Simplified Brocade FOS API: Configuring Switches

Consoli Solutions

Description:

This video discusses how to reconfigure a chassis from a workbook and how to use a chassis or logical switches as templates using the FOS RESTConf API. Enterprises with a cyclical business model may want to these scripts to quickly reconfigure their SANs.

Keywords: FOS API restore configure switch

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

Although primarily intended to use other switches as a template, this script can also be used to restore or repurpose a chassis as well.

restore.py

The intended purpose is to use previously collected data from a chassis as a template to be applied to a different chassis. The fabric map, -fm, allows you modify the FID, names, and DID in the target switch. By selecting all options, it can also be used to restore a chassis.

Nomenclature:

Restore	Refers to the reference chassis (chassis being restored from).
Target	Refers to the chassis being rebuilt.

The initial intent of this script was to use a chassis or logical switch as a template to configure another chassis or logical switch. A mapping option allows you to change the fabric name, fabric ID, switch name, and domain ID.

The nomenclature used with this script is:

Restore: Refers to the reference chassis (chassis being restored from).

Target: Refers to the chassis being rebuilt.

Additional Notes

The intent is to restore everything exposed in the API by simply reading (GET requests) all branches from the target switch and make the same request with a PATCH of all changes. That doesn't work on everything because some changes require additional processing or are changed via the operations branch. Special handling for MAPS, switch configuration, and non-default user accounts has been implemented.

Port long distance settings was not implemented in FOS as of v9.1.1c via the API so it was implemented via the CLI. The assumption is that the same login credentials used for the API work with the CLI as well.

Keep in mind that by default, only data required for the report is captured. This is why I typically capture data with the “-c *” option.

WARNING: It’s not possible to test all configurations. You need to validate switches before putting them into production.

Compared to configupload/configdownload

- The target switch does not have to be the same type of switch as the restore switch
  - Useful for switch upgrades
  - Use any switch as a template
- Greater flexibility in what to restore
- Modify the switch name, DID, fabric name, and FID
  - Useful for using a switch as a template
- Best effort
  - Errors are reported but the script continues to run

Although there is some overlap with the configdownload and configupload FOS commands, the restore.py script is intended as a template.

Best effort: For example, ports that are present in the restore switch but don't exist in the target switch would not be moved and therefor reported as an error. The script would continue to run and attempt to make all other changes.

restore.py: Input Parameters

*Option	Description
-ip, -id, -pw	Login credentials of the target chassis
-i	Captured data file from the output of capture.py, combine.py, or multi_capture.py containing data from the restore chassis.
-wwn	WWN of chassis in the input file, -i, to be restored to the target chassis specified by the login credentials. Use –scan to determine the WWN.
-p	Defines what to be restored. See next slide.
-fm	Optional. FID Map. More on this shortly.
-scan	Optional. No parameters. Scan file specified with –i for fabric information. No other actions are taken.
-eh	Extended help.

-eh: I recommend running the script with this option before attempting anything else.

restore.py: -p Actions

*Option	Description
*	**All** (full restore)
vfc	**Virtual Fabrics Clear**. Virtual fabrics will be enabled if necessary; however, virtual fabrics will not be disabled. Delete all non-default logical switches. Set all ports in all FIDs, including the default logical switch, to the default configuration.
vfs	**Virtual Fabrics Switches**. Create any logical switch that doesn't already exist. Unless specified otherwise with the -fm option, the FID, DID, fabric name, and switch name will match that of the restore chassis.
vfp	**Virtual Fabrics Ports**. Remove any ports that do not belong in the target logical switch. Add any ports not present that do belong to the logical switch.
c	**Chassis**. All chassis settings except virtual fabrics, users, and security features. It does include enabling FCR but no other FCR configuration settings are available at this time.
s	**Switch**. All logical switch configuration settings except MAPS, user friendly switch name, or the domain ID. The user friendly name and DID are set with the vfs parameter.

The –p option is used to define what gets restored to the target switch. When specifying a CSV list of actions for use with –p, the order doesn't matter. Certain things, such as moving ports to a logical switch requires the switch to be created first, so the order of actions is pre-determined. Redundant actions are removed. This is roughly the order in which actions are taken. Diving into these is beyond the scope of this video.

restore.py: -p Actions, continued

*Option	Description
p	**Port**. All port configuration settings except FCR and FCIP.
maps	**MAPS**. All MAPS custom (non-default) rules and policies. Groups are not modified.
u	**User accounts**. Creates non-default users only. Existing user accounts in the target chassis are not deleted and passwords for existing accounts are not modified.
z	**Zoning**. Restores the zoning configurations for each logical fabric
ze	**Zone Configuration Enable**. Activates the zone configuration that was active in the restore fabric. If "e" is specified, the switch enable action occurs first.
e	**Enable**. Enable all the switches and ports in the target switch that were enabled in the restore switch.

Again, this is here just for reference.

restore.py: -fm Option

Use this option to modify the target FID, fabric name, switch DID, and switch name.

