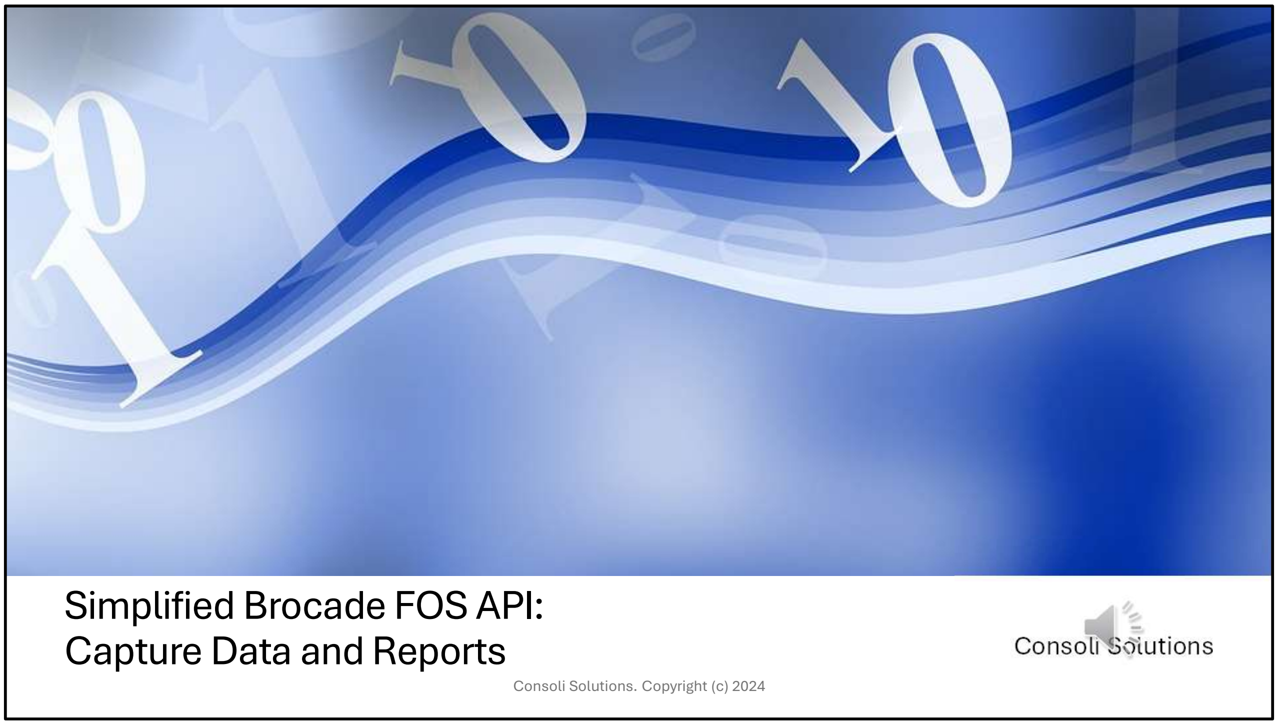


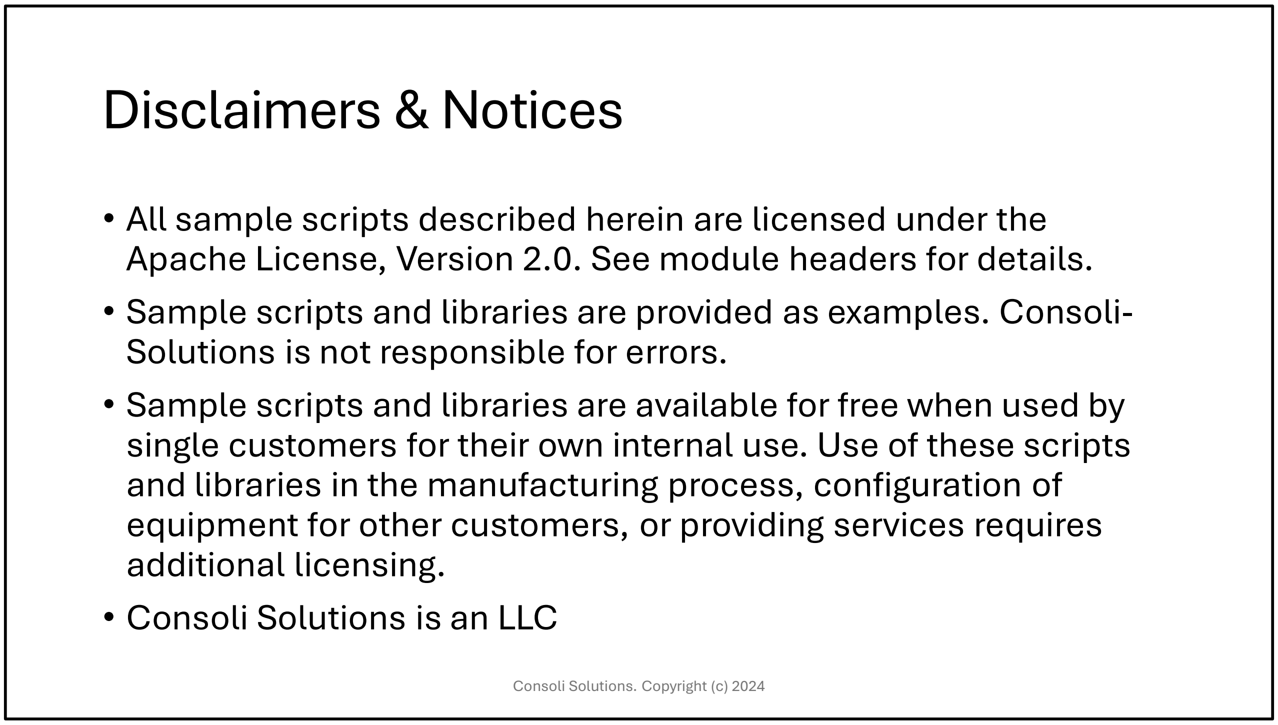
Simplified Brocade FOS API: Capture Data and Reports

Consoli Solutions

Description:

This video discusses how to use the sample scripts to capture data via the FOS RESTConf API, generate reports, and perform a nodefind across multiple fabrics.

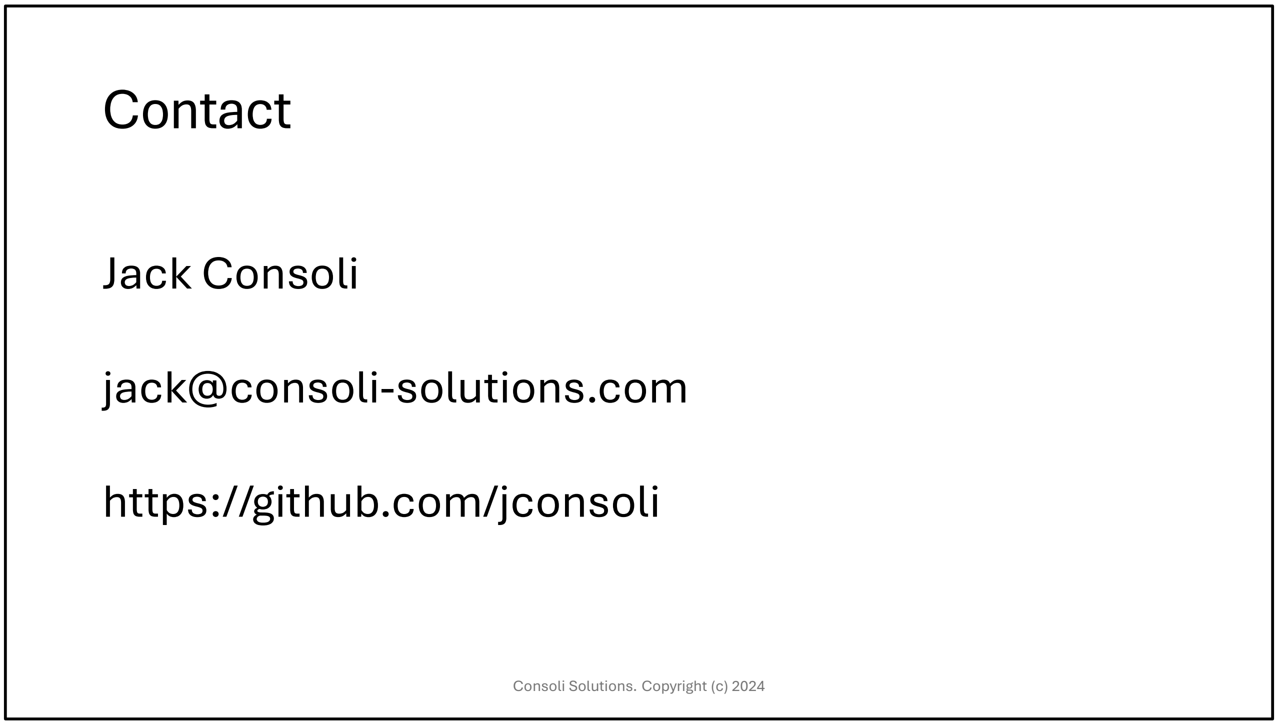
Keywords: Simplified FOS API capture data compare report nodefind



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.



