Simplified Brocade FOS API: Getting Started

Consoli Solutions

Keywords: Brocade FOS REST API Python Software Developers Kit (SDK)

Description:

How to set up your workstation and configure a Brocade SAN switch to use the FOS RESTConf API. An introduction to sample API scripts will also be covered. Includes how to install Python, load necessary libraries, configure and prepare a Brocade switch for use with the FOS RESTConf API, and the drivers and sample scripts on my github site. The libraries, supplemental documentation, and sample scripts described herein essentially make up a software developers key for using the FOS API.

Make sure you download a copy of this presentation which is available in PDF on this YouTube page as well as the supplemental documentation from github discussed later in this video. Many of the slides have additional comments and detail in the handout notes.

Disclaimers & Notices

- All sample scripts described herein are licensed under the Apache License, Version 2.0. See module headers for details.
- Sample scripts and libraries are provided as examples. Consoli-Solutions is not responsible for errors.
- Sample scripts and libraries are available for free when used by single customers for their own internal use. Use of these scripts and libraries in the manufacturing process, configuration of equipment for other customers, or providing services requires additional licensing.
- Consoli Solutions is an LLC

The legal disclaimer is here for you to read.

Contact

Jack Consoli

jack@consoli-solutions.com

https://github.com/jconsoli

Questions, comments, and feedback are always appreciated and I am available for consulting.

YouTube Videos In This Series

Simplified Brocade FOS API: Getting Started
https://www.youtube.com/watch?v=BWz7L0QOtYQ

Simplified Brocade FOS API: Capture Data and Reports
https://youtu.be/n9-Eni_AFCg

Simplified Brocade FOS API: Configuring Switches
https://www.youtube.com/watch?v=WGxXZrvhG2E

Simplified Brocade FOS API: Zoning
https://youtu.be/x1OvuRZRdA8

Scripts & Documentation: https://github.com/jconsoli

Follow the github link at the bottom of the page for a copy of this presentation and the scripts described herein. The last section of the Getting Started video discusses common features and inputs for the sample scripts.

Additional Notes

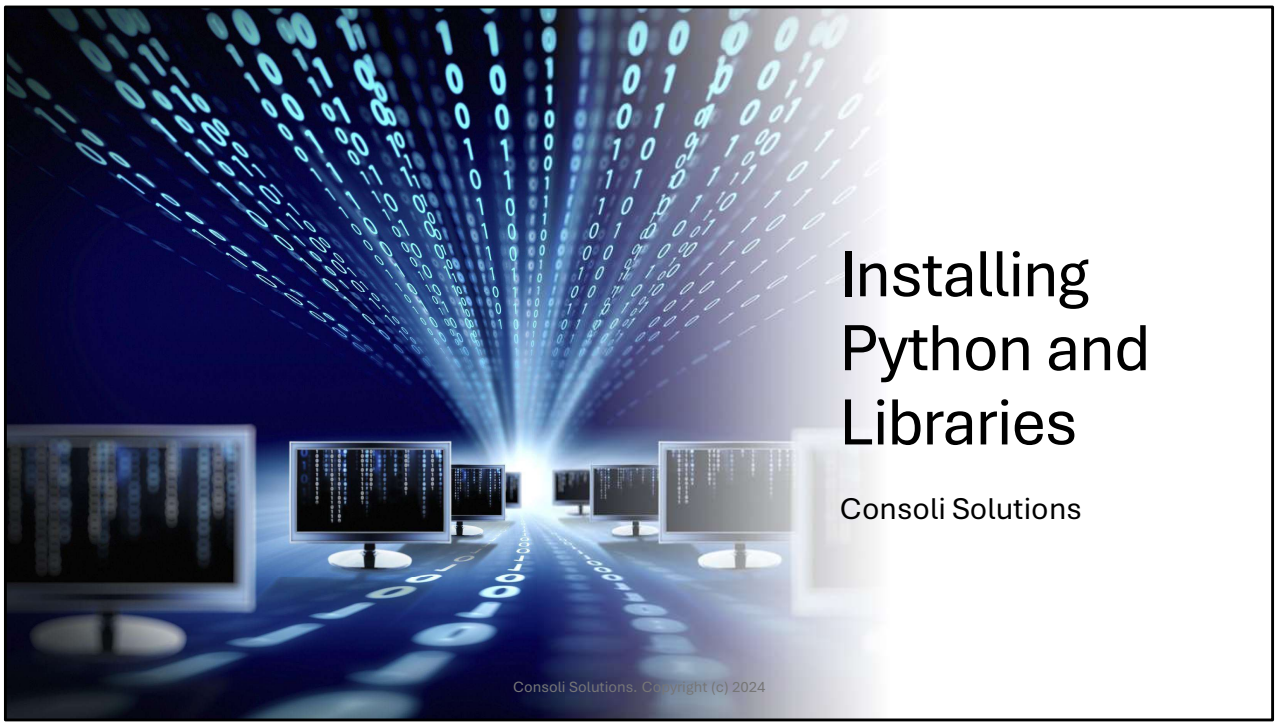
Simplified Brocade FOS API: Getting Started
Discusses how to download Python, what libraries are needed, how to install libraries, some tips for Python newbies, a few FOS settings you need to be aware of, an overview of the libraries I created to simplify scripting tasks, and an introduction to the sample scripts.

Simplified Brocade FOS API: Configuring Switches
Discusses a few nuances of configuring logical switches, Examples include: how to use sample scripts to configure logical switches from a workbook, and how to use an existing switch as a template for creating and configuring logical switches.

