Consoli Solutions, LLC

AUTOMATION

Software Developers Kit for The FOS API

12 January 2026

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

Education

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

API-220	Introduction to Fabric OS Introduction
REST-320	Fabric OS REST Implementation
REST-321	Fabric OS API Requests
ANSIBLE-320	Brocade Ansible Implementation

And if you are also interested in the SANnav API:

REST-330	SANnav Management Portal REST API Implementation
REST-331	SANnav Northbound Kafka Streaming

github

https://github.com/brocade	This is a good place to get the Yang models (useful as supplemental documentation), Ansible scripts, and sample Ansible Playbooks.
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.

Required & Recommended Software

- FOS
    - 8.2.1c or higher for Gen5. Has support for basic operations such as zoning, reading chassis FRU status, port status, and logins. Limited FICON support.
    - 9.1.1d – Builds on 8.2.1c by adding logs, complete switch configuration, and most of what is needed. Full support for FICON.
    - 9.2.0 – Fixes issues with MAPS and config upload/download
- 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. At the time this document was written, the following additional libraries were needed:

Library	For Additional Information
jdcal	https://pypi.org/project/jdcal
et_xmlfile	https://pypi.org/project/et-xmlfile
cryptography	https://pypi.org/project/cryptography
urllib3	https://pypi.org/project/urllib3
openpyxl	https://pypi.org/project/openpyxl
deepdiff	https://pypi.org/project/deepdiff

*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.

Although you may only need a subset of these depending on what samples you intend to use, installing them is quick, easy, and they take up little space. The texttable and paramiko libraries are not used by any of the sample scripts discussed herein but they are commonly used so you may as well install them as well.

The URLs for the libraries are predefined in the pip installer. The easiest way to install these libraries is to copy and paste the following into a batch file and run it in a Unix shell or DOS command prompt window:

```
pip install jdcal

pip install et_xmlfile

pip install cryptography

pip install urllib3

pip install openpyxl

pip install deepdiff
```

<u>Optional</u>

Paramiko is now part of the standard Python package and is required by some of the sample scripts. Texttable is not used by any of the sample scripts; however, it is a very commonly used library. If you are just getting started with Python, consider installing it.

```
pip install paramiko

pip install texttable
```

If the library was already installed, the pip installer will check the version and update if necessary.

FOS Specific 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 environments 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.

Author’s Note: The key difference between the brocade-rest-api-examples and brocade-rest-api-applications is that the scripts in brocade-rest-api-applications use the brcddb database while brocade-rest-api-examples do not. From a development perspective, it made sense to keep them separate. For SAN administrators just using the tools, it probably makes more sense to keep all scripts in one folder.

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.

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
```

"Exist, as in the example above, indicates that a security certificate exists. By default, the highlighted text above will be "Empty" so you will need to create a certificate.

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

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

Throttling

The throttling parameters in FOS 9.x are hard coded. They cannot be changed. For FOS 8.2.x, look for `mgmtapp --config` options in the FOS command Reference Guide. All versions of FOS support `mgmtapp -show`. When executing the `mgmtapp --show` it has been observed in some instances to display all throttling parameters normally and then be followed with:

```
log init failed. Unable to retrieve CLI history.

Failed to open CLI history file.
```

The CLI history error can be ignored.

HTTPS keepalive

When HTTPS `keepalive` is enabled, the HTTPS interface will disconnect if there is no activity on the interface within 15 sec. This time is subject to change in future versions of FOS. The timer does not start until a request has been acknowledged so the time it takes for a request to complete is not a consideration.

If the HTTPS `keepalive` is enabled, a status of 408 with a “`disconnect`” error message will be returned if the timer expires. When this happens, a new Rest session must be established.

If you intend to set breakpoints during script development you should check to see if HTTPS `keepalive` is enabled. If enabled, disable it during development and re-enable it when you are done.

To enable or disable the `keepalive` via the Rest API, search this document for “app_config”.

To see if `keepalive` is enabled from the CLI:

```
mgmtapp --show
```

See note regarding CLI history errors in the previous sub-section

To disable it:

```
mgmtapp --disable keepalive
```

To re-enable it

```
mgmtapp --enable keepalive
```

Validate Library Requirements

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.

If you are not using the brcddb or brocade-rest-api-applications you can ignore those error messages.

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.

Example:

```
py login.py -ip xx.xx.xxx.15 -id admin -pw password -s self
```

For Programmers

Tips and Unique Features For Python Newbies

PEP 8 Style Guide

The PEP 8 Style Guide defines naming, indentation, and other conventions. Although most of what is covered in this style guide is not required for Python programming, most IDE understand this format and will warn appropriately. Most sample Python scripts follow this format as well so understanding this format will make it easier to read scripts from other sources as well as make it easier for others to understand your script when sharing.

https://pymbook.readthedocs.io/en/latest/pep8.html

All of the sample scripts on github/jconsoli follow this standard.

Sphinx Documentation

The Sphinx documentation standard is commonly used in Python scripting. It is understood by most IDE. For example, if you call a method with an integer as an input parameter but the method is defined to accept a string, most IDE will warn you.

All of the sample scripts on github/jconsoli follow this standard.

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 call a method from a tabledo 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.

Quirky Behavior

Long Running Requests

For long running requests using POST, PATCH, or DELETE, the FOS API implementation uses a separate URL branch, operations, rather than return a 202 status code, which is standard HTTP protocol, a separate branch was used to handle long running requests. This is described further in "URL: root, running, and operations".

Naming Conventions

Standard PEP 8 naming conventions were not used in the API interface. Furthermore, there isn't much naming consistency. For example, a switch WWN may be referred to as "wwn" in some places and "swwn" in other places.

The initial version of the API used integers to represent many things. With new FOS releases, these began changing to text. Text is a more modern approach, especially with programs such as Python that are interpreters. Although this change was done with good in mind, the naming convention is inconsistent. Usually, mnemonics were updated by appending "-string", but sometimes it's "-str" or "-v2".

Author's Note: I had suggested creating a new branch "v2" and to implement the changes by full leaves before I retired. I'm assuming this is where "-v2" came from, but I had intended it as a branch to avoid having two different mnemonics in representing the same thing but with integers instead of text in the same leaf.

It's not difficult to resolve these naming conventions. You just need to check each branch and leaf as you use resources.

Commendable Behavior

Port configurations, port statistics, and zoning can all be done in a single operation by passing lists of operands to FOS via the API. Programmers are advised to take advantage of this because:

- Zone freezes and lockouts are kept to a minimum. A full zoning database can be replaced in under 2 seconds.
- Port changes updated in sub-second time, thereby greatly reducing updates in change control windows.
- Port statistics are a consistent point-in-time for all ports in a logical switch.

URL: root, running, and 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 v9.2.0, not all requests that require a long time to execute were implemented. • As of FOS v9.2.0, 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. • The documentation in REST API Guide is for the internal remote procedure call (RPC), not the REST interface.

	"`input`" is not included in what is sent to the switch. "`response`" is what is returned, not part of the request. Search for "operations" in `brcdapi/port.py` for an example of how to build an operations request. **WARNING:** Operations URIs return a status of 200 regardless of the request content. The subsequent request for status will return an error if the request content is invalid but otherwise always returns a status of 200. If something failed in execution, there is a human readable text that explains the error in "`message`" but there is no way to automate interpreting a failure. Furthermore, `operations` requests do not support GET. They do not match `running` branch commands one-for-one, so comparing what you sent via and operations branch and validating it via running requests are complex.

Connection Headers

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

Throttling

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.

Typically, customers set these parameters up via the CLI at installation time so CLI examples are given in the table below. For customers using a script to automated initial configuration, search this document for "app_config".

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. 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 affected 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	IN FOS 9.1 and above, this parameter is set at 30 and is no longer user configurable. 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: As of this publication date, there was no known reason to 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 set at 120 and 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 match the 9.1 setting: `mgmtapp --config -samplerequest 120`

	Recommendation: By default, all 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 requests 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 is not possible to complete that many requests in a 30 sec window so this effectively disables request throttling.
Determine current settings	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.

As of FOS 9.2, creating a logical switch still needs to be broken up into multiple requests. The methods in brcdapi/switch.py do this so programming is simplified to a single call. That module is heavily commented for programmers that will not use this library to use as an example.

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. Instead, FOS uses the operations branch to handle long running requests. See notes in the "running vs operations" section.
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 but 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.

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.

Zoning

So that the zone database remains intact after a power cycle, the zone database is stored in flash memory. Regardless of where zoning transactions originate from, once a request for zone change is received by the switch, the switch copies the flash memory to dynamic memory where all subsequent changes are made. The zone database is saved to flash and pushed to all other switches in the fabric when the zoning changes are saved. Enabling a zone configuration automatically saves, copies the dynamic memory to flash, before sending the zone changes to the other switches in the fabric.

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.

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 over 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. Note that should this be fixed in a future FOS release, there is no need to change the report scripts but best practice checking code would need to be modified to take advantage of such a fix.

The `brcddb/brcddb_bp.py` module ignores remote current, remote temperature, and all remote SFP data if `remote-media-voltage` is 0 because it has been observed that when the voltage is reported as 0 that other remote parameters are invalid.

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 session-key
```