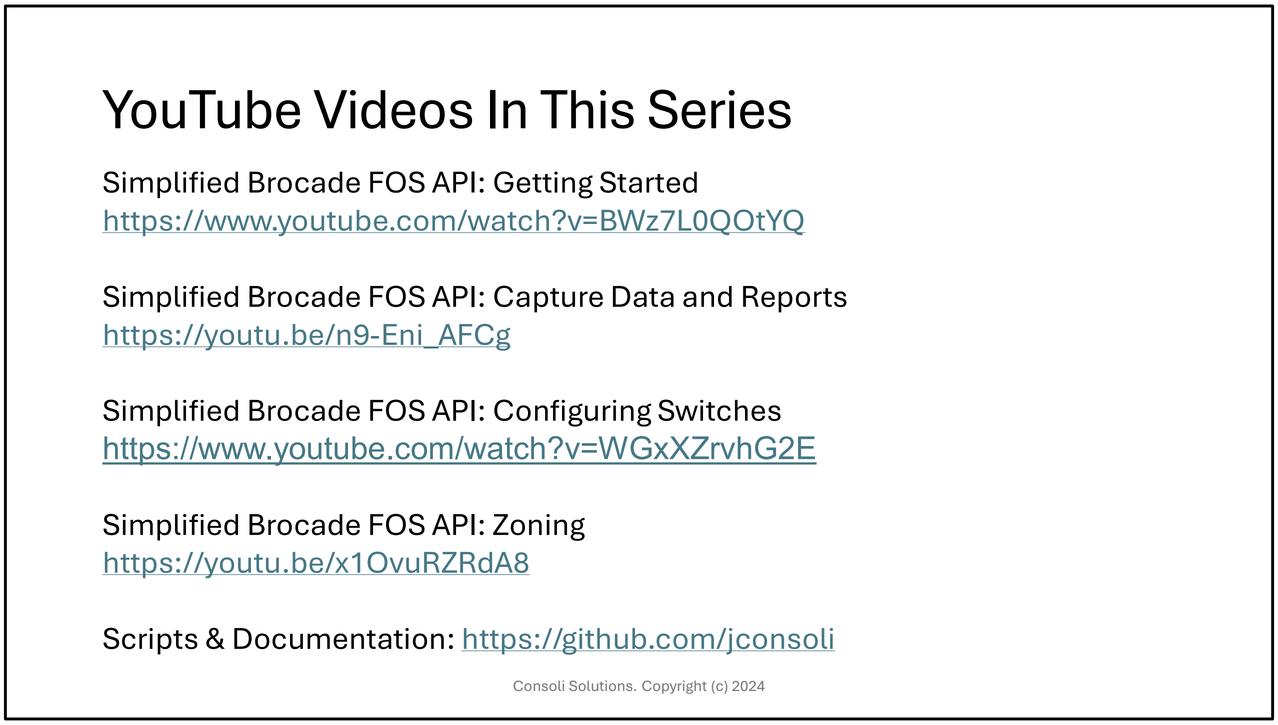
Contact

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

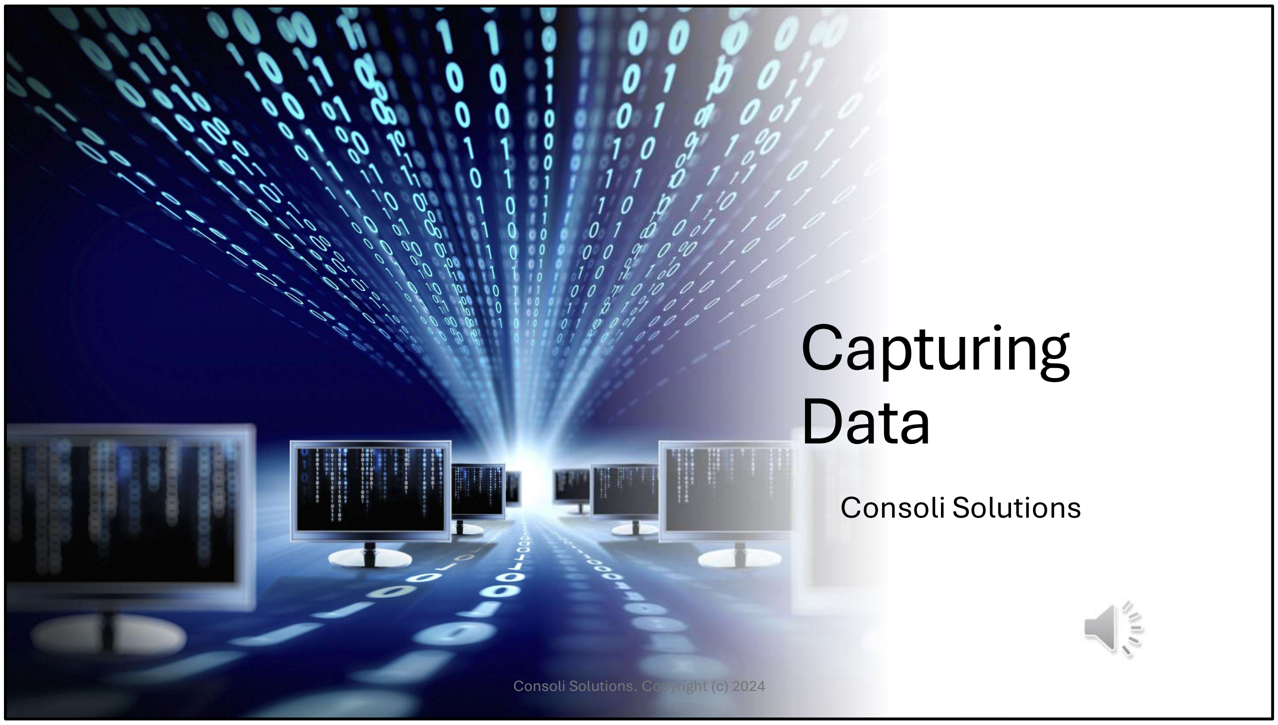
Simplified Brocade FOS API: Getting Started
Discusses how to download Python, what libraries are needed, how to install libraries, some tips for Python newbies, a few FOS settings you need to be aware of, an overview of the libraries I created to simplify scripting tasks, and an introduction to the sample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

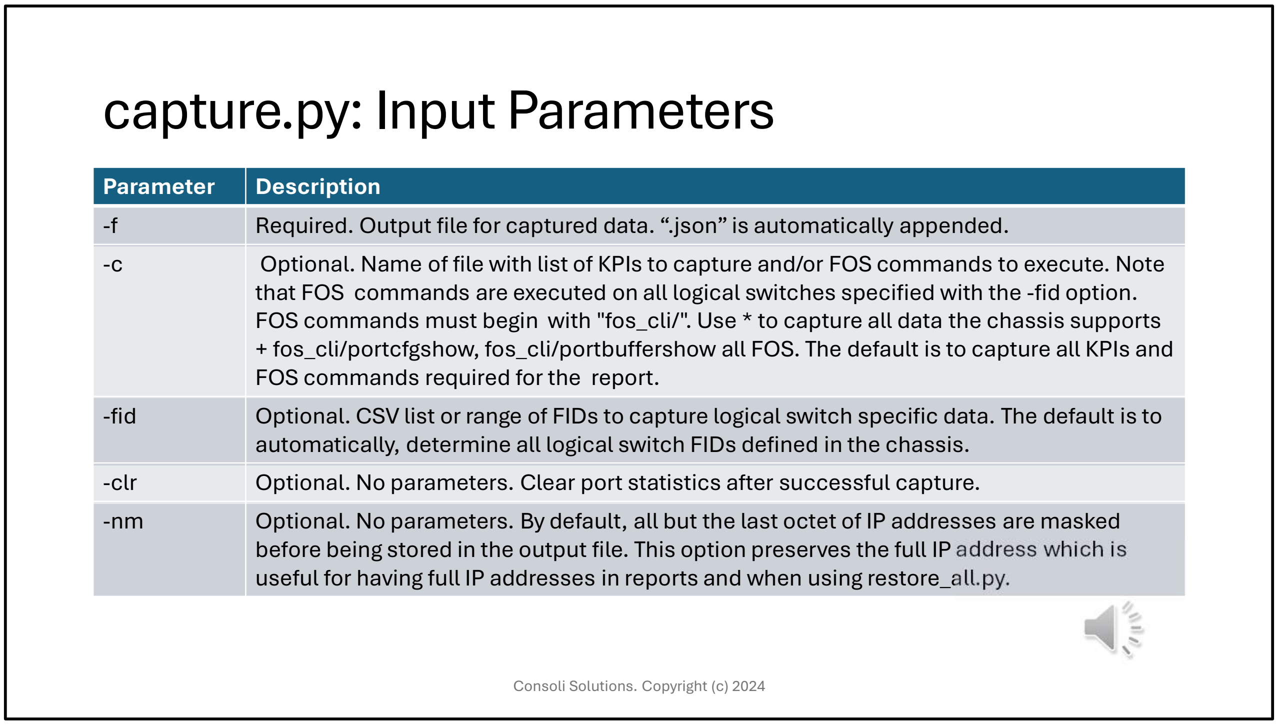
Simplified Brocade FOS API: Zoning
Zone from a workbook, restore zoning from a previous data capture, merge zoning databases from multiple fabrics, and zone from CLI



Capturing Data

Consoli Solutions

Let’s talk about capturing data first.



capture.py: Input Parameters

Parameter	Description
-f	Required. Output file for captured data. ".json" is automatically appended.
-c	Optional. Name of file with list of KPIs to capture and/or FOS commands to execute. Note that FOS commands are executed on all logical switches specified with the -fid option. FOS commands must begin with "fos_cli/". Use * to capture all data the chassis supports + fos_cli/portcfgshow, fos_cli/portbuffershow all FOS. The default is to capture all KPIs and FOS commands required for the report.
-fid	Optional. CSV list or range of FIDs to capture logical switch specific data. The default is to automatically, determine all logical switch FIDs defined in the chassis.
-clr	Optional. No parameters. Clear port statistics after successful capture.
-nm	Optional. No parameters. By default, all but the last octet of IP addresses are masked before being stored in the output file. This option preserves the full IP address which is useful for having full IP addresses in reports and when using restore_all.py.

Consoli Solutions. Copyright (c) 2024

Typically, data is captured with the multi_capture script which will be discussed later in this video. Since multi_capture launches this script, a brief explanation of those inputs is required.

In addition to the login credentials and logging options discussed in "Getting Started" video, this module has the following inputs:

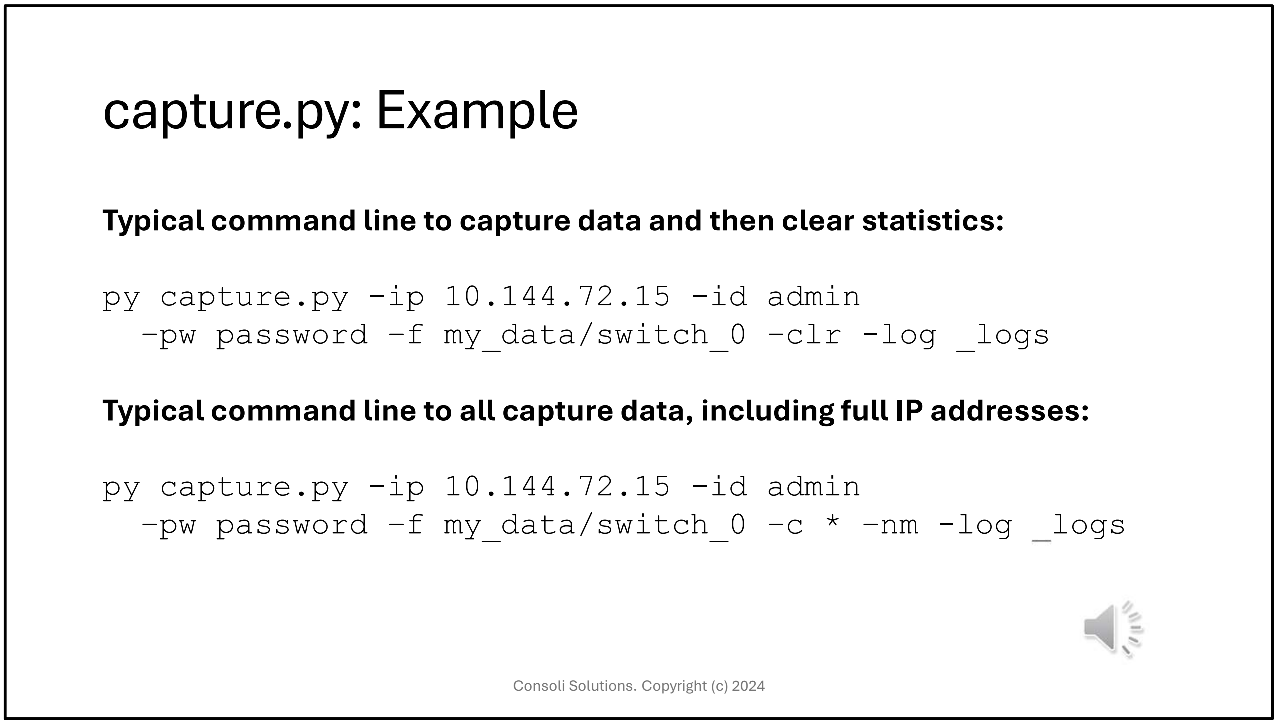
-f: Output file for captured data. When launched from multi_capture, this is the folder project folder it created.