Simplified Brocade FOS API: Capture Data and Reports (This video)
Includes examples on how to collect data from switches, generate a general report, a

comparison report, a MAPS report, and an extensive nodefind utility.

Simplified Brocade FOS API: Zoning
Zone from a workbook, restore zoning from a previous data capture, merge zoning databases from multiple fabrics, and zone from CLI



Python has become the scripting language of choice for automation. The Brocade RESTConf API works with any language that supports APIs, but most API scripts are written in Python. All of the API scripts I've seen from Brocade and other vendors, including mine, are written in Python. If you haven't done so already, you should install Python.

Install Python & Upgrade pip

Download the Python interpreter:
https://www.python.org/downloads

Check Python Version
```
py --version
```

pip Update for Windows
```
py -m pip install --upgrade pip
```

pip Update for All Else
```
python3 -m pip install --upgrade pip
```


Minimal testing was done prior to Python 3.7, so that is the lowest version of Python I recommend. As a practical matter, 3.7 is very old. I did most of my testing at 3.12. I did run into some issues with older versions of pip so my advice is to execute the upgrade command.

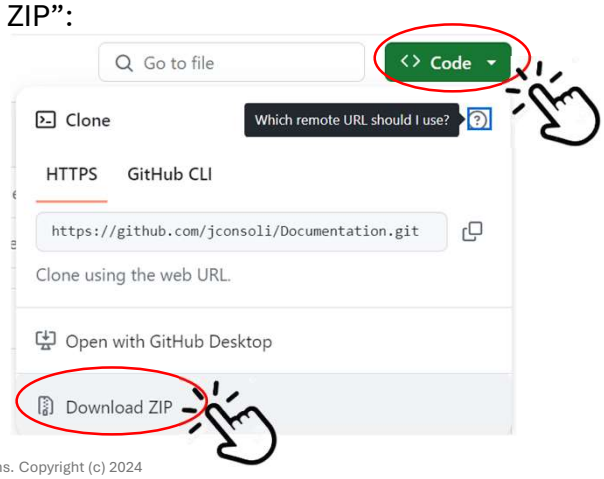
With a typical Python install, the shortcut "`py`" is configured. If your system can't find executable for "`py`", try typing out "`python`" or "`python3`".

FOS Specific Libraries & Samples

From:
https://github.com/jconsoli

Download a copy of:
- brocade-rest-api-examples
- brocade-rest-api-applications
- Documentation
- brcddb
- brcdapi

Click on “Code” and select “Download ZIP”:



The FOS specific libraries can be found on my github site. I’m only going to illustrate how to download the supplemental documentation. In addition to user documentation, this documentation includes tips and details for developers.

Use the same process to download the other folders. We’ll talk about what to do with them on the next slide. The last slide of this presentation illustrates how the FOS specific libraries help simplify programming tasks.

Additional Notes:

The documentation folder includes “FOS API SDK Documentation”, also referred to as “supplemental documentation”. In addition to documentation on the sample scripts and libraries, it contains notes for developers of things I ran into that weren’t clear. As I ran into things I thought could be better documented, I put it in here. It includes a discussion of how to correlate a login to a physical port, tips on what to use for database keys, etc. Some notes have nothing to do with FOS or Python. For example, I discovered Excel has some restrictions regarding sheet names used with hyperlinks in Excel, so I put that in here.

Where To Put Libraries, Samples, and Documentation

Put these wherever you want:
- brocade-rest-api-examples
- brocade-rest-api-applications
- Documentation

Put these in a folder with path access:
- brcddb
- brcdapi

A PyPI package for the libraries, brcdapi and brcddb was a work in progress when this video was created.

For now, copy brcddb and brcdapi to your Lib folder or any folder with path access. Unix users will need to make sure the execute attribute, –x, is set (`chmod`) for all files. All folders must have write access.

The difference between the examples and the applications is probably only significant to developers. I expect most people will put the contents of these folders all in one folder along with the documentation.

Additional Notes:

“brocade-rest-api-applications” and “brocade-rest-api-examples” are sample scripts. The applications use both the database library, brcddb, and the driver library, brcdapi. The examples only use the driver library. I doubt that difference will matter to anyone. Since I’m developing the software, I keep them in separate folders. You can organize them anyway you want. I expect most people will have a folder just for FOS scripts.

“brcdapi” and “brcddb” are libraries. A package for pip install and registration with PyPi was a work in progress when I created this video. For now, you will need to copy them to wherever you have a path to libraries. More on that on the next slide. Since the libraries are referenced by name, the folder names must remain “brcdapi” and “brcddb”.

Typically, Windows users do not need to be concerned with file permissions. For all other operating systems, execute access is required on all sample scripts and libraires. Write access is also required for all folders because Python wants to put a cache, a folder named "`__pycache__`", in each folder.

Validate Access to Libraries

Check Libraries (lib_check.py is in brocade-rest-api-applications):

```
py lib_check.py
```

Sample Output:

```
Python library path(s):
  * C:\Users\Jack Consoli\Documents\cs\scripts\brocade-rest-api-applications
  * C:\Users\Jack Consoli\AppData\Local\Programs\Python\Python312\Lib
  * …
Standard Python Libraries & Open source python libraries
  * …
FOS API driver – brcdapi & FOS API database – brcddb
  * …
```


Access to the libraries is the most common issue I help people with. I recommend running lib_check.py to check the library paths and determine what libraries are needed before attempting to install any other libraries. The output should be self explanatory. The standard Python Lib folder is going to be "Python\Python*{version}*\Lib".

