` for port configuration nuances and other details.

Normally, ports are depicted with simple s/p notation. When reading SFP information, `running/brocade-media/media-rdp`, the port number is preceded with the media type such as "`fc`"

FOS polls the ASIC for port statistics every 5 seconds with Gen 5 switches and every 2 sec on Gen 6 and Gen 7 switches. Due to variability in processing and networking, very short poll cycles may result in returning the same statistics read within the same poll cycle. Testing showed that to reliably read distinctly different FOS polls of ASIC statistics that 1 sec needed to be added to the shortest poll cycles (6 sec for Gen 5 and 3 sec for Gen 6 & Gen 7). See comments with `time-generated` in the Returned Data subsection.

Configuring Ports

API request:

`running/brocade-interface/fibrechannel`

Relevant FOS Command(s)

```
portcfg*
portdecom
portdisable
portdporttest
portenable
portflagsshow
portlog*
portname
```

```
portpeerbeacon
portshow
```

Example(s):

```
brcdapi/port.py
brocade-rest-api-examples/port_config.py
brocade-rest-api-examples/port_enable_all.py
brocade-rest-api-examples/set_port_default_all.py
```

Port Statistics

API request:

```
running/brocade-interface/fibrechannel-statistics
running/brocade-interface/gigabitethernet-statistics
```

Relevant FOS Command(s)

```
portstatsshow
portstats64show
```

Example(s):

```
brocade-rest-api-examples/ge_stats_to_db.py
brocade-rest-api-examples/stats_clear.py
brocade-rest-api-examples/stats_to_db
```

Returned Data:

Key	Type	Value
`time-generated`	int	This is epoch time. It is the time of the API request for the statistics, not the time FOS polled for the statistics. This means it cannot be used to determine if the same statistics are being read twice.
`All else`	int	All statistical counters are 64 bit counters

Media (SFP) Data

API request:

```
running/brocade-media/media-rdp
```

Relevant FOS Command(s)

```
sfpshow
```

Example(s):

```
brocade-rest-api-examples/api_get_examples.py
```