The operand is a CSV list embedded in a semicolon separated list. When no value is specified, the value from the restore chassis is used. The values are (by index into the CSV list):

Index	Description
0	The restore chassis FID.
1	The FID to be created or modified in the target chassis.
2	*The fabric name to be defined in the target chassis.
3	*The switch DID in decimal to be defined in the target chassis.
4	*The switch name to be defined in the target chassis.

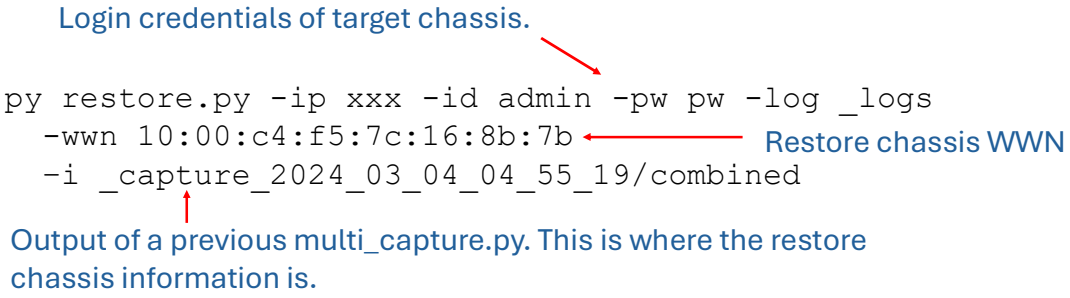
*Only applicable when vfs is specified with the -p parameter

Use the –eh command line option for a full discussion of the FID map, -fm, option. This will be a little easier to understand with the use case examples that follow on subsequent slides.

When using switch as a template to create another switch, you typically will want to change the switch name and domain ID. You may also want to use a different FID number and give the fabric a different name. If you're copying MAPS policies, you typically want to specify the fabric you want to copy from as well as the fabric to copy to. This is the purpose of the –fm option.

restore.py: Common Parameters

Anytime you see "`-common`" in the subsequent examples, it is a substitute for –ip, -id, -pw, -i, -wwn, and –logs.



Rather than clutter the examples with repetitive options, substitute these parameters every time you see –common.

TIP: I usually use the restore_all.py script to generate the correct command line syntax. We'll cover that later.

restore.py: Extended Help & Scan Examples

Extended Help

```
py restore.py –eh -log _logs
```

Scan

```
py restore.py -common
  –i _capture_2024_03_04_04_55_19/combined –scan
```


Extended help is as simple as running the script with –eh. Unlike –h, the script actually runs and sets up the log file so messages are written to the log file and echoed to the console.

I always recommend using login credentials when using the –scan option, but they are not required.

Additional Notes

-eh: The “-h” parameter is handled by an input parameter parsing li
brary that is included with the standard Python installation. That library prints help messages to the console and then exits. The code that sets up the log never gets executed. Since “-eh” is handled by the restore.py script, a log file is opened and all help messages are printed to the log with the echo to console option enabled. This is why the –log option is specified on the command line when using -eh. The –eh only prints help messages and exits. No other parameters are required. If additional options are specified, they are ignored. I often have other parameters because I’m building the command line and need some help while I’m building it. In most cases, once I nail down the syntax I put it in a batch file.

restore.py: Copy MAPS Rules, Groups, and Policies

Copy the custom MAPS rules and policies from FID 1 of the restore chassis to FIDs 1, 2, and 3 in the target chassis.

```
py restore.py -common -fm 1,1 1,2 1,3 -p maps
```

From FID 1 to FID 1, FID 2, and FID 3

Is also the same as

```
py restore.py -common -fm 1,1-3 -p maps
```

FID 1 is the restore from FID

Map everything in FID 1 of the restore chassis to FIDs 1, 2, and 3 of the target chassis.

This is an example on how to copy the MAPS policies from one switch to another. Note that there are only two parameters in the FID map, -fm, the restore FID and the target FIDs. Since nothing is being done with virtual fabrics, there is no point in having a full CSV list of all parameters. Any additional items in the CSV list would just be ignored.

I can also specify a range of FIDs.

Additional Notes

Since the only parameters changing are the maps parameters, “-p maps”, the MAPS groups, rules, and polices for FID 1 of the restore switch will replace the MAPS group, rules, and policies of the logical switches identified as FIDs 1, 2, and 3 of the target chassis. No other action will be taken. See next slide for additional comments regarding the FID range. You can also use a CSV list for target FIDs, but so as not to confuse the command line interpreter, you will need to enclose it in quotes.

restore.py: Copy MAPS Rules, Groups, and Policies

Copy the custom MAPS rules and policies from FID 1 of the restore chassis to FIDs 1, 2, and 3 in the target chassis.

```
py restore.py -common -fm 1,1;1,2;1,3 -p maps
```

Is also the same as

```
py restore.py -common -fm 1,1-3 -p maps
```

FID 1 is the restore from FID

Map everything in FID 1 of the restore chassis to FIDs 1, 2, and 3 of the target chassis.