Additional Notes:

Python library path(s): This is a list of all library paths, in the order Python looks for them. "`C:\Users\Jack Consoli\Documents\cs\scripts\brocade-rest-api-applications`" is the folder I use for script development and where I ran `lib_check.py` from. Typically, the folder you run your scripts from is included in the PATH. "`C:\Users\Jack Consoli\AppData\Local\Programs\Python\Python312\Lib`" is folder for libraries created by the Python installer.

Standard Python Libraries & Open source python libraries: I broke out Python org libraries and open source libraries separately because in the early days of Python, particularly with installing them, that made a difference. Today, it shouldn't matter.

All of these libraries can be installed using pip –install. See next slide.

FOS API driver – brcdapi & FOS API database – brcddb:
These are the libraries needed for the sample scripts. Although you can put the bradapi and brcddb folders anywhere there is a path to them, I recommend either the standard library folder or the folder where you do script development from. If you need to add your local folder to the paths, just search the web for "add library path python".

Install Standard & Open Source Libraries

```
pip install jdcal
pip install et_xmlfile
pip install cryptography
pip install urllib3
pip install openpyxl
pip install deepdiff
```


The last time I installed Python, these were the Python and open source libraries I needed to add. Your installation may be a little different. This is why I recommend running lib_check first. If you need a standard library or open source library, the procedure to install them is the same.

Simply copy and paste each pip install command you need into a shell or command prompt.

Sample Script Modifications

Windows (No need to make any changes):
#!/usr/bin/python
-*- coding: utf-8 -*-

Unix, MacOS (This is probably what you want):
#!/usr/local/bin/python
-*- coding: utf-8 -*-

z/OS & OS/400:
Note that all files, samples and libraries, must also be converted to EBCDIC
#!/usr/local/bin/python
-*- coding: ebcdic -*-

At the beginning of each sample script are two comment lines. The first is referred to as the “shebang” line. This should be in any script you write, especially for non-Windows environments. The second is rarely used because utf-8 is the default character encoding for all operating systems except z/OS and OS/400. Since these scripts are also supported on those operating systems, I explicitly added the encoding line as well.

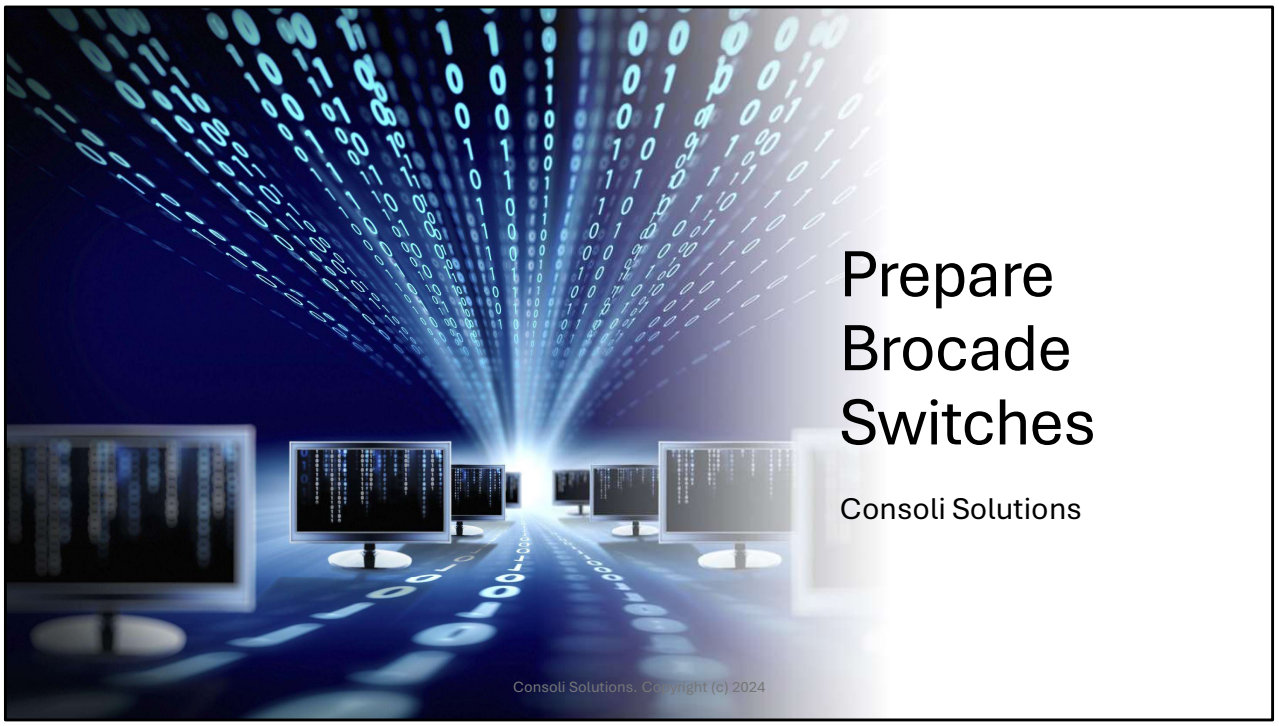
These comment lines are in the sample scripts, “brocade-rest-api-applications” and “brocade-rest-api-examples”, only. The library files in “brcdapi” and “brcddb” should not be edited.