Using json.dumps()

It is common for programmers to include a comma at the end of the last item in a list (`list`) or dictionary (`dict`) because it simplifies adding items. The `json.dumps()` library will return an exception if there is a comma after the last item.

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 certain FICON 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.

Author's Note: I left the code to handle this in the driver: brcdapi.brcdapi_rest.py. It didn't effect anything that had normal lists. I didn't have the time or resources to test everything and the code was working, so I left it in.

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 switch 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 leaves:

`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 is not a single URI that returns this type of checking.

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 it being deployed.

The Default Switch

From the factory, the default switch is FID 128. Although not often, customers do on occasion assign a different FID number to the logical switch. You cannot assume that FID 128 is the default switch.

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 is 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.

Disabling FID checking in a production environment is not recommended.

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 or Gen7 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 only 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.

All FOS rules are relevant to the API. For example, 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 o Typically, only used for service which does not include zoning operations • A terminal telnet session • A script with a telnet session
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`.

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 is 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 is 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
- Cannot contain ':' (so no WWNs in standard format). Most other special characters are not permitted either.
- The hyperlink formula does not 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

Data Validation

If you assign a data validation to a worksheet, it must be used in at least on cell on that worksheet. If not used, an error occurs when opening the workbook. If you allow Excel to fix the error, other formatting is missed.

This works:

```
_dv_yn = DataValidation(
    type='list',
    formula1='"Yes,No"',
    allow_blank=False
)
sheet.add_data_validation(_dv_yn)
```

```
row = col = 1
cell = xl_util.get_column_letter(col) + str(row)
dv.add(cell)
```

This does not work:

```
_dv_yn = DataValidation(
    type='list',
    formula1='"Yes,No"',
    allow_blank=False
)
sheet.add_data_validation(_dv_yn)
row = col = 1
cell = xl_util.get_column_letter(col) + str(row)
# dv.add(cell)
```

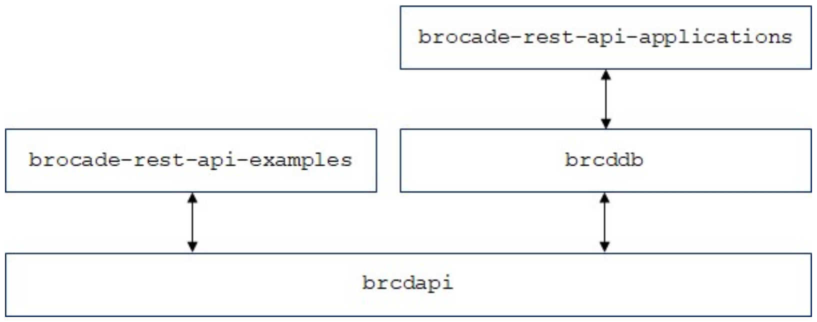
Working With The FOS Rest API

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

It is not always obvious how resources are related. For example, it is 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.

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”

Design

The general approach is to read data and then so something with the data.

brocade-rest-api-examples

All API example scripts read (GET) whatever data is required and then act on that data. Some of these examples simply display data to the console while others act on command line input, read data, and then take an action (POST, PATCH, or DELETE).