This is an example on how to copy the MAPS policies from one switch to another. Note that there are only two parameters in the FID map, -fm, the restore FID and the target FIDs. Since nothing is being done with virtual fabrics, there is no point in having a full CSV list of all parameters. Any additional items in the CSV list would just be ignored.

I can also specify a range of FIDs.

Additional Notes

Since the only parameters changing are the maps parameters, “-p maps”, the MAPS groups, rules, and polices for FID 1 of the restore switch will replace the MAPS group, rules, and policies of the logical switches identified as FIDs 1, 2, and 3 of the target chassis. No other action will be taken. See next slide for additional comments regarding the FID range. You can also use a CSV list for target FIDs, but so as not to confuse the command line interpreter, you will need to enclose it in quotes.

restore.py: Copy MAPS Policies Continued

Copy the MAPS policies from FID 1 of the restore chassis to all FIDs in the target chassis except FID 128. This is a convenient way to replicate custom MAPS definitions to all FIDs without having to determine the target FIDs in advance.

```
py restore.py -common –fm 1,1-127 -p maps
```


Copy the MAPS policies from FID 1 of the restore chassis to all FIDs in the target chassis except FID 128. This is a convenient way to replicate custom MAPS definitions to all FIDs without having to determine the target FIDs in advance. Except when creating logical switches, the target FIDs are automatically determined and only existing FIDs are used. A warning message indicating which FIDs do not exist are displayed, but there are no consequences.

restore.py: Don't Do This

```
py restore.py -common –fm 1,1-127 -p vfs,vfp
```

The script does whatever you ask it to do without question. Even if you're asking it to do something that doesn't make sense.

Above, the script was asked to create 127 logical switches in the target switch all matching FID 1 of the restore switch.

The power to do whatever you want also gives you the power to do a lot things you don't want to do.

In this example, a logical switch, FID 1, would be created in the target chassis with the same fabric name, switch name, and DID as the restore switch. It would then move all the ports matching the ports in the restore switch that was created as FID 1. The script would then create FID 2, which would succeed, but setting the switch name would fail because switch names must be unique in FOS. It would retain the default switch name defined by FOS when it was created. It would set the DID to the same DID used for the restore switch, which means it will have the same DID as FID 1. Then it would move all the ports that were just moved to FID 1 into the new logical switch, FID 2. The script would continue to create logical switches and move ports in this manner until the maximum number of logical switches supported for the target chassis was reached. There after, it would still attempt to create the remaining logical switches, but each attempt would fail. FOS may terminate the session due to an excess of failed API requests.

restore.py: Template for Virtual Fabrics

Repurpose a chassis by removing all port configurations and all logical switches. Recreate FID 1 in the target switch as FID 11, fabric name prod_odd_fab, switch DID 31, and name the switch switch_31. Recreate FID 3 in the target switch as FID 13, fabric name backup_odd_fab, switch DID 41, and name the switch switch_41.

```
py restore.py -common -p vfc vfs vfp s p maps e -fm
 1,11,app_1_odd_fab,31,switch_3 ; 3,13,backup_odd_fab,41,switch_41
```

Index	Description
0	Restore FID: 3
1	Target FID: 13
2	Target fabric name: backup_odd_fab
3	Target DID: 41
4	Target switch name: switch_41

I'm just going to go over the FID map for the second switch. Note the semi-colon between the two.

This example is a typical use case for the FID map when using a chassis as a template to create logical switches in another chassis. In this example, assume two logical switches, FID 1 and FID 3 exist in the restore chassis:

FID 1 Could be for local production traffic while FID 3 could be for extended disaster recovery. They could have different MAPS policies.

Additional Notes

In this example, the target chassis is being repurposed. The vfc parameter for –p means all ports will be set to the factory default and moved to the default switch. Then all logical switches, except the default logical switch, will be deleted. Vfs then tells the script to create FID 1 of the restore chassis as FID 11 in the target switch with the following attributes:

Fabric name: app_1_odd_fab

Domain ID 31
Switch name: switch_31

Similarly, as illustrated in the graphic, FID 3 of the restore chassis will be created as FID 13, fabric name backup_odd_fab, Domain ID 41, switch name switch_41.

“vfp” adds the ports to the logical switches. “s” copies all the switch settings except MAPS. “p” moves and configures all the ports. “maps” recreates the custom MAPS policies, rules, and groups. “e” enables all the logical switches and ports when done if the matching switch and ports were enabled in the restore chassis.