-c: Optional. By default, only data used by the reports is captured. If you intend to use captured data for other scripts, such as restore, consider using an asterisk, *. When using an asterisk, the script will attempt to request data for everything FOS supports. Depending on the chassis type, some of those requests may not be supported so you will see error messages which can be ignored. Using this option with file names is outside the scope of this video.

-fid: Optional. By default, logical switches are automatically determined and data is captured for all FIDs. Using this option with a list of FIDs is outside the scope of this video.

-clr: Clears port statistics after port statistics have been captured.

-nm: Stores full IP addresses. By default, only the last octet of an IP address is stored.



capture.py: Example

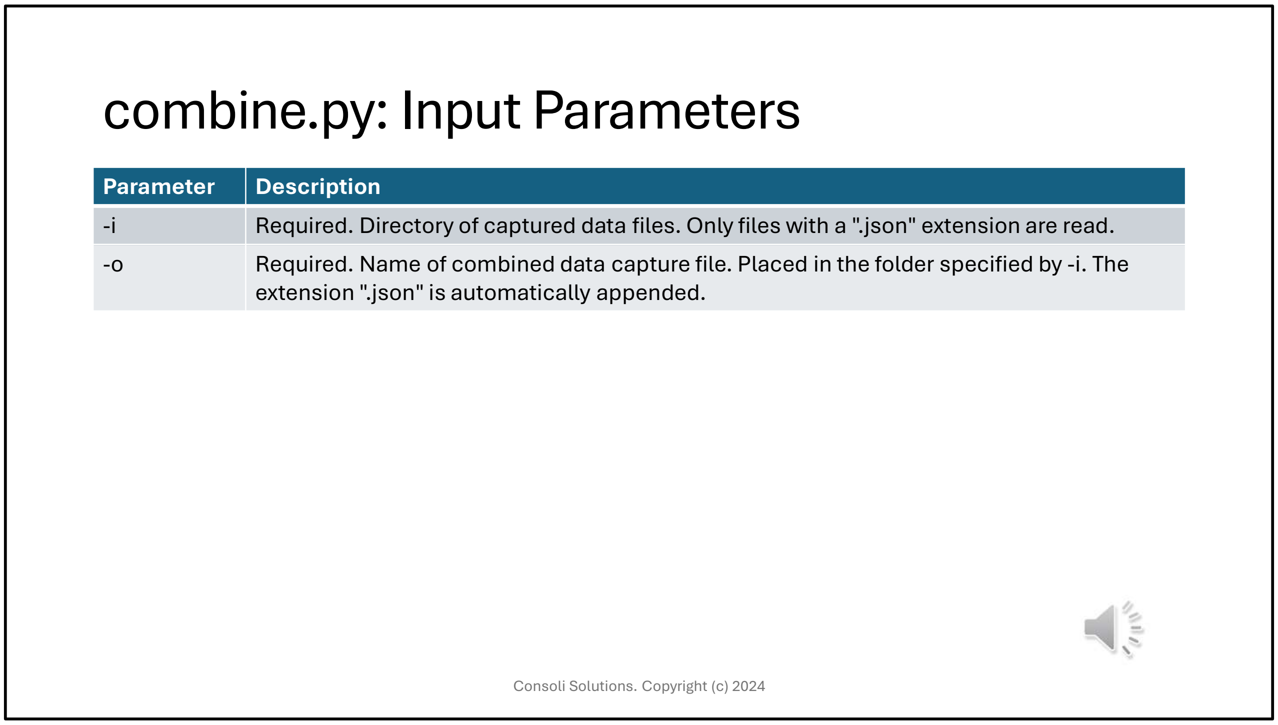
Typical command line to capture data and then clear statistics:

```
py capture.py -ip 10.144.72.15 -id admin
  -pw password -f my_data/switch_0 -clr -log _logs
```

Typical command line to all capture data, including full IP addresses:

```
py capture.py -ip 10.144.72.15 -id admin
  -pw password -f my_data/switch_0 -c * -nm -log _logs
```

The intended use of these examples is to provide you with something that you can copy, paste, and modify for your own use. Note that I’m using a backslash in the file name. A forward slash is interpreted as a parameter separator. If you want to use a forward slash in the file name, it must be enclosed in quotes.



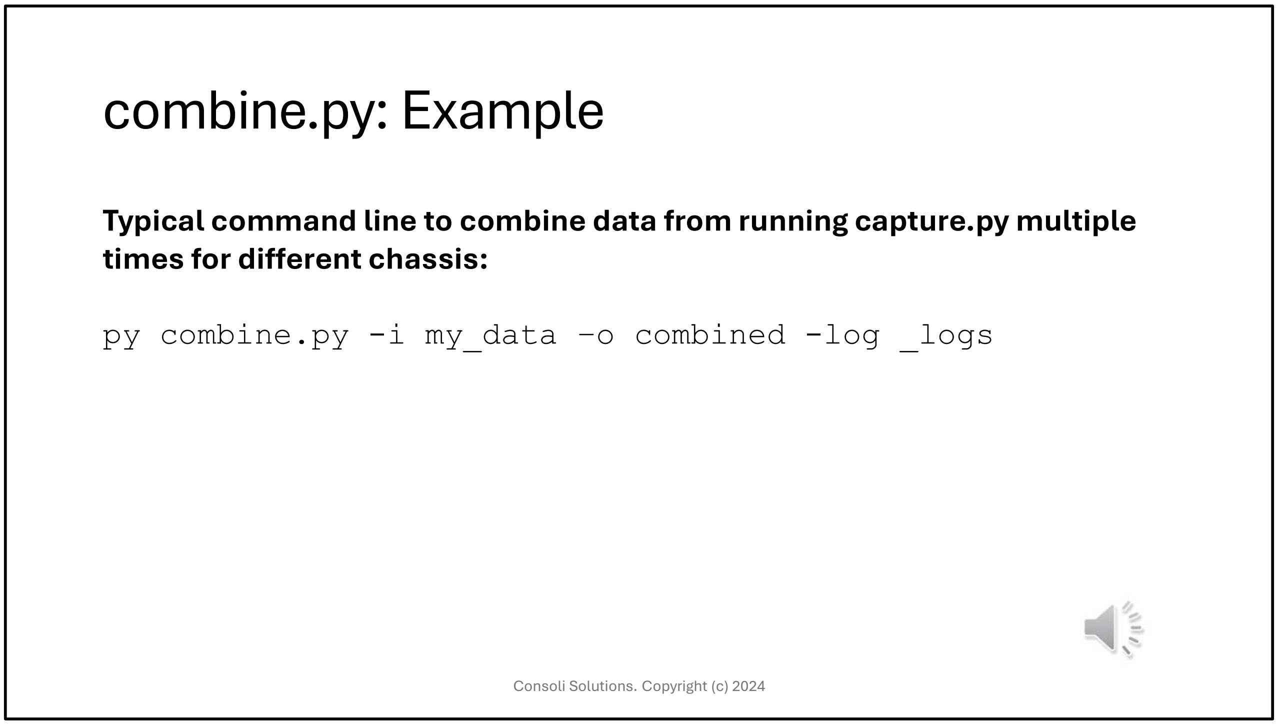
combine.py: Input Parameters

Parameter	Description
-i	Required. Directory of captured data files. Only files with a ".json" extension are read.
-o	Required. Name of combined data capture file. Placed in the folder specified by -i. The extension ".json" is automatically appended.

As with the capture script, this script is typically launched from other scripts. multi_capture is one of those scripts. The most common reason for using it as a stand alone script is when you want to include capture data from another SAN that was not included in the original set of captured data. If you re-run this script, delete the combined data file first.

-i: This is a folder, not an individual file. When launched from multi_capture, this is the project folder it created.

-o: This is the name of the output file only. It is placed in the folder specified with the –i parameter. When launched from multi_capture.py, this file is named “combined”.

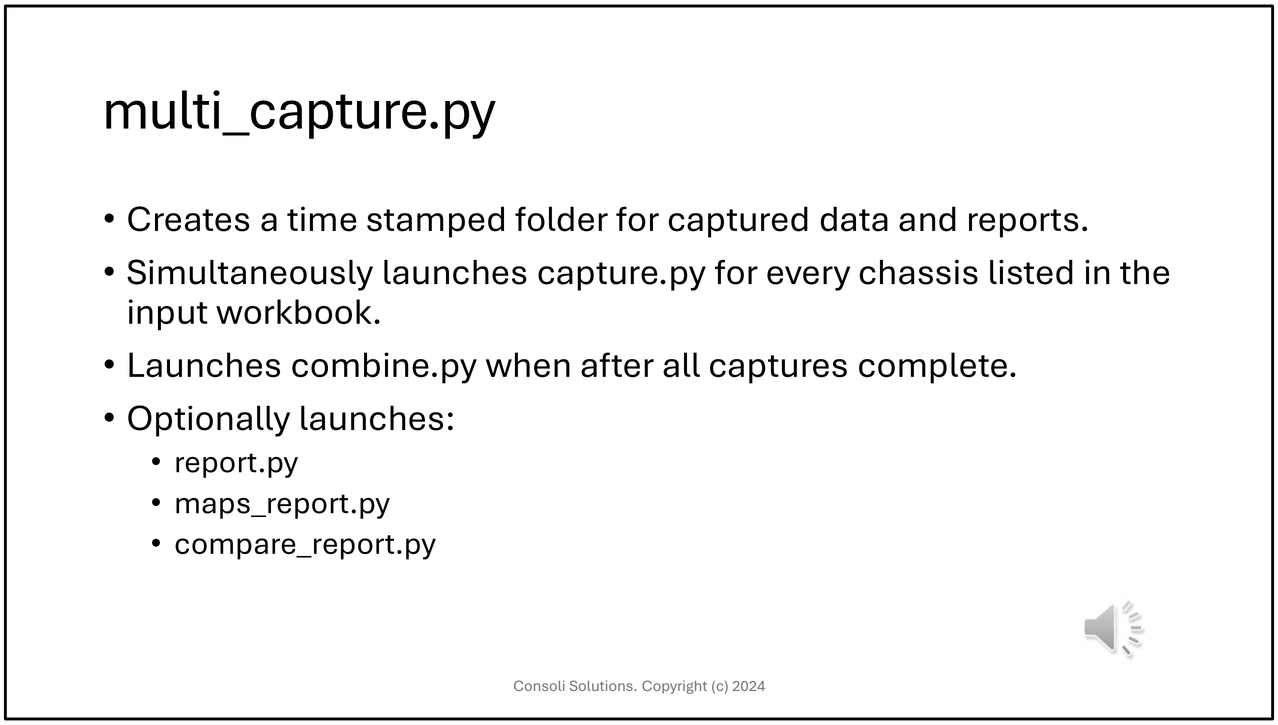


combine.py: Example

Typical command line to combine data from running capture.py multiple times for different chassis:

```
py combine.py -i my_data -o combined -log _logs
```