These examples rely on brcdapi. They do not use brcddb.

brcddb

The anchor for all of these libraries is in the class objects, `brcddb/class`. A Python class object is created for each area, such as chassis, fabric, and switch. Some of these objects have subordinate objects. For example, the switch object contains port objects. These objects are placed in a project object which essentially is a hierarchical relational database.

All data read from a switch is a dictionary (`dict`) in JSON format or XML. In all of the sample scripts and libraries discussed herein, only JSON is used. These dictionaries are added to the associated object.

The purpose of the database is to simplify tasks. For example, to perform a maintenance operation on a storage port it may be useful to determine all server applications zoned to that port so that the application owners can be notified. Using this database, in a single line of code all login objects zoned to the devices logged in on that port.

Some noteworthy features:

api.interface	Simplifies adding data to the brcddb class objects and performing GET requests on multiple FIDs. The most useful method is get_batch() which accepts a list of URIs, sorts out chassis level and logical switch level URIs, operates on the chassis URIs first and then the logical switch level requests. All requested data is added to the appropriate class object. Logical switch data can automatically be collected on all logical switches or a specified list of FIDs.
api.zone	Contains a methods to simplify bulk zoning operations.
report	Several modules to simplify report creation using brcddb objects

brocade-rest-api-applications

The applications rely on the brcddb database as well brcdapi libraries for reading and sending data. Data is read from the switch or switches and added to the appropriate class objects.

The data capture scripts build the brcddb database, convert it to standard supported JSON dictionaries, and write the JSON output to a file. Some of the application scripts rely entirely on previously collected data while using it is optional with other scripts. This is useful for:

- Reading data from multiple switches in parallel by launching multiple capture sessions, capture.py, and reassembling the data after it completes.
- Capturing data for future use. For example, zone_restore.py

Import Hierarchy

Import hierarchy for the brcdapi and brcddb libraries.

Author's Note: Yes, this could have been organized better.

Tier 0	
brcdapi.log	Can be imported by any module.
Tier 1	
brcdapi.gen_util	Can import Tier 0. Can be imported by any module in Tier 2 and above.
brcdapi.file	
brcdapi.fos_cli	
brcdapi.excel_fonts	
brcdapi.excel_util	
Tier 2	
brcdapi.util	Can import anything in Tier 1 and below. Can be imported by any module in Tier 3 and above.
Tier 3	
brcdapi.fos_auth	Can import anything in Tier 2 and below. Can be imported by any module in Tier 4 and above.
brcdapi_rest	
brcdapi.sannav_auth	
Tier 4	
brcdapi.port	Can import anything in Tier 3 and below. Can be imported by any module in Tier 5 and above.
brcdapi.zone	
Tier 5	
brcdapi.switch	Can import anything in Tier 4 and below. Can be imported by any module in Tier 6 and above. Can be imported by any application in brocade-rest-api-examples.
Tier 6	
brcddb.brcddb_chassis	Can import anything in Tier 5 and below. Can be imported by any module in Tier 7 and above.
brcddb.brcddb_common	
brcddb.brcddb_login	
brcddb.classes.util	

Tier 7	
brcddb.brcddb_zone	Can import anything in Tier 6 and below. Can be imported by any module in Tier 8 and above.
brcddb.classes.alert	
brcddb.util.copy	
brcddb.util.obj_convert	
brcddb.util.search	
brcddb.util.util	
Tier 8	
brcddb.brcddb_switch	Can import anything in Tier 7 and below. Can be imported by any module in Tier 9 and above.
brcddb.app_data.alert_tables	
brcddb.app_data.report_tables	
brcddb.classes.chassis	
brcddb.classes.iocp	
brcddb.classes.login	
brcddb.classes.port	
brcddb.classes.zone	
Tier 9	
brcddb.brcddb_port	Can import anything in Tier 8 and below. Can be imported by any module in Tier 10 and above.
brcddb.classes.fabric	
brcddb.classes.switch	
brcddb.util.maps	
Tier 10	
brcddb.parse_nx	Can import anything in Tier 9 and below. Can be imported by any module in Tier 11 and above.
brcddb.classes.project	
brcddb.util.iocp	
Tier 11	
brcddb.brcddb_port	Can import anything in Tier 10 and below. Can be imported by any module in Tier 12 and above.
brcddb.util.compare	
Tier 12	
brcddb.brcddb_fabric	Can import anything in Tier 11 and below. Can be imported by any module in Tier 13 and above.
brcddb.util.parse_cli	
Tier 13	
brcddb.brcddb_bp	Can import anything in Tier 12 and below. Can be imported by any module in Tier 14 and above.
brcddb.api.interface	
brcddb.report.utils	
Tier 14	
brcddb.api.zone	
brcddb.report.chassis	