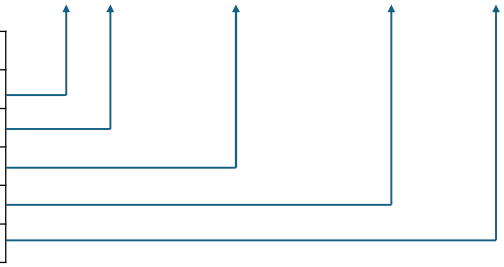
Typically, chassis settings aren’t changing when repurposing a switch, so I left that out. Zoning is fabric specific so I left “z” and “ze” out. I left custom user creation out, “u”, but if the switch came from an entirely different division or it’s a new switch, it might make sense to include “u”.

restore.py: Template for Virtual Fabrics

Repurpose a chassis by removing all port configurations and all logical switches. Recreate FID 1 in the target switch as FID 11, fabric name prod_odd_fab, switch DID 31, and name the switch switch_31. Recreate FID 3 in the target switch as FID 13, fabric name backup_odd_fab, switch DID 41, and name the switch switch_41.

```
py restore.py –common –p vfc,vfs,vfp,s,p,maps,e –fm
 1,11,app_1_odd_fab,31,switch_31;3,13,backup_odd_fab,41,switch_41
```

Index	Description
0	Restore FID: 3
1	Target FID: 13
2	Target fabric name: backup_odd_fab
3	Target DID: 41
4	Target switch name: switch_41



I'm just going to go over the FID map for the second switch. Note the semi-colon between the two.

This example is a typical use case for the FID map when using a chassis as a template to create logical switches in another chassis. In this example, assume two logical switches, FID 1 and FID 3 exist in the restore chassis:

FID 1 Could be for local production traffic while FID 3 could be for extended disaster recovery. They could have different MAPS policies.

Additional Notes

In this example, the target chassis is being repurposed. The vfc parameter for –p means all ports will be set to the factory default and moved to the default switch. Then all logical switches, except the default logical switch, will be deleted. Vfs then tells the script to create FID 1 of the restore chassis as FID 11 in the target switch with the following attributes:

Fabric name: app_1_odd_fab

Domain ID 31
Switch name: switch_31

Similarly, as illustrated in the graphic, FID 3 of the restore chassis will be created as FID 13, fabric name backup_odd_fab, Domain ID 41, switch name switch_41.

“vfp” adds the ports to the logical switches. “s” copies all the switch settings except MAPS. “p” moves and configures all the ports. “maps” recreates the custom MAPS policies, rules, and groups. “e” enables all the logical switches and ports when done if the matching switch and ports were enabled in the restore chassis.

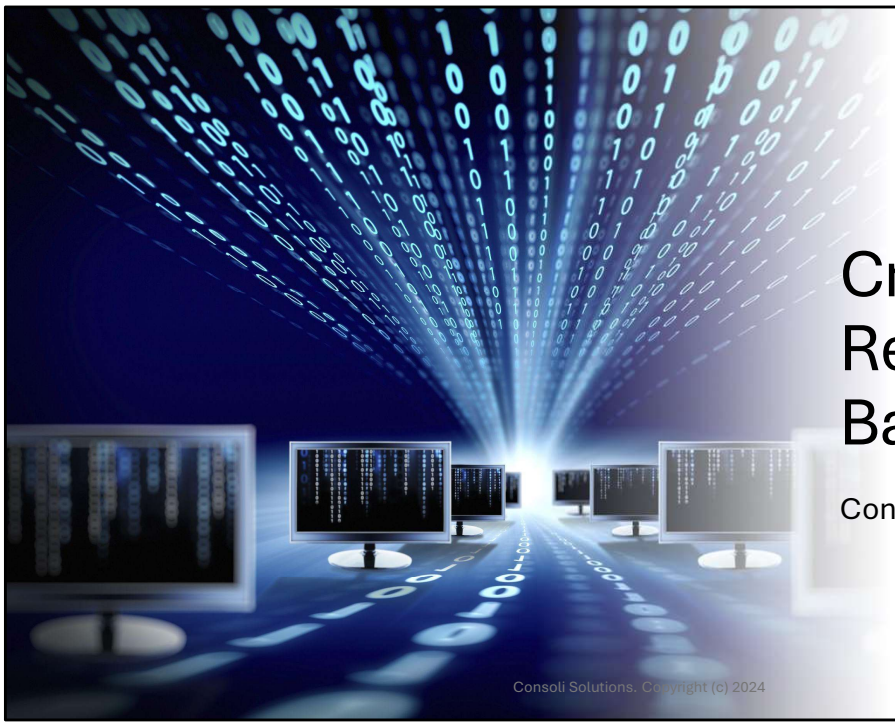
Typically, chassis settings aren’t changing when repurposing a switch, so I left that out. Zoning is fabric specific so I left “z” and “ze” out. I left custom user creation out, “u”, but if the switch came from an entirely different division or it’s a new switch, it might make sense to include “u”.

restore.py: Replace a Chassis Example

Full chassis restore:

```
py restore.py –common –p *
```


This pretty straight forward. All logical switches are deleted, all ports set to the factory default, and a full restore performed.



