Automation

DAY ONE: ENABLING AUTOMATED NETWORK VERIFICATIONS WITH JSNAPY

What happens when you combine JSNAP and Python? You get JSNAPy, a powerful network verification tool that can automate your data collection and verification tasks.

By Premesh Shah

DAY ONE: ENABLING AUTOMATED NETWORK VERIFICATIONS WITH JSNAPY

Every time a network engineer changes the configuration of a running network, there are always two nagging questions: *will it work* and *will it break any existing services?* Although actually changing the configuration might take only 10-20 minutes, a network engineer can end up spending more than 2-3 hours running verification checks.

What if there was a tool that could simplify pre- and post checks, including data collection and verification, and then inform the engineer of the details only if something is wrong, or doesn't match the pre-defined parameters?

JSNAPy is an evolution of the original JSNAP module: the next-gen module for automating network verification. Not only has JSNAPy simplified data collection, it has also reduced the amount of effort needed for verification, and that means your late night cut overs just got a lot shorter, safer, and completely verified. Take this *Day One* book into the lab today and explore JSNAPy.

> *"As our industry moves towards NetDevOps, network verification and audit are key pillars of the automation lifecycle. This excellent book introduces you to JSNAPy, a flexible and powerful tool built on the concepts of its predecessor, JSNAP. Premesh Shah nicely guides you through the journey of using JSNAPy to automate network verifications even if you're not an experienced coder."*
>
> Diogo Montagner
>
> Resident Engineer, Juniper Networks, JNCIE #1050

IT'S DAY ONE AND YOU HAVE A JOB TO DO, SO LEARN HOW TO:

- Understand the importance of data collection and verification.
- Use JSNAPy to automate data collection and verification.
- Enhance your existing network verification process.
- Audit customer networks.
- Integrate JSNAPy with existing network automation tools.

ISBN 978-1-941441-45-9
51600
9 781941 441459

JUNIPER NETWORKS

Day One: Enabling Automated Network Verifications with JSNAPy

By Premesh Shah

JUNIPer® NETWORKS

Published by Juniper Networks Books
Authors: Premesh Shah
Technical Editor: Diogo Montagner
Editor in Chief: Patrick Ames
Copyeditor: Nancy Koerbel
Illustrator: Karen Joice

ISBN: 978-1-941441-45-9 (paperback)
Printed in the USA by Vervante Corporation.

ISBN: 978-1-941441-46-6 (ebook)

Version History: v1, January 2017
2 3 4 5 6 7 8 9 10

http://www.juniper.net/dayone

About the Author
Premesh Shah is a Network Solutions Architect with over 12 years of experience in designing, building, automating, and operating networks in both Service Provider and Enterprise networks. He is JNCIE-SP and JNCIE-ENT.

Author's Acknowledgments
I would like to first dedicate this book to my family: my wonderful and beautiful wife Vibhuti, and my lovely son, Jeremie, for their love, patience, sacrifice, and support while writing this book. They have been very kind and supporting, as always, during my journey.

I also would like to thank the following amazing people and teams for helping me write this *Day One* book: a special thank you to Patrick Ames, Priyal Jain, Nitin Kumar, Diogo Montagner, Timothy Hicks, and Kevin Chao for reviewing and giving me valuable input.

Finally, I would like to thank my mentors and my peers who have guided me and stood by me all these years. Thank you to my manager, Cem Yildirim from Juniper, who has been supportive of what I wanted to do and helped me achieve it. And lastly, but most importantly, thank you to all those special people – my relatives and my friends, who stood by me during the highs and lows of life.

Welcome to Day One

What You Need to Know Before Reading This Book

The author has made few assumptions while writing this book:

- You are a network engineer focused on operation, migration, or design and need to verify network data after every network change.
- You have basic understanding of programming languages.
- You have basic understanding of XML and XPath.
- You are familiar with Junos operational commands and have an understanding on how to read the output of the commands.
- You are willing to simplify your day-to-day work with automation.

What You Will Learn by Reading This Book

After reading this book you will be able to:

- Understand the importance of data collection and verification
- Automate data collection and verification
- Use JSNAPy to automate data collection and verification.
- Enhance your existing network verification process.
- Audit customer networks.
- Integrate JSNAPy in the existing automation tools.