The intended use of this examples is to provide you with something that you can copy, paste, and modify for your own use



multi_capture.py

- Creates a time stamped folder for captured data and reports.
- Simultaneously launches capture.py for every chassis listed in the input workbook.
- Launches combine.py when after all captures complete.
- Optionally launches:
  - report.py
  - maps_report.py
  - compare_report.py

Other than performing some management functions to create a project folder and launch other scripts, it doesn't do anything else. Typically, data is captured for a full SAN or multiple SANs. My advice is to capture data from all switches in both the A & B fabrics and, at least occasionally, all fabrics in your organization. It's not uncommon to have zone members in the wrong fabric. By capturing all data, the report will flag zone members that are found in different fabrics.

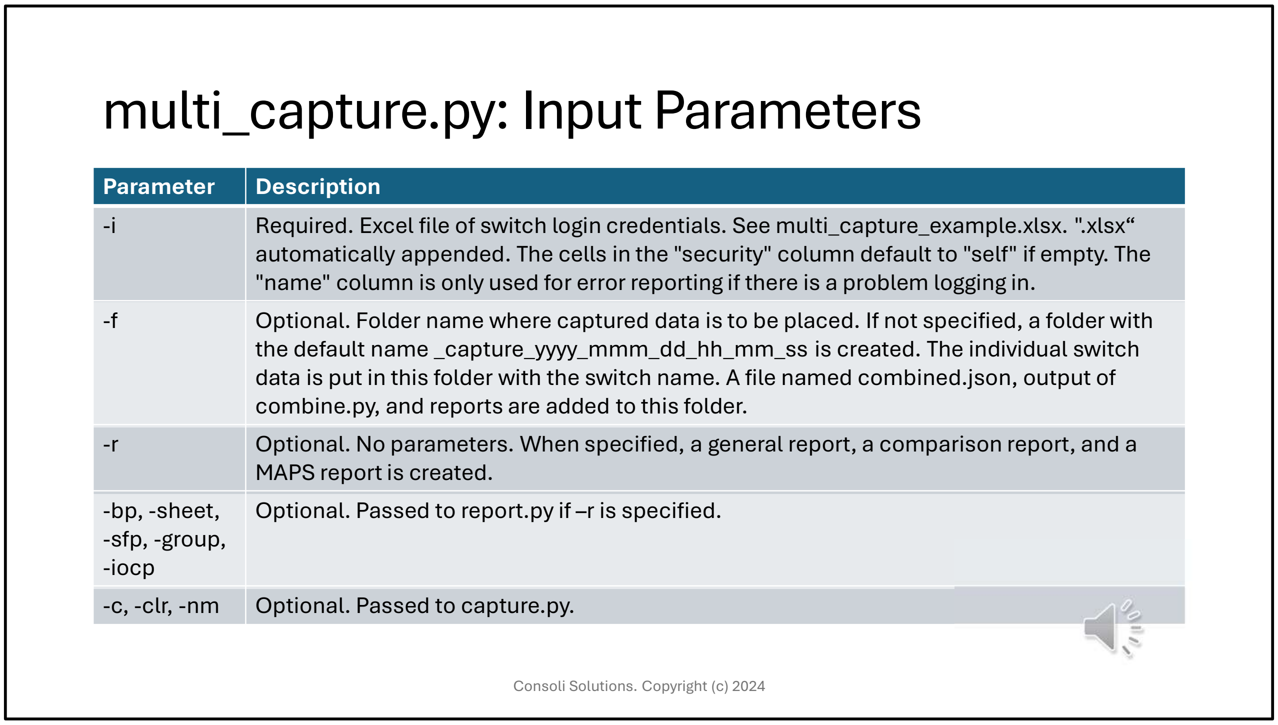
With a Windows PC and just 16G of memory, I was able to launch 200 instances of capture at the same time. That means I had 200 file handles open and 200 HTTP sessions open at once without issue. I didn't have 200 switches so the HTTP sessions were spoofed. My concern was the number open HTTP sessions. My conclusion of this test was that you will have to make a sales rep very happy to break it.

Additional Notes

To minimize data capture time, the capture script is launched for every chassis at the same time. There is sufficient processing time between launching capture so there is no need to worry about duplicate log file names. Log file names are time stamped down to the millisecond. Based on research and some experimentation, for 200 chassis there were no problems having that many file handles open or HTTP sessions open.

After all instances of capture complete, combine.py is then launched. The output file is simple named: “combined.json” and placed in the aforementioned folder.

If –r is specified, the report, compare_report, and maps_report scripts are launched. They are launched serially because very large SANs will have very large data captures. Having three report utilities running at the same time can consume excessive memory. The compare_report and maps_report scripts don’t take much time to run, so there is little advantage in running them in parallel.



multi_capture.py: Input Parameters

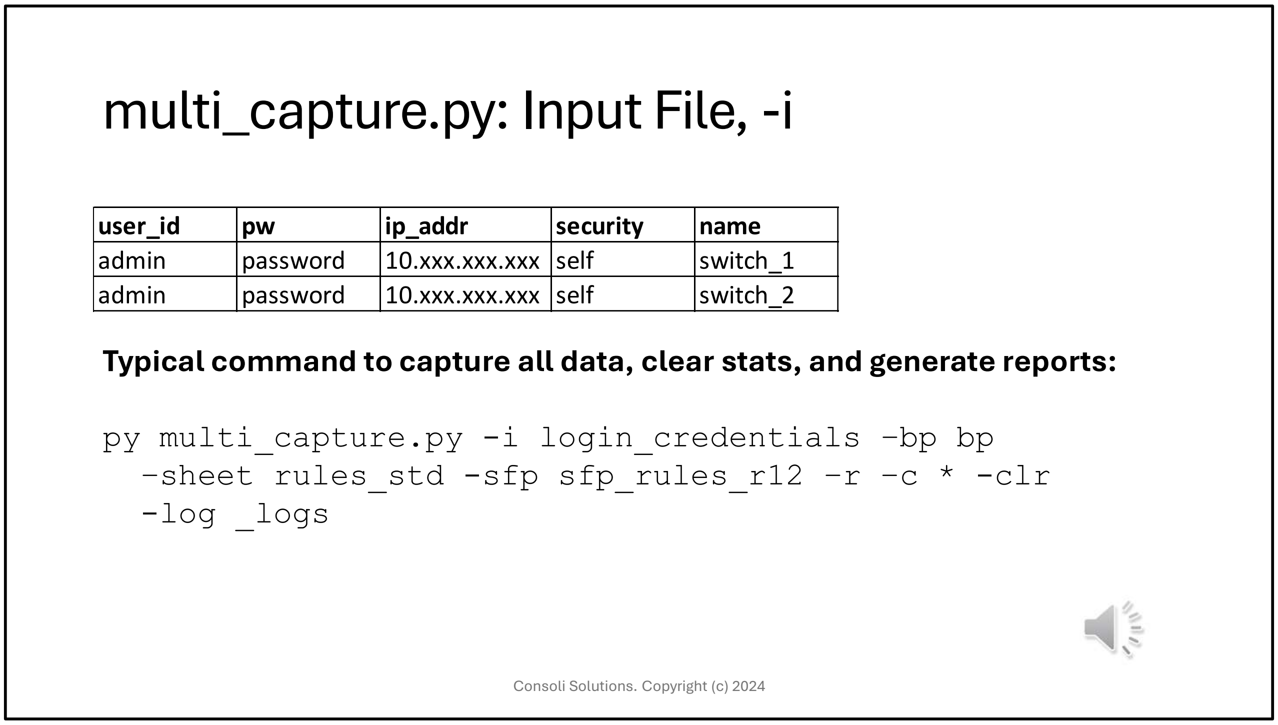
Parameter	Description
-i	Required. Excel file of switch login credentials. See multi_capture_example.xlsx. ".xlsx“ automatically appended. The cells in the "security" column default to "self" if empty. The "name" column is only used for error reporting if there is a problem logging in.
-f	Optional. Folder name where captured data is to be placed. If not specified, a folder with the default name _capture_yyyy_mmm_dd_hh_mm_ss is created. The individual switch data is put in this folder with the switch name. A file named combined.json, output of combine.py, and reports are added to this folder.
-r	Optional. No parameters. When specified, a general report, a comparison report, and a MAPS report is created.
-bp, -sheet, -sfp, -group, -iocp	Optional. Passed to report.py if –r is specified.
-c, -clr, -nm	Optional. Passed to capture.py.

-i: The name of the workbook with the login credentials. Since this module captures data from multiple chassis, it would be to cumbersome to add all of the login credentials via the command line, so login credentials are read from an Excel workbook. Use “multi_capture_example.xlsx” from brocade-rest-api-applications as a template. This workbook is discussed in greater detail on the next slide.

-f: Allows you to specify a project folder name. Use of this option is outside the scope of this video. By default, a time stamped folder name is created.

-r: Optional. No parameters. When specified, launches the report, compare_report, and maps_report scripts

“-c, -clr, and –nm” are passed to capture.py. “-bp, -sheet, -sfp,-group, and –iocp” are passed to the report script which will be covered shortly. .py. Capture was recently covered. report.py is covered in the next section.



multi_capture.py: Input File, -i

user_id	pw	ip_addr	security	name
admin	password	10.xxx.xxx.xxx	self	switch_1
admin	password	10.xxx.xxx.xxx	self	switch_2

Typical command to capture all data, clear stats, and generate reports:

```
py multi_capture.py -i login_credentials –bp bp
  –sheet rules_std -sfp sfp_rules_r12 –r –c * -clr
  -log _logs
```

Workbook:

Most of this is pretty self explanatory. The security column does not default to self, so it must be explicitly stated. The “name” column is required, but it is only used for error reporting. For example, if the login fails, this name appears in the error message. You should use a name that you can easily identify with the chassis, but any printable characters are supported.