Create Restore Batch

Consoli Solutions

restore_all.py

Creates a batch file for all chassis in the project file. The –ip and –wwn parameters are automatically extracted from the input file, -i.

```
py restore_all.py –o restore_batch –id admin –pw password
  -i _capture_2024_03_08_16_11_04/combined
  –p vfc,vfs,vfp,s,p,maps,e
```

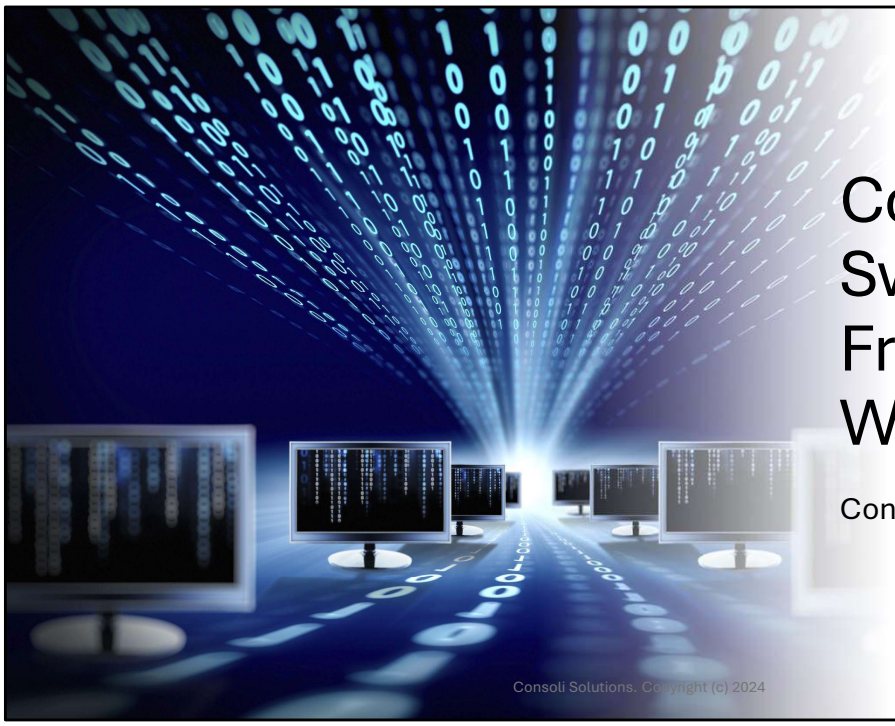
Sample Output:

```
start py restore_all.py -ip 10.144.72.15 –id admin –pw password
  -i _capture_2024_03_08_16_11_04/combined
  –wwn 10:00:c4:f5:7c:16:8b:7b –p vfc,vfs,vfp,s,p,maps,e
```

In Windows, “start” means run whatever follows in the background. The Unix equivalent is “&”

Building command lines with all of these options is tedious so like me, you’ll probably create batch files or scratch files with all the parameters. Before I finished testing, I got tired of doing that manually, so I created a script to build a batch file that recreates everything in the restore chassis to a target chassis with the same input parameters and then edited it as needed.

This is an easy way to generate the command line, even if you are just going to copy and paste a single line out of this file.



Configure Switches From a Workbook

Consoli Solutions

switch_config.py

- Uses an Excel workbook as a template
  - Fixed_Port_Switch_Configuration_yymmdd
  - X6-4_Switch_Configuration_yymmdd
  - X6-8_Switch_Configuration_ yymmdd
  - X7-4_Switch_Configuration_ yymmdd
  - X7-8_Switch_Configuration_ yymmdd
- Existing switches are only modified if necessary
  - Typically used to add ports to an existing switch
  - Switches and existing ports are not disabled

These workbook templates are in brocade-rest-api-applications. Copy the one that matches the switch type you want to configure. Editing these workbooks is covered on subsequent slides.

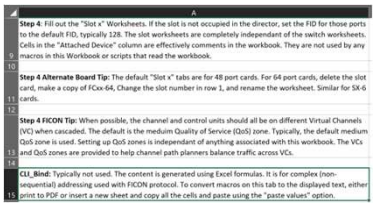
Additional Notes

Switches are not recreated if they already exist and ports are not set to the default if they already exist in the logical switch. Existing switches and ports will be modified to match the workbook if there is a mismatch; however, they will not be disabled. This means changes that require the switch or port to be disabled will fail. You will need to manually disable them before using this script to make changes that require a switch or port to be offline. This only effects existing switches and ports. Newly created logical switches and their ports are disabled before making any configuration changes. Enabling them is the last action taken.