Windows users shouldn’t need to make any changes. Typically, Unix, Unix variants, MacOS, z/OS, and OS/400 use the local space. Check with your system admin. If you don’t have a system admin and your not sure what to do, the recommended edits are probably what you need. All of the sample scripts must have the correct shebang and coding comments. Use VI, notepad, or any plain text editor.

Additional Notes:

zLinux should be utf-8 but I ran into a display issue with a customer. After some research we discovered CODEPAGE 437/500 in the Hercules configuration file is required to properly display special characters. Useful reference: IBM RedBook "Practical Migration to Linux on System z", SG24-7727.

For z/OS and OS/400, in addition to changing the encoding scheme to EBDCIC in the sample scripts, all files, including the libraries, will also need to be converted to EBCDIC.



This concludes everything I think you need to know about installing and configuring your PC or workstation for Python. Now we need to talk about what you need to do with your Brocade switches so that you can use the API. All of this is discussed in greater detail in the supplemental documentation.

Determine If A Private Key Exists

To check to see if a certificate exists:

```
seccertmgmt show -all
```

Output:

```
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
```


I'm assuming everyone wants to use HTTPS with a self-signed certificated. Depending on the version of FOS you are using, HTTP may not even be allowed. A certificate from a CA is not discussed in this video.

The first thing you need to do is to check to see if your switches are configured for a self-signed certificate. From an SSH session with the management interface:

```
seccertmgmt show -all
```

Scroll down to the "Certificate Files" section of the output. Look down the "PVT Key" column to the "HTTPS" row. In this example, HTTPS has already been configured with a private key. I know this because "Exist" is in this cell. There is no need to do anything else. If this is "Empty", you will need to create a certificate.

Create A Self-Signed Certificate

You should only do this if a certificate is required. Enter the following command in an SSH session with the management interface:

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


Only do this if the certificate is “Empty”. Simply copy and paste this command into an SSH session command prompt. Of course, it’s a good idea to go back to the previous slide and validate that the certificate was in fact created.

API Configuration Settings

Enable the REST interface:

```
mgmtapp --enable rest
```

To change the number of permitted sessions:

```
mgmtapp --config –maxrestsession 3
```

Sample rate:

```
mgmtapp --config -samplerequest 240
```

Enable the Keepalive:

```
mgmtapp --enable keepalive
```


Unless you plan on allowing multiple REST sessions to access the switch at the same time, just copy and paste these commands into an SSH session with the switch management interface and move on. Few people need to make any changes to these settings. There is a detailed explanation on throttling and these settings in the supplemental documentation and the Brocade REST API Guide.

Additional Notes:

With the first release of FOS that supported the API, the sample request was 32. I believe the default in FOS 9.2 is 120. Keep in mind that when FOS is upgraded, default settings are only applied to settings that have yet to be configured. If you upgraded FOS, the old settings will remain.

In my own testing, I discovered occasions when 120 was limiting so I always set it to 240. Once a script exceeds the maximum sample request, a busy status is returned for all subsequent requests and the script must pause and re-drive the request for the remainder of the login session. This results in very slow performance. The brcdapi driver handles the pause and re-drives the request. I don't recall the exact number, but it will fail after 5 or 6 attempts to re-drive the request.

I don’t know where the sample request time can be checked in FOS via the CLI. I’m sure it’s somewhere, but I always recommend setting it. It cannot be set via the API, or at least not with FOS 9.2.

Check Configuration Settings

```
default_switch:FID128:admin> mgmtapp --show
  REST Configuration:
       Interface State: Enabled
       Effective Protocol: HTTPS only
       HTTP State: Enabled
       Session Count: 3
  HTTPS Configuration:
       KeepAlive : Enabled
       KeepAliveTimeout : 15sec
```

- Must be "Enabled"
- Typically "HTTPS only"
- Must be enabled
- Typically 3 but can be as high as 10
- Typically "Enabled"
- Always 15 (may be adjustable in future releases of FOS)

You can also use app_config.py in brocade-rest-api-examples to check these settings. If you are a developer and will be setting breakpoints in your code, make sure you disable the KeepAlive.

Use this command to validate the settings made on the previous slide. You can also use app_config.py in brocade-rest-api-examples to check these settings. You can also make changes with app_config.py as well. Of course, you can't use an API script to enable the API if it's disabled. I'm assuming most people will make these changes and validate them using an SSH session to the management interface during initial switch configuration.

The reason I'm taking the time to discuss app_config.py is because developers will need to disable the KeepAlive when setting break points. I have a batch file I run that includes using app_config.py to disable the KeepAlive before I begin any development work and another batch file that re-enables it when I'm done. I'm assuming most developers will do something similar.

Additional Notes

1. Obviously, the API interface must be enabled before you can use it.
2. Security, HTTP or HTTPS, must be configured properly.
    1. HTTP is rare. Nearly all interfaces are configured for HTTPS with a self signed certificate.

    2. I believe a certificate from a CA is possible but I've never tried it and I don't know of anyone who uses anything other than a self signed certificate.
1. The maximum number of Rest sessions must be set to meet your needs
    1. Sessions from SANnav are counted separately and therefore do not have to be accounted for when configuring the chassis for API use.
    2. The default, and minimum number, is 3. Three is adequate for most people. Needing more is for advanced users and beyond the scope of this video.
2. KeepAlive
    1. When enabled, the API login session will expire if a request has not been sent to the switch since the last response from the switch within the KeepAliveTimeout time. The time it takes to process a request and respond is not counted.
    2. Although not required, enabling the keepalive is recommended and may be required by your organizations security practices.