brcddb.report.fabric	Can import anything in Tiew 14 and below. Can be imported by any application in brocade-rest-api-applications.
brcddb.report.graph	
brcddb.report.iocp	
brcddb.report.login	
brcddb.report.maps	
brcddb.report.port	
brcddb.report.switch	
brcddb.report.zone	

Common

Standard Script Documentation

All scripts, whether in a library of a sample script, contain the following:

Public Methods & Data - A table near the beginning of the block comments at the beginning of the file with a summary of the public methods and, if applicable, public data.

Standard Python file information. For example:

```
__author__ = 'Jack Consoli'
__copyright__ = 'Copyright 2022, 2023 Jack Consoli'
__date__ = '29 Mar 2023'
__license__ = 'Apache License, Version 2.0'
__email__ = 'jack_consoli@yahoo.com'
__maintainer__ = 'Jack Consoli'
__status__ = 'Released'
__version__ = '1.0.5'
```

All public and most private methods use standard Sphinx documentation. For example:

```
def sample_func(input_a, input_b):
    """Brief description.

    :param input_a: Brief description of input_a
    :type input_a: list
    :param input_b: Brief description of input_b
    :type input_b: int
    :return: Brief description of return value.
    :rtype: dict, None
    """
```

Command Line Options

These options are common with all scripts in brocade-rest-api-applications and most scripts in brocade-rest-api-examples:

-d	Debug mode. Typically only used by developers. All data sent to or received from the switch is formatted and printed to the log.
-dm	Only available with applications that read Excel Workbooks for input. Workbooks are always read using `brcdapi.excel_util.read_workbook()`. This utility reads the worksheets into a list of worksheets which contain two-dimensional lists so that the sheet contents can be read in the same grid like fashion, row and column, that are consistent with the Excel worksheet grids. Options are: 0 Read the workbook only 1 Read the workbook and write the lists to a JSON file in the same directory and same file name as the workbook but with a .json extension instead of .xlsx. This same file name is used with options 2 and 3. 2 Do not read the workbook. Instead, read the JSON file. 3 Typically the default. If the last modified time stamp of the Excel file is later than that of the JSON file, behave as option 1. If the JSON file is newer, behave as option 2. The original purpose of this was to avoid having to read and parse the same Excel file multiple time. Reading large Excel files can be very time consuming. None of the scripts described herein use Excel file large enough for this to be a factor but since there were internal scripts that did read large Excel workbooks it was easy to pass it an option to same them as JSON. This Furthermore, people using these scripts may have other reasons to read large workbooks so it is provided as a utility.
-h	Prints help information and exits.
-id	User id. Only used for scripts that require switch login.
-ip	IP address of switch. Only used for scripts that require switch login.
-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.
-nl	A log file is not created when this option is set. Output is still printed to the screen.
-pw	Password. Only used for scripts that require switch login.
-s	Obsolete with FOS 9.0 and above. The default is HTTPS, `-s self`. None, `–self none`, for HTTP is supported with versions of FOS below 9.0. Only used for scripts that require switch login.
-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.

brocade-rest-api-examples

Miscellaneous (api_get_examples & cli_poll_to_api)

Most of these do not have a user interface. They are intended for programmers to use as references and to interrogate responses:

api_get_examples – Several GET commands

cli_poll_to_api – A list of URIs with the equivalent FOS command in comments. This has not been updated

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.

Examples:

Increase the number of rest sessions to 10:

```
py app_config.py -ip xx.xx.xx.xx -id admin -pw password -s self
-max_rest 10
```

Disable the keep-alive

```
py app_config.py -ip xx.xx.xx.xx -id admin -pw password -s self
-ka_dis
```

Certificates (certs_eval & certs_get)

certs_get reads and decodes certificate information from a switch and displays that information on the console.