- Uses an Excel workbook as a template which are available on github. There is a different workbook for each chassis type.
  - yymmdd is a timestamp representing the latest revision
  - See “Instructions” page for details

        - Use “Fixed _Port_Switch_Configuration” for all fixed port switches.
            - You’ll just get an error if you try to configure non-existent ports
            - The port index is the same as the port number so mainframe customers can use the same workbook
            - Works on FCIP switches as well, however, it will not configure the IP ports, routes, and tunnels.
        - There is a different slot numbering scheme for 4 slot and 8 slot bladed chassis which is why they need to be different
        - The port index (relevant for mainframe, FICON, zoning) is different between X6 and . Although I can’t think of a reason why you couldn’t use an X6 workbook to configure an X7 chassis and vice versa, I never tested it. The mainframe zoning, of course, would be off.
- Existing switches are only modified if necessary
    - Typically used to add ports to an existing switch
    - Some switch configuration parameters, such as the domain ID (DID) cannot be changed while the switch is online.
    - The switch_config.py script will not disable a switch.
- Basic switch configuration only. It does not modify or configure:
    - FCIP
    - Fibre channel routing
    - Encryption/Compression
    - F-Port trunking
    - Long distance settings

switch_config.py: Worksheets



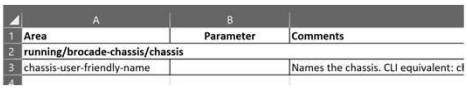
Step 4: Fill out the "Slot x" Worksheets. If the slot is not occupied in the director, set the FID for those ports to the default FID, typically 128. The slot worksheets are completely independant of the switch worksheets. Cells in the "Attached Device" column are effectively comments in the workbook. They are not used by any macros in this Workbook or scripts that read the workbook.

Step 4 Alternate Board Tip: The default "Slot x" tabs are for 48 port cards. For 64 port cards, delete the slot card, make a copy of FCxx-64, Change the slot number in row 1, and rename the worksheet. Similar for SX-6 cards.

Step 4 FICON Tip: When possible, the channel and control units should all be on different Virtual Channels (VC) when cascaded. The default is the meduim Quality of Service (QoS) zone. Typically, the default medium QoS zone is used. Setting up QoS zones is independant of anything associated with this workbook. The VCs and QoS zones are provided to help channel path planners balance traffic across VCs.

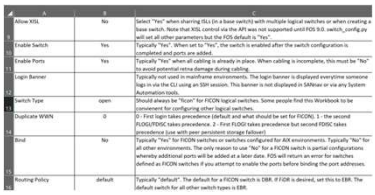
CLI_Bind: Typically not used. The content is generated using Excel formulas. It is for complex (non-sequential) addressing used with FICON protocol. To convert macros on this tab to the displayed text, either print to PDF or insert a new sheet and copy all the cells and paste using the "paste values" option.

Instructions



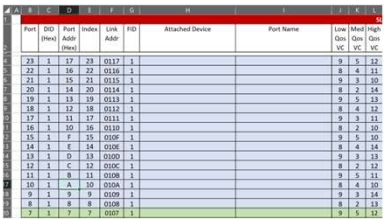
Area	Parameter	Comments
running/brocade-chassis/chassis		
chassis-user-friendly-name		Names the chassis. CLI equivalent: cl

Chassis Configuration



Allow XISL	No	Select "Yes" when sharing ISLs (in a base switch) with multiple logical switches or when creating a base switch. Note that XISL control via the API was not supported until FOS 9.0. switch_config.py will set all other parameters but the FOS default is "Yes".
Enable Switch	Yes	Typically "Yes". When set to "Yes", the switch is enabled after the switch configuration is completed and ports are added.
Enable Ports	Yes	Typically "Yes" when all cabling is already in place. When cabling is incomplete, this must be "No" to avoid potential retna damage during cabling.
Login Banner		Typically not used in mainframe environments. The login banner is displayed everytime someone logs in via the CLI using an SSH session. This banner is not displayed in SANnav or via any System Automation tools.
Switch Type	open	Should always be "ficon" for FICON logical switches. Some people find this Workbook to be convenient for configuring other logical switches.
Duplicate WWN	0	0 - First login takes precedence (default and what should be set for FICON). 1 - the second FLOGI/FDISC takes precedence. 2 - First FLOGI takes precedence but second FDISC takes precedence (use with peer persistent storage failover).
Bind	No	Typically "Yes" for FICON switches or switches configured for AIX environments. Typically "No" for all other environments. The only reason to use "No" for a FICON switch is partial configurations whereby additional ports will be added at a later date. FOS will return an error for switches defined as FICON switches if you attempt to enable the ports before binding the port addresses.
Routing Policy	default	Typically "default". The default for a FICON switch is DBR. If FIDR is desired, set this to EBR. The default switch for all other switch types is EBR.