Information Experience

This *Day One* book is singularly focused on one aspect of networking technology. There are other resources available at Juniper Networks, from white papers to webinars to online forums. It's highly recommended that you peruse the technical documentation to become fully acquainted with the configuration process of Junos devices, and to get a better understanding of the configuration process flow. The technical documentation can be located at www.juniper.net/documentation.

Preface

Every time a network engineer changes the configuration of a running network, the first and most important question is – will the change work? The second most important question is – will the change break any existing service? The second most important question is – will the change break any existing service? Although configuring the actual change might take only 10-20 minutes, an engineer may end up spending more than two to three hours doing pre- and post-checks to ensure implemented changes will work.

What if there was a tool that could simplify the effort of pre- and post-checks, including data collection and verification, and let the engineer know the details only if something breaks?

In addition to such pre- and post-checks, regular audits are crucial to maintain any network. Networks need to be audited for everything from simple issues, such as determining whether basic management protocol is configured properly, to more complex concerns, such as whether all the advance protocol services that are configured correspond to a predefined template.

As a Network Solutions Architect with more than twelve years of industry experience automating, designing, operating, and troubleshooting various major networks across the globe, I have worked with different teams such as operation, migration, design, and testing. Each of these teams has a common concern: how can they best collect data and verify that all their existing services are working fine? For any activity, whether it's migration or the addition of a new service or feature, an upgrade cannot be completed without ensuring that existing customer or service has not been disturbed.

JSNAPy has been a great help to me in facilitating data collection and verification of networks. As a part of the evolution of JSNAP, JSNAPy is the next-gen module to automate network verification. Not only has JSNAPy simplified data collection, it has also reduced the amount of effort needed for verification. Verification, which can normally take anywhere from 30 minutes to two hours, has been drastically reduced to a few minutes.

This *Day One* book introduces JSNAPy to the reader and shares the experience I gained while working on JSNAPy. It also covers moving from JSNAP to JSNAPy. This move can be executed smoothly and the next chapter begins by showing you how JSNAPy can automate collection and verification of networks. Become a part of this journey by using JSNAPy and you will reduce the risk, effort, and time needed for your own network verifications.

Premesh Shah, January 2017

Chapter 1

Automating Data Collection and Verification of Networks

Collecting network output data (state of the network) before and after every network activity is an important part of network operations because of the potential to break any existing services. Best practice is to collect the output of configuration and log files, as well as to collect the output of all the commands that provide evidence the services are running, before the activity starts, and then to perform the same process again, once the activity is finished.

Once you have captured the data showing the test results and outputs of key commands, verification can be fairly simple for the technical expert. The normal way to verify is to analyze the data and check any abnormal log, events, or differences between the output of the before and after results. If you encounter any issues, you need to follow your contingency plan, which is typically a rollback plan resulting in either full or partial rollback, depending on the activity. If you are reading this book, then you should be familiar with the flow chart shown in Figure 1.1.

Why Automate Collection and Verification?

When initiating any change in the network, you need to first decide on what information to collect, and then ask yourself if you are collecting enough. This process can be repetitive and time consuming, but it is important. It is considered a best practice to collect network information and its status in order to verify that everything is working fine and there are no surprises after any change or update order.

NOTE Collecting data is a way of grabbing a snapshot of the state of the network just before the change. Remember you need to collect data before the activity *and* after the activity.

Automation simplifies the data collection process, as well as assuring the network engineer that the data collection is error free while reducing the time it takes to collect it. JSNAPy is a tool that can help you automate data collection in addition to verifying the snapshots.

Verification provides evidence that the network activity performs its intended functions and meets all requirements listed in the Method of Procedure (MOP), functional, and allocated baselines. Verification is a key risk-reduction activity in the implementation and integration of a feature, or a service, and enables you to catch any defects proactively.