Validate Access to the Switches

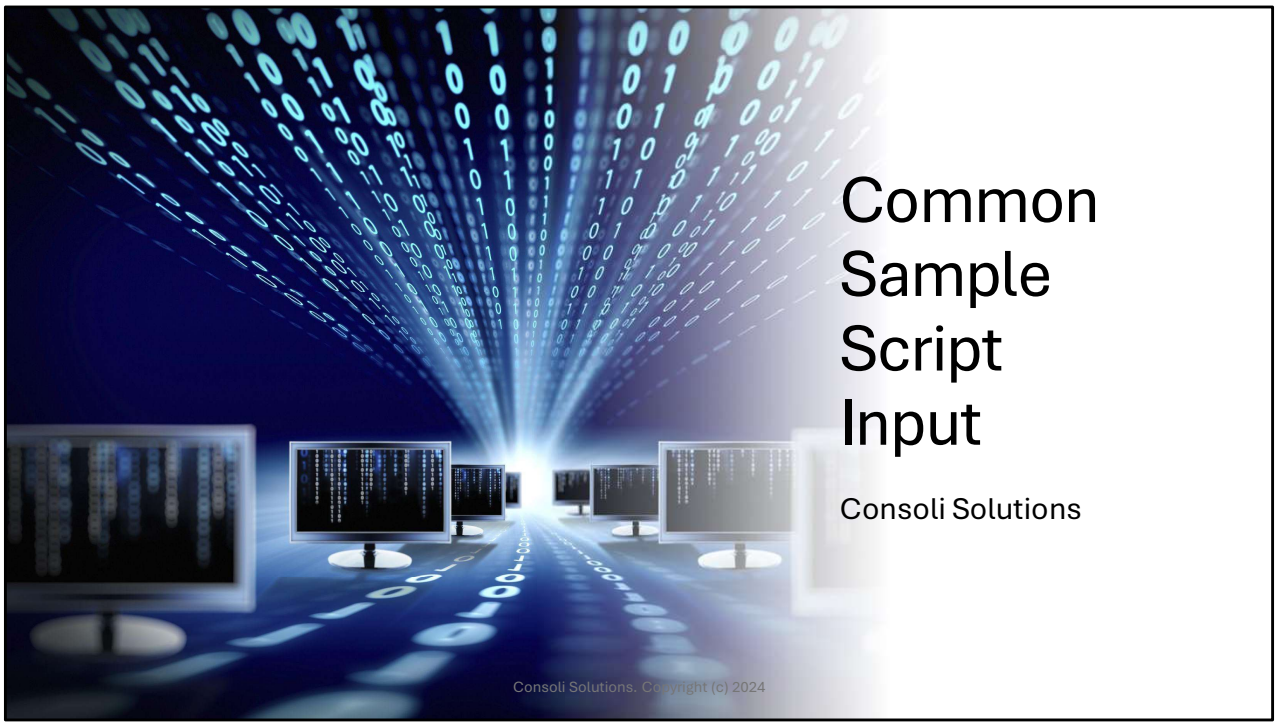
```
py login_test.py -ip xxx -id admin -pw pw -log _logs
```

The login_test.py script attempts to log in and log out. It doesn't do anything else. To do this, it needs access to the driver library, brcdapi, and access to the switch. Running this test, therefore, validates access to the library, the network is configured properly, and the switch is configured properly for API access.

The login_test.py script attempts to log in and log out. It doesn't do anything else. To do this, it needs access to the driver library, brcdapi, and access to the switch. Running this test, therefore, validates access to the brcdapi driver library, the network is configured properly, and the switch is configured properly for API access.

For most environments, you will need to hard set the IP address on your PC or workstation to be a unique IP address on the same subnet as the switches. A web search should return several results on how to do that. If your organization has a network admin, check with the network admin first.

Once you get past this test, everything else should be smooth sailing.



So as not to have to repeat discussing the common sample script inputs with each sample script video, I'm going to cover them now.

Common Options

*Option	Description
-ip	IP address of chassis.
-id	User ID
-pw	Password. The password is not recorded or displayed anywhere other the command line or workbook the password is contained in.
-s	Optional. Security, HTTP or HTTPS mode. HTTPS is used by default with a self-signed certificate.
-d	Optional. No parameters. When set, additional debug information and all content sent and received to/from the API, except login information, in human readable format is printed to the log.
-log	Optional. All messages are written to a log and optionally printed to STD OUT. Messages not printed to the log are typically additional information for error scenarios. The log file is named "log_xxx.txt" where "xxx" is a time and date stamp. All entries in the log are also time stamped. The –log option allows you to specify a folder to put the log files in. By default they are placed in the same folder where the script was run from.

-ip, -id, -pw, -s: Only used with scripts that access a chassis. These are standard login credentials. HTTPS with self signed certificates is the default. I doubt anyone will ever use the –s option to disable it, but it's there if you need it.

-d: Adds debug information and prints all requests and responses to the log. Normally, this option is not used because it clutters the log and console output with information that is only useful to developers. If you report a repeatable problem to me, I may ask you to re-run the script with this option and send me the log file.

-log: Except for lib_check.py, all scripts write messages to a log function that echoes log messages to the console. Error detail is not always echoed to the console. A log file with the prefix "_log_" + a time stamp in msec is created. The timestamp is in msec so all log files will have a unique name. By default, log files are placed in the same directory where the script was run from. This option allows you to specify a folder to put the log files in to keep your script folder from getting cluttered with log files. For developers, the log library includes two main functions: log() and exception(). The exception() function inserts a stack trace, calls log(), and then flushes the log file cache.