Switch Configuration



Port	DID (Hex)	Port Addr (Hex)	Index	Link Addr	FID	Attached Device	Port Name	Low Qos VC	Med Qos VC	High Qos VC
23	1	17	23	0117	1			9	5	12
22	1	16	22	0116	1			8	4	11
21	1	15	21	0115	1			9	3	10
20	1	14	20	0114	1			8	2	14
19	1	13	19	0113	1			9	5	13
18	1	12	18	0112	1			8	4	12
17	1	11	17	0111	1			9	3	11
16	1	10	16	0110	1			8	2	10
15	1	F	15	010F	1			9	5	10
14	1	E	14	010E	1			8	4	14
13	1	D	13	010D	1			9	3	13
12	1	C	12	010C	1			8	2	12
11	1	B	11	010B	1			9	5	11
10	1	A	10	010A	1			8	4	10
9	1	9	9	0109	1			9	3	14
8	1	8	8	0108	1			8	2	13
7	1	7	7	0107	1			9	5	12

Port Configuration

switch_config.py: Chassis Worksheets

Area	Parameter	Comments
running/brocade-chassis/chassis		
chassis-user-friendly-name	My Chassis Name	Names the chassis. CLI equivalent: chassisname

- To name the chassis, simply replace “My Chassis Name” with your chassis name.
- If you are adventurous and want to make any other chassis changes:
  - If the cell in the “Area” column contains “running” or “operations” it is assumed to be a new branch. All else is assumed to be a leaf of that branch.
  - The value goes in the cell in the “Parameter” column.

Chassis changes are rare so the initial launch of this script, and therefore the workbook, did not have any chassis definitions in it. Well, chassis changes are almost rare. Changing the chassis name is fairly common. I didn’t think of it until later so I just shoe horned this in. The other sheets are more use friendly. Reading a URI and value from the workbook was quick and easy. Other than the brief comment on the slide, discussing other changes and URIs is outside the scope of this video. If you want to rename a chassis, just replace “My Chassis Name” with whatever name you want to give the chassis. If you don’t want to rename the chassis, just clear that cell. If you leave everything else alone, you don’t need to know anything about URIs.

A neater user interface is in consideration for the future. The advantage of this approach is that you can do whatever you The disadvantage is that you need to know the branches and leaves of the URIs used by the API to make other chassis changes.

Additional Notes

If you intend to use this script to modify other chassis parameters, here’s what you need to know:

- If the cell in the Parameter column is empty, no changes are made.
- The “comments” column is not used by the script. In fact, only the “Area” and “Parameter” columns are used. You can add other columns wherever you wand if desired. The columns “Area” and “Parameter”, but otherwise you can give your columns any name you want.
- There isn’t any checking by the script to validate any of the branches, leaves, or parameters. A runtime error will be returned by FOS if anything is wrong.

switch_config.py: Switch Worksheets

Area	Parameter	Comments
Fabric ID (FID)	128	FIDs are necessary for FOS to diffe determine what logical switches c chassis must have a unique FID an same FID. A FID can be any integer to use FID 128 for the default logic number to always represent a FICC cascaded, use different FID numbe customers, the FID is not used in H
Fabric Name		The recommended best practice is used in MVS or in HCD but this ma recommended. Most mainframe c same name as the switch + "_fab".
Switch Name		The switch name is not used in HCI DID in the switch name so that it is
Domain ID (DID)	0x01	In FICON, the domain ID is the high

Typically, you modify the “Switch_x” worksheet. Make copies when configuring multiple logical switches. As long as the sheet name begins with “Switch_” it doesn’t matter what the sheet name is. I always replace “x” with the FID number. The DID and FID fields for the “Slot_x” worksheets are used to determine what ports belong with which logical switch. A DID and FID in the “Slot_x” worksheets that don’t have a matching “Switch_x” worksheet are ignored. Typically, I leave the default switch at FID 128 without having a “Switch_x” worksheet with FID 128. That way, all ports with FID 128 are ignored.

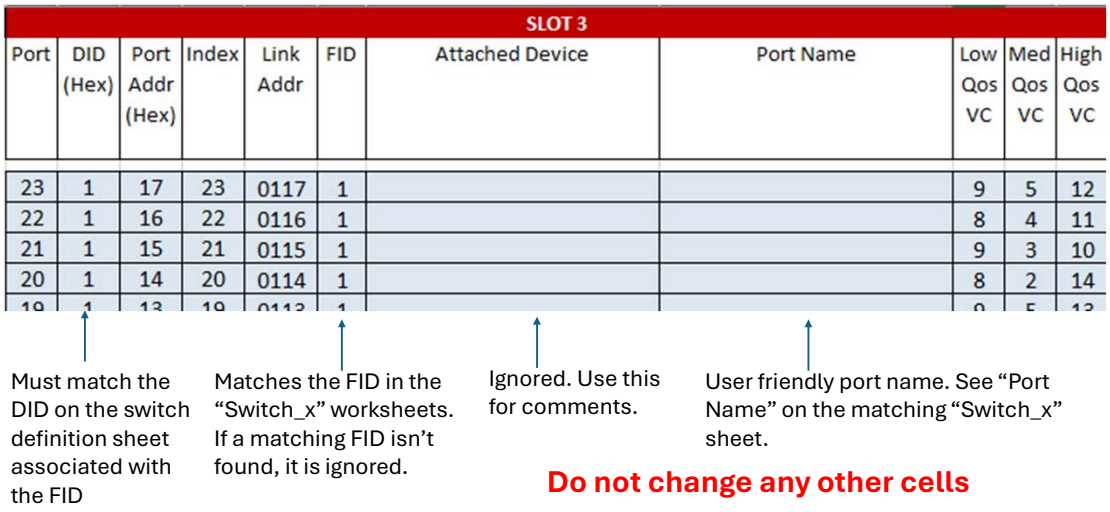
There isn’t much to add to what isn’t already in the “Comments”. Just fill out the forms. Pull down menus are used anywhere I could to help ease parameter choosing.