There is no limit as to how many rows you can use.

Only these columns are read. As long as the header remains in row 1 and you don’t reuse any of these header names, you can add whatever other columns you want. It’s common to add columns for physical location and other documentation. Columns are found by name, so the order of columns doesn’t matter.

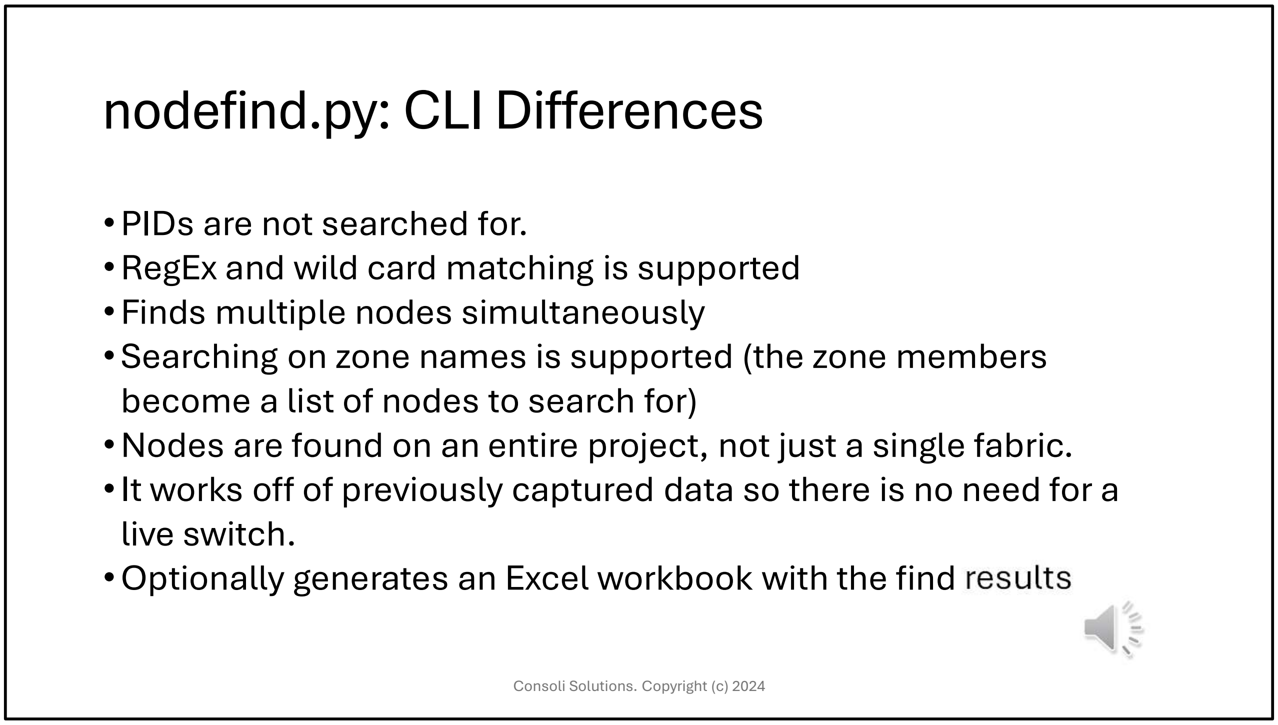
Typical command:

As with other examples, this is here for copy and paste. This is how I typically launch the script.

Reports

Consoli Solutions

Now that we’ve captured data, let’s generate some reports.



nodefind.py: CLI Differences

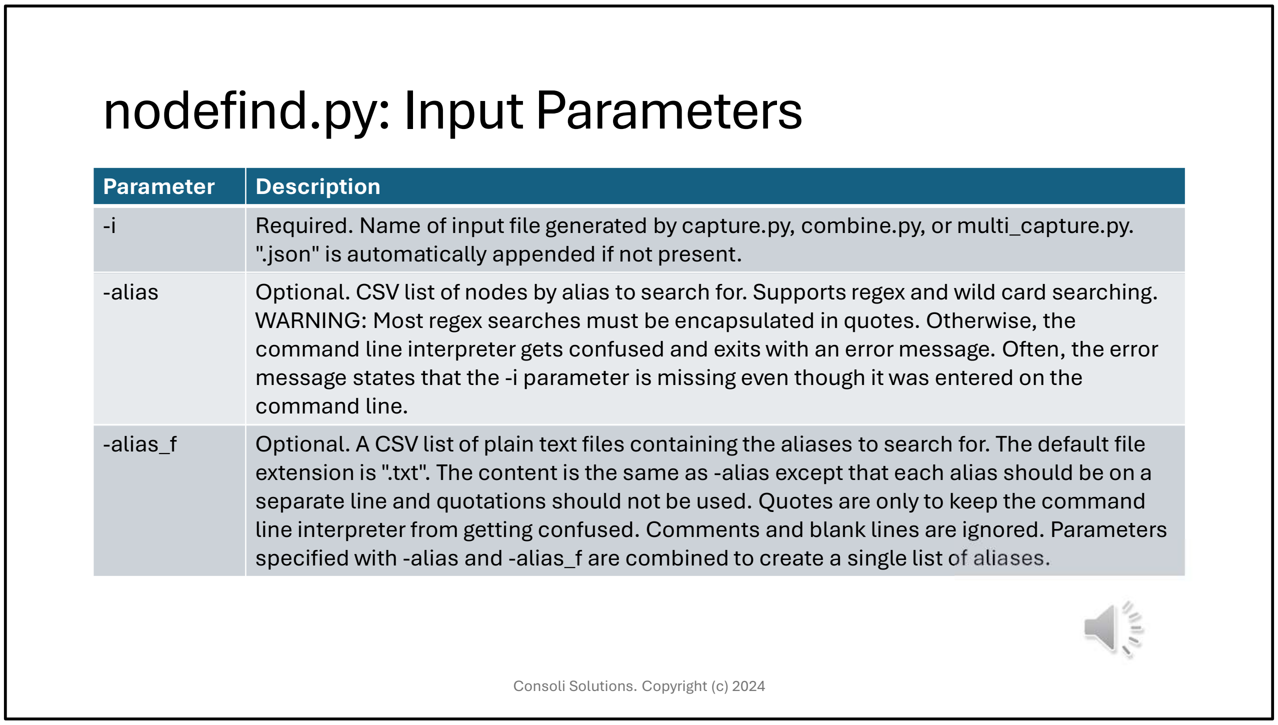
- PIDs are not searched for.
- RegEx and wild card matching is supported
- Finds multiple nodes simultaneously
- Searching on zone names is supported (the zone members become a list of nodes to search for)
- Nodes are found on an entire project, not just a single fabric.
- It works off of previously captured data so there is no need for a live switch.
- Optionally generates an Excel workbook with the find results

This is the most frequently used script I have. That's not surprising because nodefind is one of the most frequently used CLI command.

RegEx & wild card matching: This is the most valuable feature. I'm often looking for all nodes belonging to a server cluster or storage array.

Find on entire project: This is valuable when searching for a server cluster or storage array because I when I'm searching that way, I want to see both the A & B fabric.

workbook: This is convenient because often, when searching for server clusters or storage arrays, it's because I'm preparing a list of ports for cable teams to work on.



nodefind.py: Input Parameters

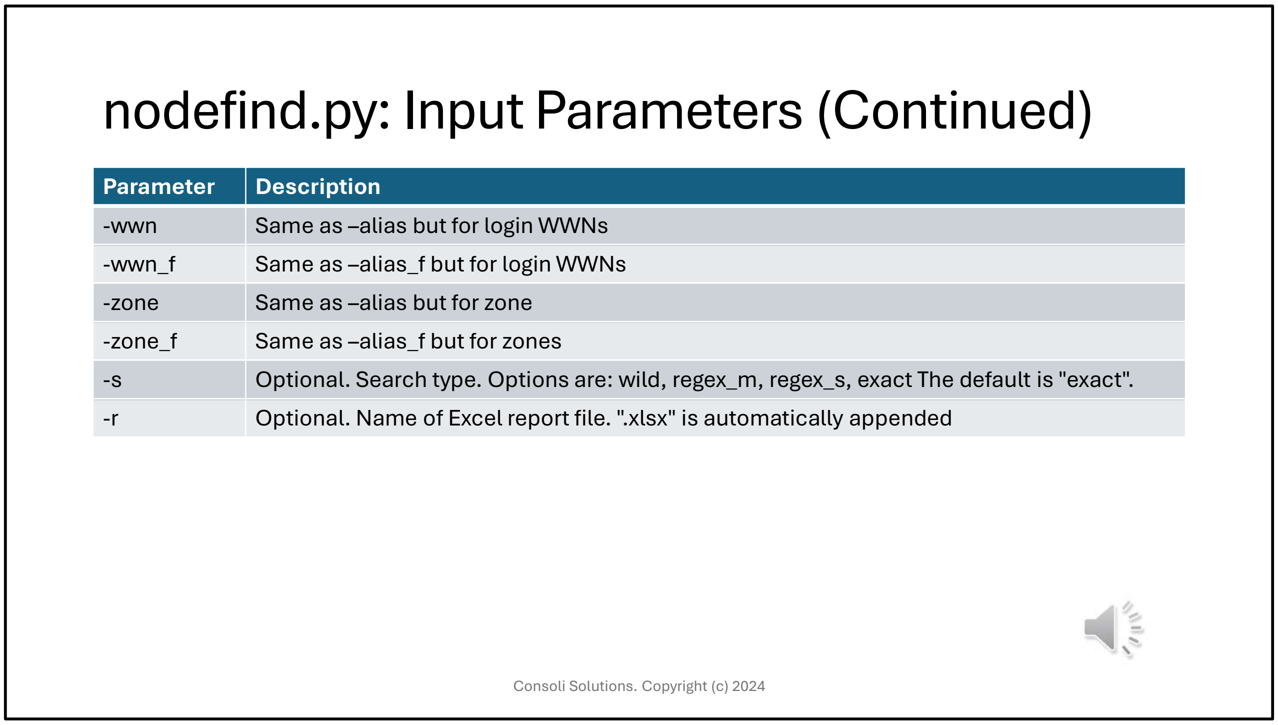
Parameter	Description
-i	Required. Name of input file generated by capture.py, combine.py, or multi_capture.py. ".json" is automatically appended if not present.
-alias	Optional. CSV list of nodes by alias to search for. Supports regex and wild card searching. WARNING: Most regex searches must be encapsulated in quotes. Otherwise, the command line interpreter gets confused and exits with an error message. Often, the error message states that the -i parameter is missing even though it was entered on the command line.
-alias_f	Optional. A CSV list of plain text files containing the aliases to search for. The default file extension is ".txt". The content is the same as -alias except that each alias should be on a separate line and quotations should not be used. Quotes are only to keep the command line interpreter from getting confused. Comments and blank lines are ignored. Parameters specified with -alias and -alias_f are combined to create a single list of aliases.