Example:

```
py certs_get.py -ip xx.xx.xx.xx -id admin -pw password -s self
```

certs_eval illustrates how to read certificates, evaluate them for expiration dates, generate a CSR, and install certificates. This module uses the features added in FOS 9.1.1 that allow all data to be transferred in-band so there is no need for an external SCP/SFTP server. CSRs and certificates are read/written from/to a workbook in a method that can easily be swapped out for customer code to automate that aspect using their own servers.

Example:

Although intended as an illustration for programmers, it may be useful as is to evaluate certificates. To generate reports:

```
py certs_eval.py –i cert_eval_sample_input –a eval –r
certs_eval_report
```

Capture Statistics (ge_stats_to_db, stats_to_db)

These are programming examples only on how to capture statistics. See stats_c.py and stats_g.py in the applications for a tool to capture and graph statistics in Excel.

port_config

Supports the following actions:

name	Set the port name. Port list, -p, must be s/p:port_name
disable	Disable ports
p_disable	Persistently disable ports
enable	Enable port
p_enable	Persistently enable ports. Note: “persistently enable” is nomenclature carried over from the FOS CLI. There is only a persistently disabled flag which is set after disabling a port using “p_disable”. An “enable” will fail if the port is persistently disabled. “p_enable” clears the persistently disabled flag and then enables the ports.
decom	Decommission ports (E-Ports only)
clear	Clear port statistics
default	Disable and set ports to the default port configuration
eport_enable	Enable ports for use as an E-Port Note: This is not a port enable, p_enable. By default, all ports are universal ports meaning they can be used for device (N-Port) or switch to switch (E-Port). Ports can be defined for just switch to switch or device connections.
eport_disable	Disable ports for use as an E-Port. See notes with eport_enable.
nport_enable	Enable ports for use as an N-Port. See notes with eport_enable.
nport_disable	Disable ports for use as an N-Port. See notes with eport_enable.
reserve	Reserves POD license for ports
release	Releases POD license for ports

Example:

Reserve POD licenses for ports 0-12 and then enable them on FID 1

```
py port_config.py –ip xx.xx.xx.xx –id admin –pw password –s self
–fid 1 –p 0-12 \
    –a reserve,p_enable
```

brocade-rest-api-applications

Capture Data (capture, combine, multi_capture)

Captures data for a single chassis. Outputs a JSON formatted brcddb database entry for the chassis.

Examples:

Capture data used by report.py (a sub-set of all data)

```
py capture.py –ip xx.xx.xx.xx –id admin –pw password –s self –f
mydata/chassis_0
```

Capture all data. As of this writing, the supplied script that uses data not collected for report.py is chassis_restore.py. Customers that want to collect all data for potential future use without having to define what data is should use this option.

```
py capture.py –ip xx.xx.xx.xx –id admin –pw password –s self –f
mydata/chassis_0 –c *
```

Combine data from multiple captures into a single project:

```
py combine.py –i mydata –o mydata/combined
```

Note: The output of capture.py is a project that contains just one chassis. Although the most common use for this script is to combine multiple file outputs from capture.py, combine.py combines projects regardless of the number of chassis in the project.

Using a workbook for login credentials, launch multiple instances of capture.py, wait until the complete, then launch combine.py, and then generate a report:

```
py multi_capture.py –i mydata/combined –r –bp bp –sfp
sfp_rules_r10
```

Note: For those worried about the number of file handles/descriptors and HTTP connections while launching multiple instances of capture.py, each instance of capture.py will need one HTTP connection and open a log file. Other than machine memory, there are no limitations as to the number of open HTTP connections in either Unix or Windows. The maximum number of open file handles in Windows is 512. The maximum number of file descriptors in Unix by default is 1024. Linux and other Unix variants support at least 1024 file descriptors by default. As a practical matter, these limitations are not restrictive.

Reports (compare_report, report, port_count)

compare_report.py compares two brcddb database files (projects). Contains filtering to remove nuisance changes. These filters are hard coded. To modify them search for "_control_tables" in compare_report.py.