Common Options (Continued)

*Option	Description
-nl	Optional. No parameters. When set, a log file is not created. Messages are still printed to the console. The default is to create a log file.
-sup	Optional. No parameters. Suppress all output to STD_IO except the exit code and argument parsing errors. Useful with batch processing where only the exit status code is desired. Messages are still printed to the log file.
-scan	Optional. No parameters. Scan switches and files for fabric information. No other actions are taken.
-fid, -did, -p	In many instances, these options permit a CSV list, a range, or * (all). A range can be embedded in the CSV list. For example: "-p 3/0,4-5/0-2" specifies port 3/0, 4/0, 4/1, 4/2, 5/0, 5/1, 5/2.
-eh	Optional. Extended help. Available for scripts with complex input options.
-h	Prints help to the console. This is handled by a Python library before the log file is setup. These messages are printed to the console only.

-nl: Disables the creation of a log file, but messages are still printed to the console

-sup: Suppresses printing all messages, except the exit code, to the console. This may be useful with batch processing. This option suppresses printing all but the exit code the console.

-scan: Only supported with applications that read a captured data file. This just tells you what's in the file and exits. No other processing is performed. If login credentials are also supplied, the output will also include information about the chassis. Sample output is on the next slide.

-fid, -did, -p: Only supported with applications that need FID, DID, or ports for input. Where multiple FIDs, DIDs, or ports makes sense, ranges and CSV lists are supported.

-eh: Only available with scripts that have complex input options. These messages are also printed to the log. No other processing takes place.

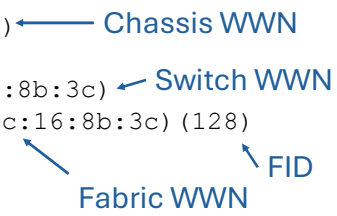
-h: Except for lib_check.py, all sample scripts print help information to the console and exit with the standard –h option. A log is not created.

Sample –scan Output

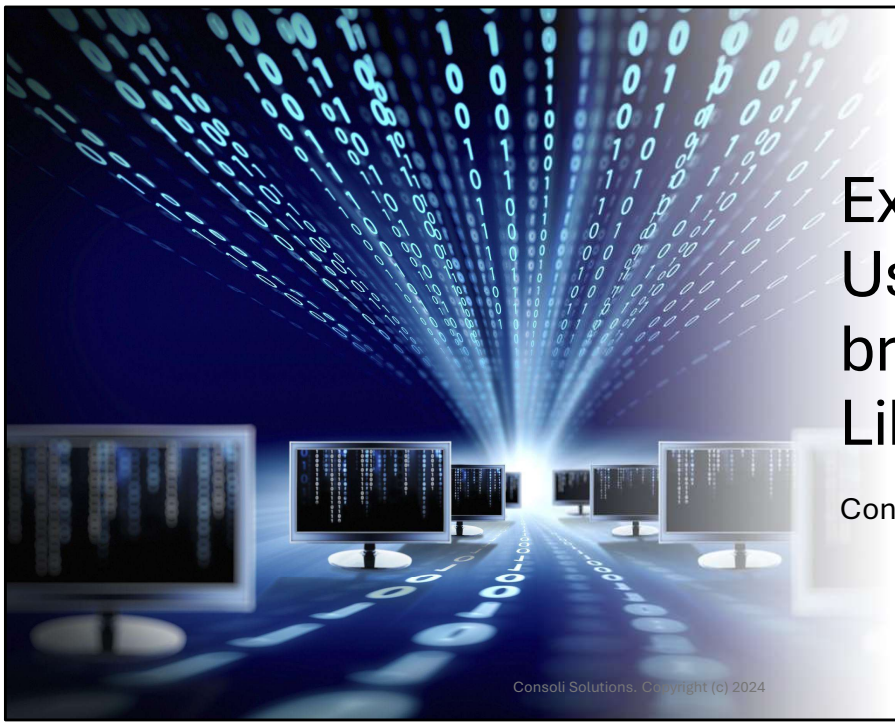
```
Scan of _capture_2024_05_15_08_00_46/combined.json
_____________________________________________________
Chassis: GSH_Test_Chassis (10:00:c4:f5:7c:16:8b:7b)

  Switch: default_switch DID: 1 (10:00:c4:f5:7c:16:8b:3c)
    Member of Fabric: REPLICATION_B (10:00:c4:f5:7c:16:8b:3c)(128)
    Zone Configurations:
      zone_cfg_0 (effective zone configuration)

  Switch: Base_Switch DID: 35 (10:00:c4:f5:7c:16:8b:3d)
    Member of Fabric: 10:00:c4:f5:7c:16:8b:3d (60)
    Zone Configurations:
      None
```



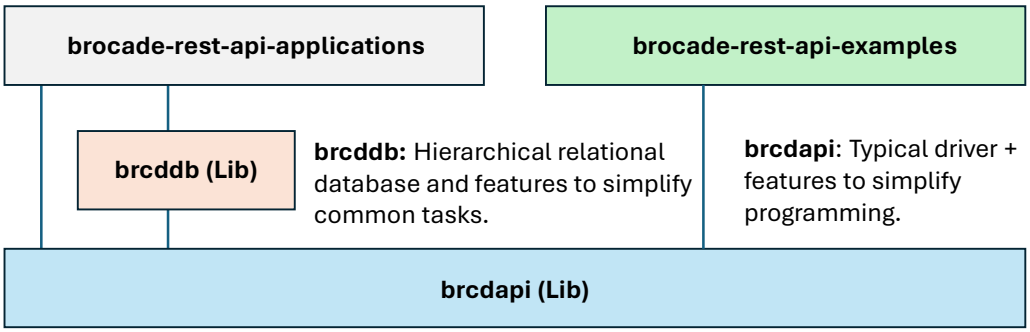
Chassis, switch, and fabric names do not have to be unique, so for scripts that require it, they are identified by their WWN. Each logical switch in each chassis is listed with the fabric the logical switch is a member of and the zone configurations defined in that fabric.