Note the warning with –alias regarding the use of quotes. This applies to –wwn and –zone as well. RegEx text often contains characters that are interpreted as parameter separators so they need to be enclosed in quotes. IMO, the resulting error message is a bug in the parser library. I'm not writing my own custom argument parser, so I'm living with this.

i: Typically the folder name created by multi_capture slash combined

-alias: An alias or CSV list of aliases to search for. The alias may contain ReGex or wild card characters. See the –s option discussed on the next slide.

-alias_f: Just another way to enter aliases. Sometimes, it's easier to copy and paste a list of aliases into a file.



nodefind.py: Input Parameters (Continued)

Parameter	Description
-wwn	Same as –alias but for login WWNs
-wwn_f	Same as –alias_f but for login WWNs
-zone	Same as –alias but for zone
-zone_f	Same as –alias_f but for zones
-s	Optional. Search type. Options are: wild, regex_m, regex_s, exact The default is "exact".
-r	Optional. Name of Excel report file. ".xlsx" is automatically appended

Consoli Solutions. Copyright (c) 2024

-wwn, -wwn_f: Same as alias and alias_f but for WWNs.

-zone, -zone_f: Same as alias and alias_f but for the zones. When using a RegEx or wild cards, the filter is applied to the zone name, not the zone members. The zone members are added to a list of nodes to search for.

The script builds a list of nodes to search for, so you can use any combination of input parameters that define what nodes to look for.

-s: This is the search type. It is applied to all of the inputs (–alias, -alias_f, -wwn, -wwn_f, -zone, and -zone_f). See the handout notes for additional comments and a summary of RegEx and wild card syntax.

-r: When specified, a report with a page similar to the “Logins” sheet produced by report.py is created.

Additional Notes

Keep in mind that nodefind is searching for logins, which are always WWNs, not aliases. Internally, aliases are converted to WWNs. The FOS nodefind command works the same way. If the alias isn't defined, the result will still be "Not found". A future consideration is to differentiate between an alias not being defined vs. the login not being found.

Summary of Wildcard and RegEx Syntax

Search the web for 'python fnmatch.fnmatch' for additional informaiton

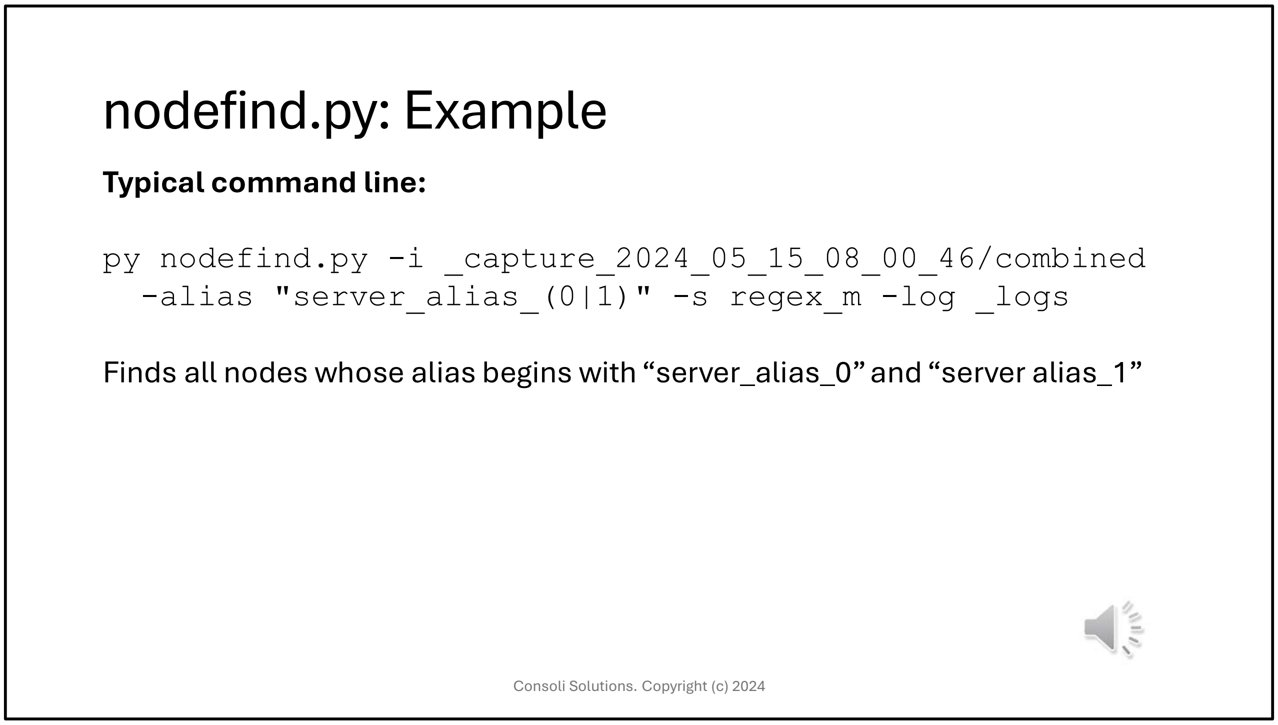
`*`	`matches everything`
`?`	`matches any single character`
`[seq]`	`matches any character in seq`
`[!seq]`	`matches any character not in seq`

Summary of ReGex strings

Search the web for 'regex' for additional information. A regex match must match the begining of the string. A regex search matches any instance of the regex in the string.

`abc…`	`Letters`
`123…`	`Digits`
`\d`	`Any Digit`
`\D`	`Any Non - digit character`
`.`	`Any Character`
`\.`	`Period`
`[abc]`	`Only a, b, or c`
`[ ^ abc]`	`Not a, b, nor c`
`[a - z]`	`Characters a to z`
`[0 - 9]`	`Numbers 0 to 9`
`\w`	`Any Alphanumeric character`
`\W`	`Any Non - alphanumeric character`
`{m}`	`m Repetitions`
`{m, n}`	`m to n Repetitions`
`*`	`Zero or more repetitions`
`+`	`One or more repetitions`
`?`	`Optional character`
`\s`	`Any Whitespace`
`\S`	`Any Non - whitespace character`
`^ …$`	`Starts and ends`
`(…)`	`Capture Group`
`(a(bc))`	`Capture Sub - group`
`(.*)`	`Capture all`

```
(abc | def )  Matches abc or def
```



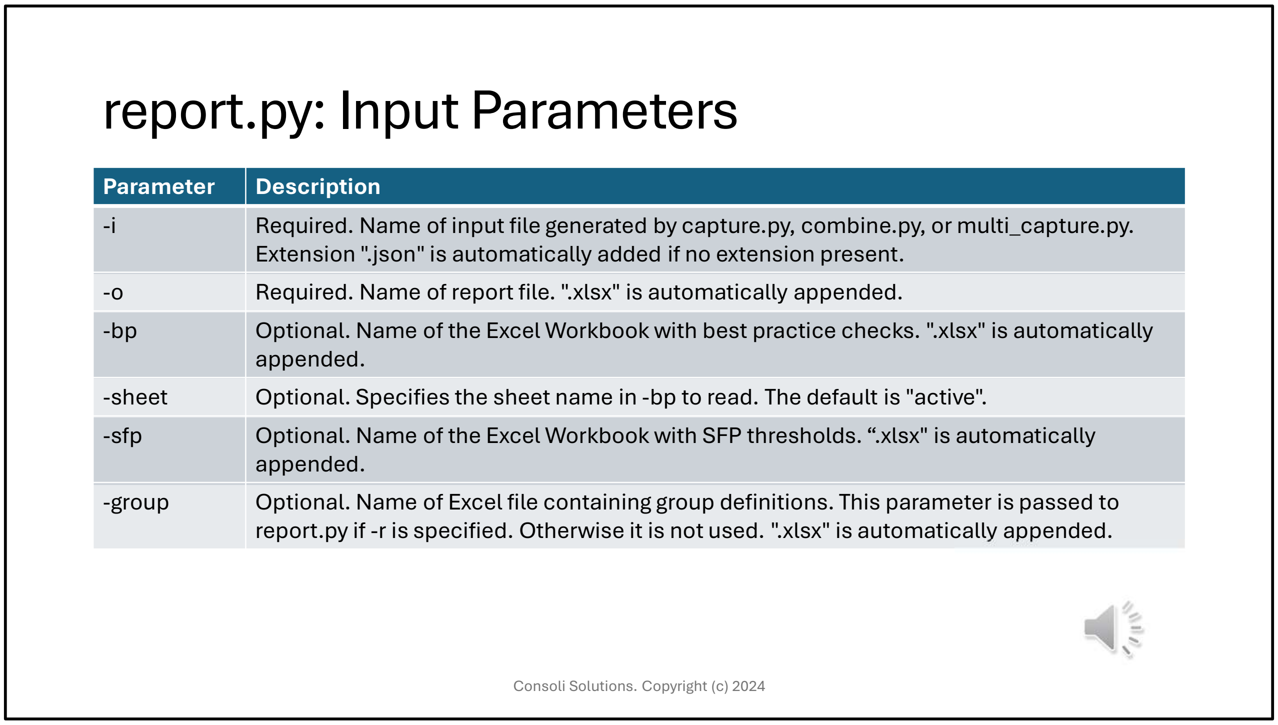
nodefind.py: Example

Typical command line:

```
py nodefind.py -i _capture_2024_05_15_08_00_46/combined
  -alias "server_alias_(0|1)" -s regex_m -log _logs
```

Finds all nodes whose alias begins with “server_alias_0” and “server alias_1”

This is just a reference example.



report.py: Input Parameters

Parameter	Description
-i	Required. Name of input file generated by capture.py, combine.py, or multi_capture.py. Extension ".json" is automatically added if no extension present.
-o	Required. Name of report file. ".xlsx" is automatically appended.
-bp	Optional. Name of the Excel Workbook with best practice checks. ".xlsx" is automatically appended.
-sheet	Optional. Specifies the sheet name in -bp to read. The default is "active".
-sfp	Optional. Name of the Excel Workbook with SFP thresholds. ".xlsx" is automatically appended.
-group	Optional. Name of Excel file containing group definitions. This parameter is passed to report.py if -r is specified. Otherwise it is not used. ".xlsx" is automatically appended.

The output of report.py is general purpose report. Although the format is different and there are some differences in content, it is similar in nature to a SAN Health report. It contains chassis, fabric, switch summaries, port statistics, etc. For mainframe SANs, it also contains reports with RNID data and an IOCP analysis sheet.