Additional Notes

One parameter I do want to comment on is the “Duplicate WWN” field. Not because it’s complex, but because I frequently see it set wrong. If you are using storage that fails over to another port in the event of a failure on the storage port, it will present that WWN on another port. From the fabric perspective, when this happens, it is usually seen as a duplicate WWN. By default, FOS rejects any duplicate WWN. That

means if there is a fault in the storage array, the login for the faulted port will remain active instead of the new login. Typically, vendors use the word “persistent” in the name for this feature. If you have this feature enable in your storage array, you want FOS to replace the existing WWN with the most current WWN. When set incorrectly, this can be a ticking time bomb that can express itself years after the switch configuration is complete and everyone is looking at the storage array when the problem is in the switch.

switch_config.py: Port Worksheets



SLOT 3										
Port	DID (Hex)	Port Addr (Hex)	Index	Link Addr	FID	Attached Device	Port Name	Low Qos VC	Med Qos VC	High Qos VC
23	1	17	23	0117	1			9	5	12
22	1	16	22	0116	1			8	4	11
21	1	15	21	0115	1			9	3	10
20	1	14	20	0114	1			8	2	14

The DID and FID cells autofill based on the previous value (which is from the bottom row up). Keep that in mind, especially if there are gaps.

"Attached Device" is a comment only. It was initially intended for mainframe planning purposes, hence the name "Attached Device". It is not used by the script. The "link Addr" is also a mainframe reference.

The slot number is 0 for fixed port switches.

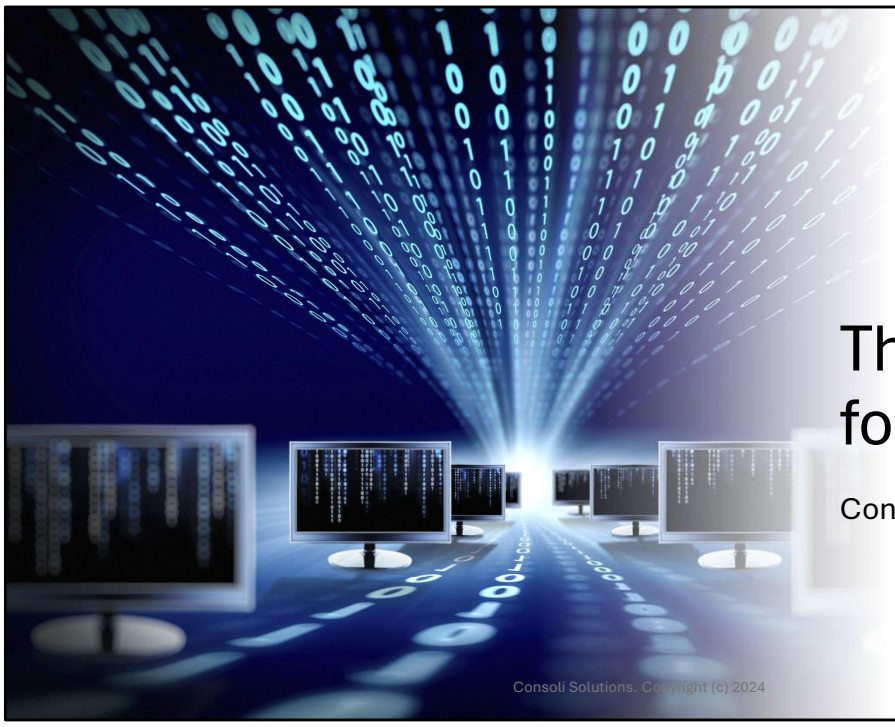
Additional Notes

The "Port Addr" is the middle byte of the fibre channel address. I can't think of a reason why anyone would care about this field. It's here to simplify the Excel formula to build the Link Addr.

The "Link Addr" is the first two bytes of the fibre channel address. This is a useful reference for mainframe users. It does not include the least significant byte, the ALPA, which initially was for loop addresses. It is automatically calculated from the DID and Port Addr. I can't think of a reason why anyone not in a mainframe environment

would care about this field.

Low, Med, and High QoS VC are the virtual channel numbers used. FOS automatically calculates these VCs. They are here for reference only, primarily for mainframe (FICON) users. They recommended best practice is to balance traffic across all VCs when ISLs (fabric with more than one switch) is used. Typically, open systems environments need to balance traffic based on usage. SANnav is typically used to analyze traffic, but the stats_c.py and stats_g.py to that using the API. Since traffic in a mainframe environment is usually balanced, a balanced ratio of channel ports to control units is adequate.



Thank you for watching

Consoli Solutions