Example
Using
brcddb
Library
Consoli Solutions
Consoli Solutions. Copyright (c) 2024

Architecture

brocade-rest-api-applications: Examples using the brcddb database and features.

brocade-rest-api-examples: Examples using the brcdapi driver.



This is a high level overview of the architecture. Starting from the bottom, the brcdapi library is a traditional driver with some value added functions. The brcdb library is a hierarchical relational database. The brocade-rest-api-examples only use the driver. They were originally intended as examples for developers, but some are useful as stand-alone utilities. Most of them have a command line interface. The brocade-rest-api-applications rely heavily on the brcddb library and are discussed in the other YouTube videos in this series.

Additional Notes

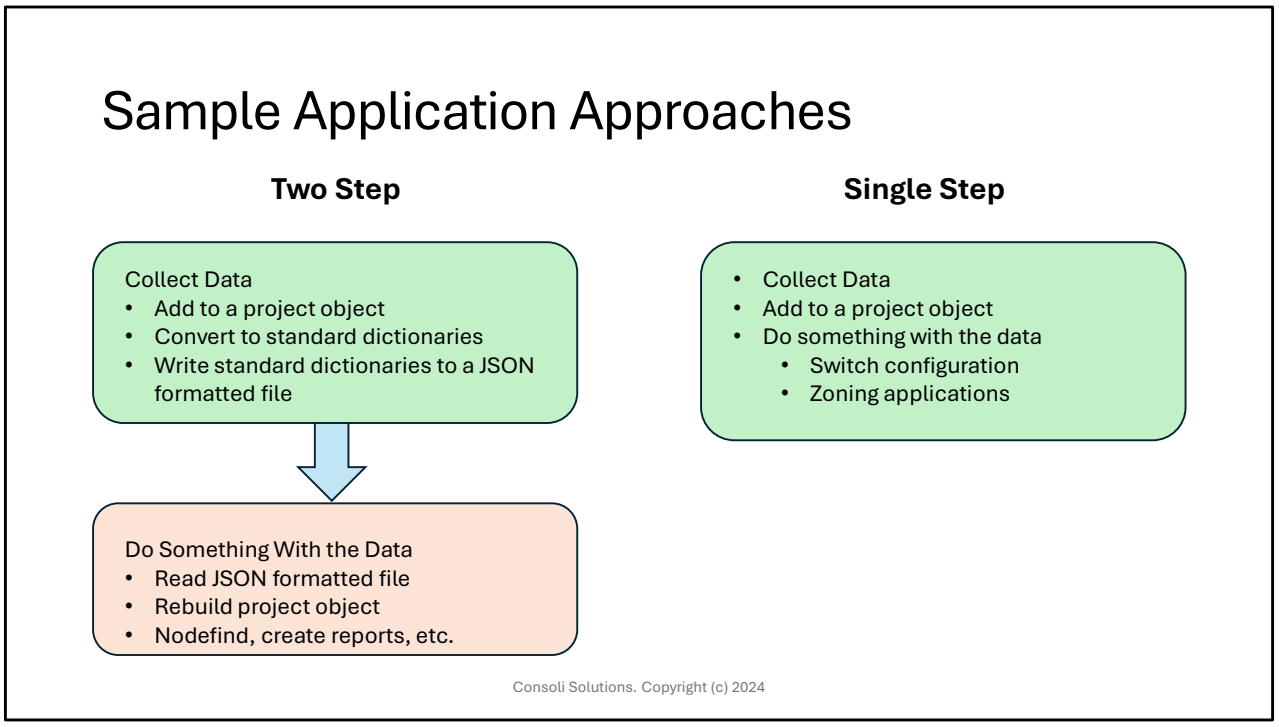
A list of each module and description is contained in “FOS_API_SDK_Documentation”

The libraries allow script developers to take a systems programming approach when developing scripts.

The brcdapi library includes a single interface to the switch API. It automatically pauses and then re-drives request when the chassis, fabric, or switch is busy. There is also a debug mode that allows you to automatically write all GET responses to a file. There is also a read mode in which logins are spoofed and GET requests are read from

the previously written files. This not only alleviates the need for a physical switch during some script development but it's much faster to read data from a file than to request it from a switch. This is not a full switch emulator. It only works with GET. The file names are a hash of the IP address, URI, and FID. If they same GET request is issued twice, the previous file is overwritten.

Since brcdapi has a single interface to the FOS API, it was easy to add a debug mode that prints request/response detail to the log. More on this later.



There are two types of applications. Some take a two step approach whereby data is captured from a chassis, written to a file, and then read back using a different script. Others combine these two steps. Some are a hybrid. This is why there is a –scan option.

Additional Notes:

Two Step:

- Collect Data
  - Add to a project object
  - Convert to standard dictionaries
  - Write standard dictionaries to a JSON formatted file
- Do Something With the Data
  - Read JSON formatted file
  - Rebuild project object
  - This approach is typical of the node find, report, and statistics gathering applications.