-i: When launched from multi_capture.py, this is the folder created + “/combined” which is the output of combine.py.

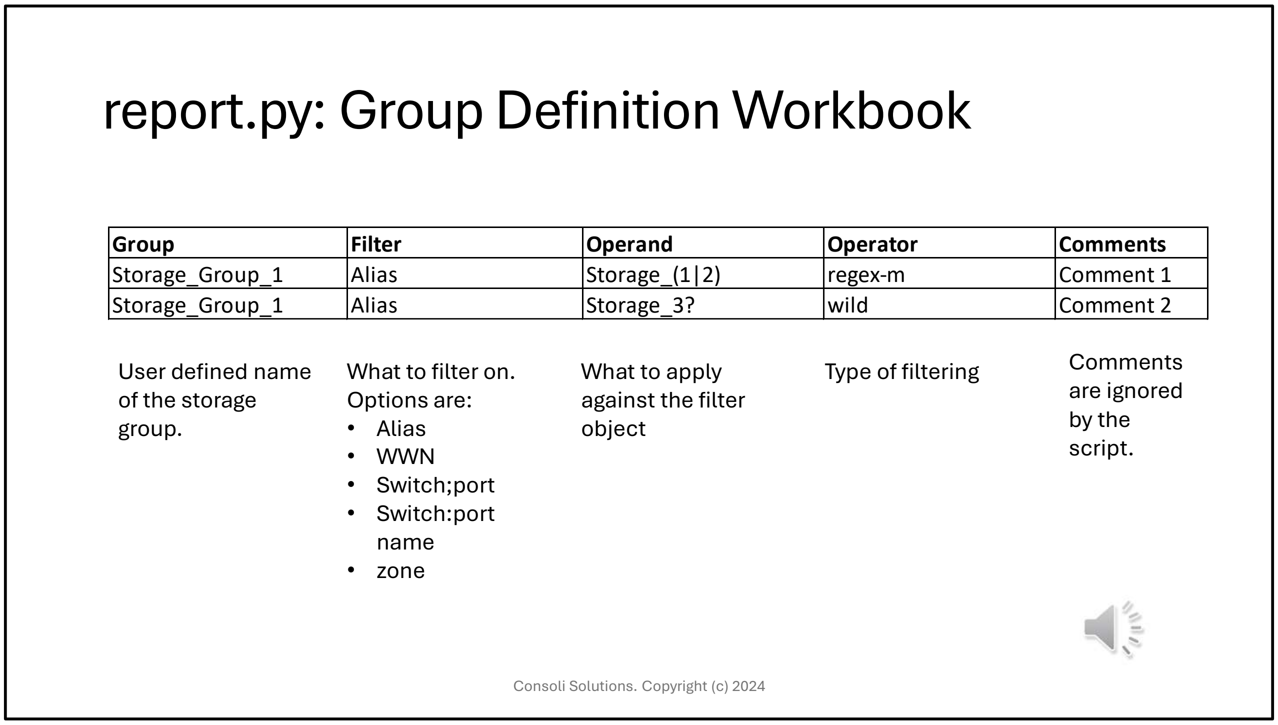
-o: When launched from multi_capture.py, it’s that same folder + “/report_yyyy_mm_dd_hh_mm_ss”

-bp: Typically used. If your just getting started, set this to the example included with brocade-rest-api-applications. The “Instructions” sheet provides details on how to adjust parameters and enable/disable checking of certain items. This is the –bp option specified with multi_capture. If you are just getting started, set this to bp. That workbook is included with brocade-rest-api-applications.

-sheet: This determines what sheet to use for best practice checking. If your just getting started, set this to “rules_std”. This is the –sheet option specified with multi_capture.

-sfp: If your just getting started, set this to “sfp_rules_r12”. That workbook is included with brocade-rest-api-applications. The “Instructions” sheet explains how to make adjustments. This is the –sfp option specified with multi_capture.

-group: Optional. This is the name of the workbook with group definitions. If you’re just getting started, omit this parameter. Everyone has seen a SAN report before. Since this is the only worksheet in the report that’s unique from typical SAN reports, this is going to be the focus of the remainder of the report discussion. This is the –group option specified with multi_capture. If you’re just getting started, leave this option out.



report.py: Group Definition Workbook

Group	Filter	Operand	Operator	Comments
Storage_Group_1	Alias	Storage_(1|2)	regex-m	Comment 1
Storage_Group_1	Alias	Storage_3?	wild	Comment 2

User defined name of the storage group.	What to filter on. Options are: • Alias • WWN • Switch;port • Switch:port name • zone	What to apply against the filter object	Type of filtering	Comments are ignored by the script.

Copy group.xlsx, which is in brocade-rest-api-applications, and edit to your needs. The “Instructions” sheet has syntax details, tips, and other notes.

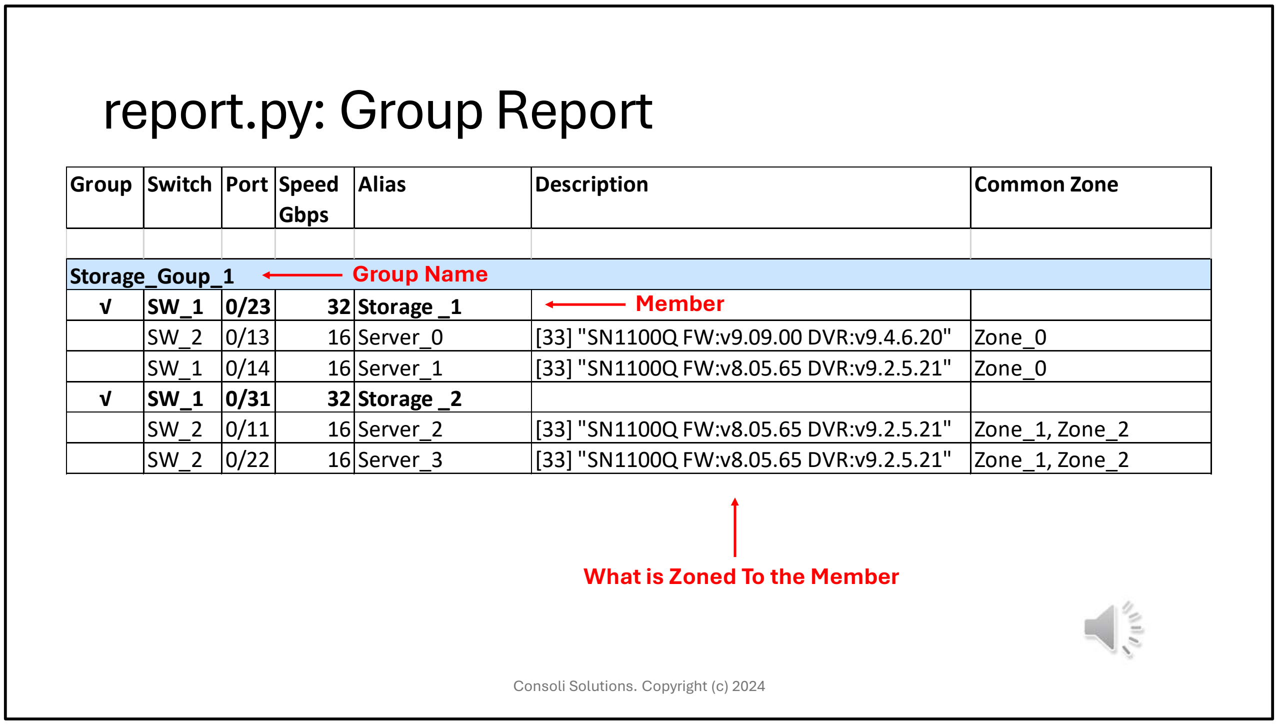
Group: Any length of printable characters is supported. If the same group name appears more than once, the group is added to. In this example, “Storage_Group_1” contains all the RegEx matches for Storage_(1|2) and all the wild card matches for Storage_3?.

Filter: Usually “Alias”. Using WWNs is typically only used with LSAN zones. Details on how to define a group by WWN are in the “Instructions” worksheet.

Operand: This is what to look for. In this example, “Storage_(1|2)” is a RegEx match which means any alias matching Storage_1 or Storage_2. “Storage_3*” is a wild card search and matches any alias that begins with Storage_3. I’m essentially building a nodefind list to define the group.

Operator: This defines how the operand is to be treated. Options are: regex-s (RegEx search), regex-m (RegEx match), wild (standard * and ?), and exact. Zone names are case sensitive. Unless otherwise specified in a RegEx, all matching is case sensitive.

Comments & any other column: As was the case for the multi_capture input workbook, as long as the header remains in row 1 and you don’t reuse any of these header names, you can add whatever other columns you want. Columns are found by name, so the order of columns doesn’t mater.



report.py: Group Report

Group	Switch	Port	Speed Gbps	Alias	Description	Common Zone
Storage_Goup_1						
√	**SW_1**	**0/23**	**32**	**Storage _1**		
	SW_2	0/13	16	Server_0	[33] "SN1100Q FW:v9.09.00 DVR:v9.4.6.20"	Zone_0
	SW_1	0/14	16	Server_1	[33] "SN1100Q FW:v8.05.65 DVR:v9.2.5.21"	Zone_0
√	**SW_1**	**0/31**	**32**	**Storage _2**		
	SW_2	0/11	16	Server_2	[33] "SN1100Q FW:v8.05.65 DVR:v9.2.5.21"	Zone_1, Zone_2
	SW_2	0/22	16	Server_3	[33] "SN1100Q FW:v8.05.65 DVR:v9.2.5.21"	Zone_1, Zone_2

To simply the illustration, not all columns are shown. The group name and member is what was defined in the workbook discussed on the previous slide. The description is a hash of name server data and FDMI data. FDMI is typically only presented with servers. Usually there is something for storage in the name server data but in this case there wasn't any which is why the storage descriptions are blank.