Example:

```
py compare_report.py -b mydata/combined_0 -c mydata/combined_1
    -r compare_report
```

report.py generate a SAN Health like report

Example:

```
py report.py -i mydata/combined -o my_report -bp bp
    -sfp sfp_rules_r10 -group group
```

port_count.py generates a detailed report of login speeds and device types logged into each port. It is typically used to determine SAN requirements when upgrading attached devices or the fabric switch.

Example:

```
py port_count.py -i mydata/combined -o port_count_report
```

Statistics (stats_c, stats_g)

stats_c.py is used to collect statistics. The typical use is to evaluate load balancing on a storage enclosure so typically two instances of this module are run, one for fabric A and one for fabric B. It only logs in once so if the polling interval, -p, exceeds 15 seconds you will need to make sure the keep alive is disabled. All data is collected in memory and only written to a file upon completion. The intended use is to collect statistics at short intervals, less than 15 seconds, over up to an 8 hour period so no thought was given to long polling cycles or excessively long data capture periods.

Example:

Collect statistics for logical switch FID 1 every 5 seconds for 2 hours:

```
py stats_c.py -ip xx.xx.xx.xx -id admin -pw password -o stats
    -fid 1 -p 10 -m 1440
```

stats_g.py is used to process the data captured with stats_c.py and put that data into an Excel workbook. Running a report with the –group option may be useful for determining the ports. The difference for each statistical counter between each poll cycle (so the first poll is just used as a reference) is parsed into a separate worksheet for each sample. Graphing is optional but typically used.

Example:

Graph the Tx and Rx counters for ports 3/14 and 4/14

```
py stats_g.py –i stats –gp 3/14,in-frames,out-frames;4/14,
    in-frames,out-frames –r report
```

Graph the top 5 ports with the most accumulated BB credit starvation

```
py stats_g.py –i stats –gs bb-credit-zero,avg-5 –r report
```

Zoning (cli_zone, zone_config_x, zone_merge, zone_restore)

With all zoning scripts, all FOS rules zoning rules apply.

cli_zone.py was intended as a programming example for people familiar with CLI zoning but some people find it useful because they have other scripts that generate CLI and bulk zone updates using the API are significantly faster than copy and paste into an SSH session. It also has the advantage of testing the CLI commands before applying them.

zone_config_x.py was a lab tool only used to create zones for test purposes. Since it has a test mode, some people found it useful to use for production zoning.

zone_merge.py is used to merge zones between two or more fabrics. The input zoning information can come from project files from previously collected data or real time.

Example:

The WWN of the fabrics and the zone configurations to merge is needed. Determine all fabrics and zone configurations:

```
py zone_merge.py –i my_config_file -scan
```

To test the zone merge before implanting it:

```
py zone_merge.py –i my_config_file -t
```

To merge the zones into a zone configuration and activate it:

```
py zone_merge.py –i my_config_file –cfg zone_cfg -a
```

chassis_restore.py

This is similar to the FOS command `cfgdownload`. The intent was to allow customers to create a template and apply it to other switches but it can also be used to restore a replacement chassis. Capturing data with the `-c *` option is required if user accounts are to be restored. Future features may require this option as well.

The differences are:

- Operates on a previously collected project file.
- Customer user accounts are recreated
- Other things, such as customer MAPS rules, are intended to be added in the future
- There is an option to map FIDs so the FIDs do not need to match
- All restore parameters are optional
- It makes a best effort restore so if something fails it reports the error but keeps running
- When restoring logical switches, all existing logical switches are deleting (which means all ports are disabled and returned to the default state) and then logical switches are rebuilt
- There is a -`eh` option which prints additional detail to the console and log.
- There is -`scan` option so you can interrogate the previously collected data.

Examples:

As with zone_restore.py, it is typical to scan the project file you are restoring from first:

```
py chassis_restore.py -ip xx.xx.xx.xx -id admin -pw password
    -scan
```

To fully restore a chassis to look like chassis WWN 10:00:c4:f5:7c:16:8b:7b from a project file:

```
py chassis_restore.py -ip xx.xx.xx.xx -id admin -pw password
    -p * -wwn 10:00:c4:f5:7c:16:8b:7b
```