Single Step

- Skips converting data to a JSON, written to a file, and then read back and rebuild the project object
- This approach is typical of applications that modify configurations such as zoning and switch configuration applications

Some applications are a hybrid

How FOS Specific Libraries Simplify Tasks

```
uri_for_capture = (
    'running/brocade-fibrechannel-switch/fibrechannel-switch',
    'running/brocade-interface/fibrechannel',
    …
)
session = api_int.login(user_id, pw, ip_addr, sec, proj_obj=proj_obj)
api_int.get_batch(session, proj_obj, uri_for_capture)
for fab_obj in proj_obj.r_fabric_objects():
  alias_l = brcddb_fabric.aliases_by_name(fab_obj, storage_xyz_??', 'wild')
  storage_alias_obj_l = [fab_obj.r_alias_obj(alias) for alias in alias_l]
  port_obj_l = obj_convert.obj_extract(storage_alias_obj_l, 'PortObj')
  zone_obj_l = obj_convert.obj_extract(storage_alias_obj_l, 'ZoneObj')
  all_alias_obj_l = obj_convert.obj_extract(zone_obj_l, 'AliasObj')
wb = excel_util.new_report()
report_port.port_page(wb, None, 'Logins', 0, 'login', port_obj_l)
excel_util.save_report(wb, 'Sample_Report')
```


This is not intended as a coding tutorial. It is intended to illustrate how using objects with a hierarchical relational database and utilities simplifies programming tasks. For simplicity, I truncated the list of URIs and removed uninteresting code for this discussion.

Assume I want to migrate off a storage array. To do that, I need to know what servers are zoned to the storage. For the physical cable plant team, I also need to know what ports the storage is connected to.

uri_for_capture: Data needed to identify where logins are and the zone database.

session: Login session object.

get_batch(): Automatically determines which URIs are chassis level and which are logical switch level. A full URL is built and sent to the chassis. Since I didn't pass it a list of FIDs to operate on in this example, switch level URIs will be sent to each logical switch in the

chassis. The response from the chassis is processed. Objects are automatically created, attached to the project object, and data from the chassis is sorted out and attached to the objects the data belongs to.

proj_obj.r_fabric_objects(): returns a list of fabric object found in the previous line of code.

aliases_by_name(): returns a filtered list. In this example, a list of alias names matching the wild card search "storage_xyz_??" would be returned. RegEx and exact matching is alsp supported.

storage_alias_obj_l: This is an example of using a Python list comprehension to build a list of storage alias objects using the alias names.

port_obj_l: The function obj_extract() takes a list of objects and returns a list of corresponding objects of the type specified. In this example, I passed it a list of alias objects and ask for a list of associated port objects. Duplicate objects are removed. So now I have all the ports where the storage is connected.

zone_obj_l: I'll also need to know everything zoned to the storage, so I'll use obj_extract() again to get all the associated zone objects.

all_alias_obj_l: For a list of server aliases, just subtract storage_alias_obj_l

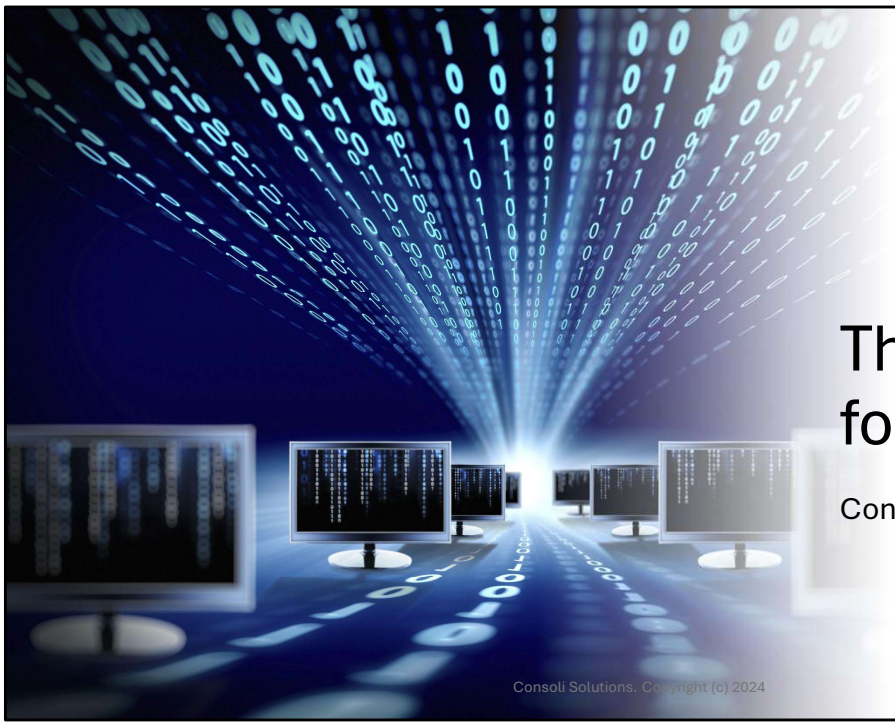
new_report: To create a workbook to pass along to the cable teams, I just get a workbook object.

port_page: Add a pre-canned port page with detailed port configuration information to the workbook.

save_report: Then save the report.

I actually wrote the code to do this before stripping it down to a simple example. I know where to copy and paste most of this, so for me, it took just 2 hours. As with

most software, if I spent more time thinking about what I was writing before I wrote it, I could have done it in even less time.



Thank you for watching

Consoli Solutions