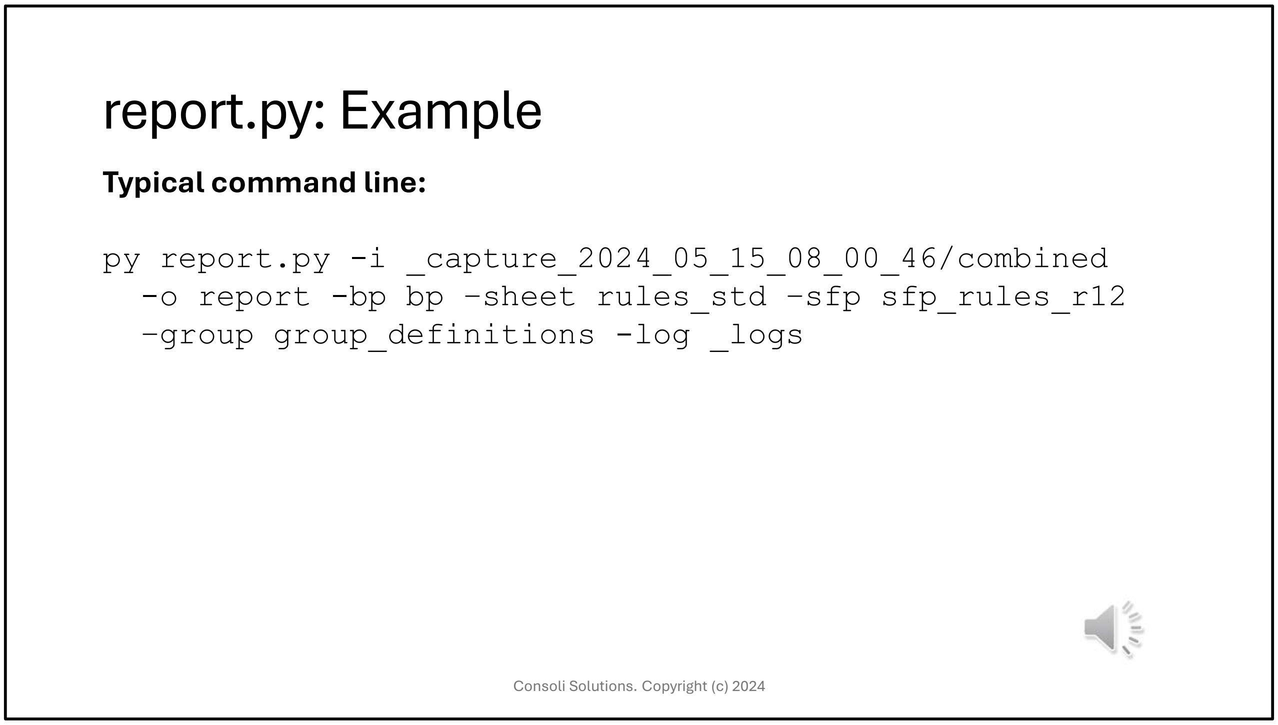
Additional Notes

To simplify the illustration, the following columns are not shown:

- Comments
    - Nearly all reports have a comments column.
    - These are not the comments you entered in the configuration workbook discussed on the previous slide.
    - Anything that effected the associated port, such as bit errors or MAPS alerts, would have a comment
- Port Name
    - The user defined port name
- Zoned To

    - A count of the number of devices zoned to this device which are displayed in the subsequent lines.
- WWN
    - Login WWN

Description: This is a hash of FDMI and nameserver data. Storage, as of this writing anyway, does not provide FDMI data. Usually, there is
something descriptive in the nameserver data, but in this case there wasn’t. The intent of this slide is to discuss what’s in the report. I hacked some customer data, so it’s not technically accurate.



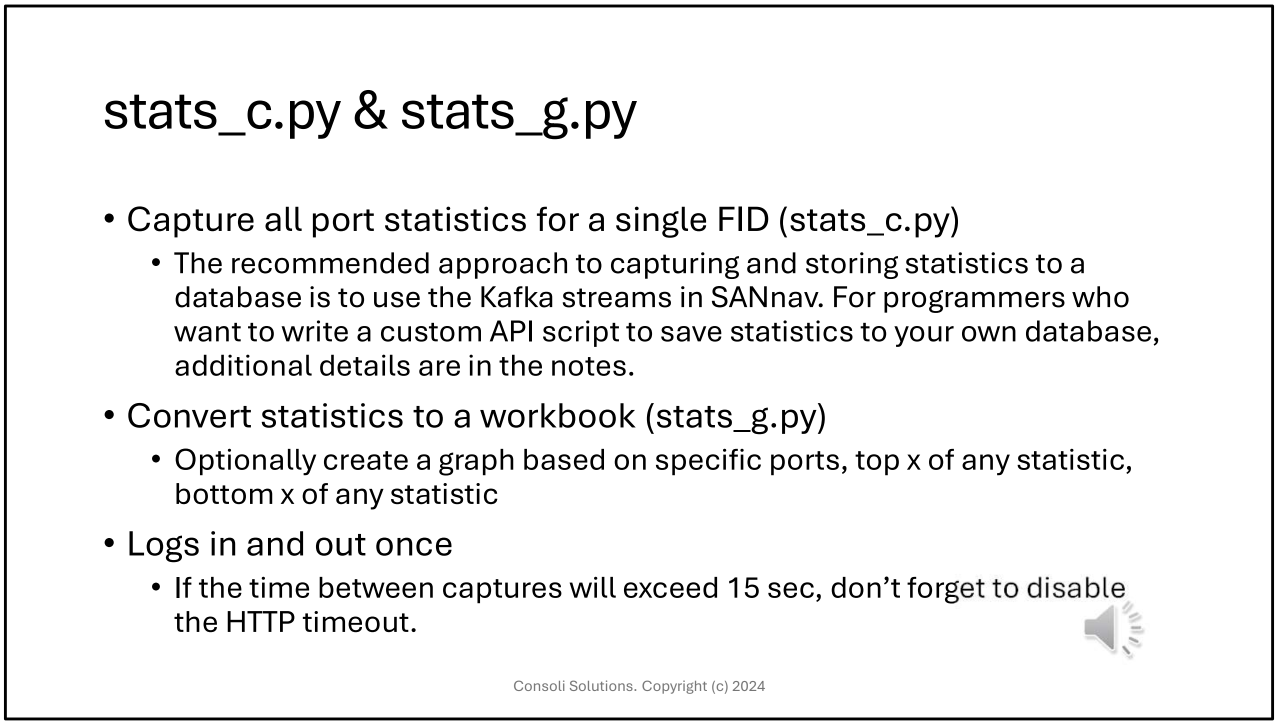
report.py: Example

Typical command line:

```
py report.py -i _capture_2024_05_15_08_00_46/combined
  -o report -bp bp –sheet rules_std –sfp sfp_rules_r12
  –group group_definitions -log _logs
```

Usually, I use multi_capture.py with the –r option to create this report. When I run this script by itself, it's usually because I modified the best practice (-bp), SFP rules (-sfp), or the groups (-group) to look for something specific.

TIP: If you are looking for something specific, copy and modify one of the best practice sheets, set the "check" column to "False" for everything, then set the item you are looking for to "TRUE". The resulting best practice page will only contain issues you are looking for. For example, when trouble shooting one of the first things I look for are C3 discards. To look for just C3 discards, copy the rules_std sheet, set the check column to all "False", then set the cell in the "Check" column to True for: PORT_C3_DISCARD, PORT_TXC3_DISCARD, and PORT_RXC3_DISCARD. The resulting best practice summary sheet in the report will only contain ports with C3 discards.



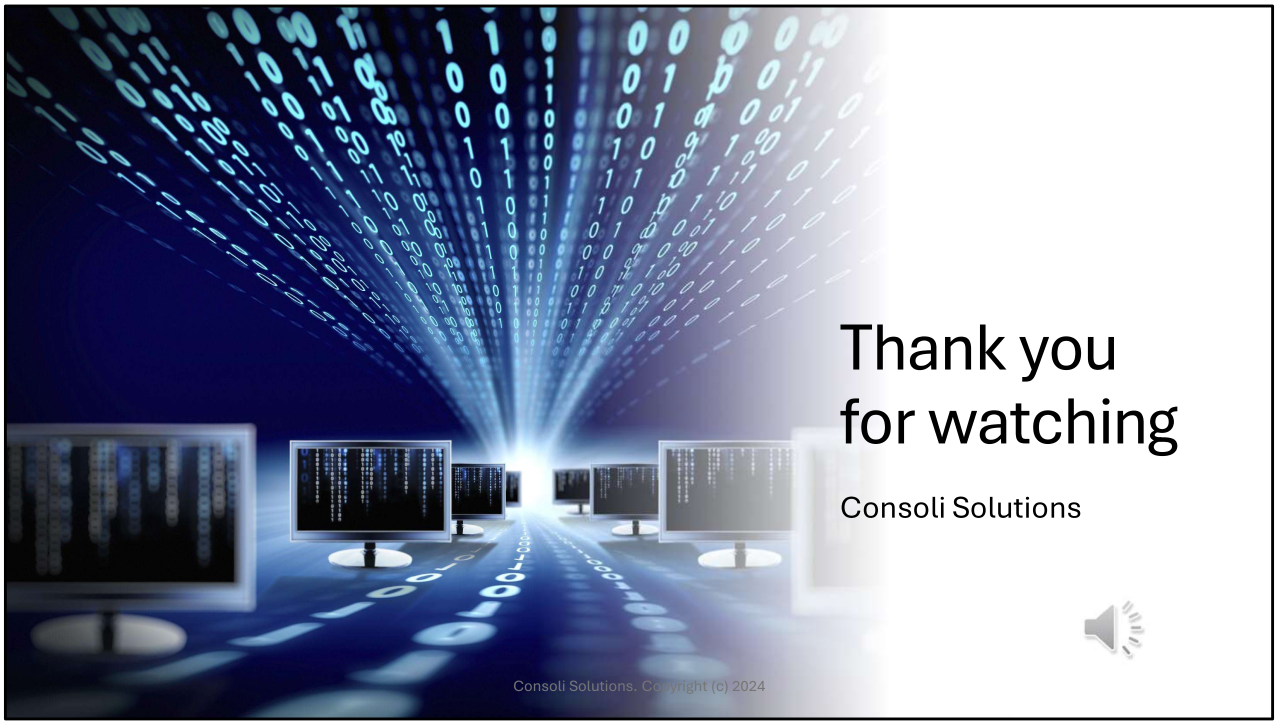
stats_c.py & stats_g.py

- Capture all port statistics for a single FID (stats_c.py)
  - The recommended approach to capturing and storing statistics to a database is to use the Kafka streams in SANnav. For programmers who want to write a custom API script to save statistics to your own database, additional details are in the notes.
- Convert statistics to a workbook (stats_g.py)
  - Optionally create a graph based on specific ports, top x of any statistic, bottom x of any statistic
- Logs in and out once
  - If the time between captures will exceed 15 sec, don't forget to disable the HTTP timeout.

These two scripts are only worth a brief mention.

They were created before SANnav had this capability. At that time, Network Advisor was limited to 20 ports and did not collect statistics for many counters. Although not used much anymore, they are sometimes useful for troubleshooting when you don't know which ports to look at yet or there is a need to capture more data than SANnav is capable of capturing.

Most people use the Kafka Streams from SANNav to capture statistics. If you're looking for a way to capture statistics and add them to your own database via the API, look for stats_to_db_app.py or stats_to_db.py in the sample scripts.



Thank you for watching

Consoli Solutions