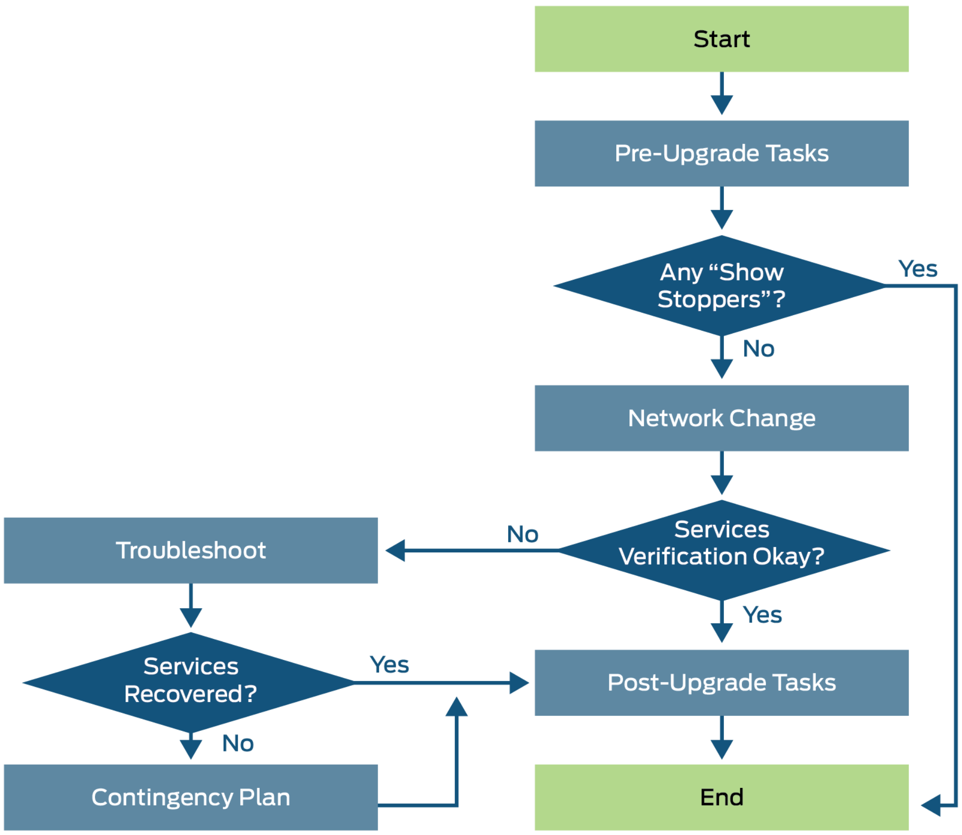


Figure 1.1 Change Management Flow Chart

data collection. It can be based on single data collection output or the comparison of two sets of data collection output. Verification of services, although time consuming, gives the assurance or guarantee that the activity and the objective is successful.

For any activity, whether it's testing a new feature or upgrading the OS, the actual time it takes to execute that activity is typically 10-20% of the allotted maintenance window time, whereas verification can take more than 40-50% of that time.

Most of the planned network activity is done after work hours, hence the verification is done either late at night or early in the morning, so a network engineer can be exhausted by the time network verification begins. There are a large number of lines of output to examine at these late hours, so automating the verification process can prevent incidents without waking up management.

There are a few tools in the market that can compare two files and print out the difference between them, but they are often not intelligent enough to identify a sequence change or a genuine increment of the data (like uptime, etc.).

So it is a must to automate the verification. What you need is a tool that is an open standard, that knows the sequences, and that can understand what information should be used to compare pre-activity and post-activity data. That tool is *JSNAPy*.

Pros and Cons of Automation

Automation, a word used so often today, is important, but it is not simple to automate everything because automation comes with cost and effort. Let’s quickly review the pros and cons.

The pros of automation include:

- Reducing the time and effort needed during activity.
- Reducing the need to have an expert conducting the testing. With automation an expert need only be involved if the test fails and extra validation is needed.
- Increasing the accuracy and reducing missed errors.
- Increasing efficiency by running more tests in a short time frame.
- Proactively identifying the issues.
- Increasing cost savings and product quality.

The cons of automation include:

- It's time consuming and requires an up-front investment.
- It requires a different skill set than traditional network testing.
- Additional testing is needed to verify if the automation is working as expected.
- Developers should have visibility of each and every use case.
- It's a continuous process.

Four Target Areas to Automate

While automation has its challenges, the benefits far outweigh the initial investment, in part because the scale of most verification tasks requires automation. To enhance the success of your automation project, target the following to get the most benefits:

- Automate the highly repetitive tasks first.
- Automate the things that need accuracy.
- Automate time-consuming tasks to reduce the activity time.
- Automate the task that needs human expertise.

BEST PRACTICE Take the time to study and detail what you should automate in your network, and why, before attempting to employ automation.

JSNAP and JSNAPy

Let's look at a tool that can help you automate the most important and repetitive complex data collection and verification.

JSNAP, and its next-gen version, JSNAPy, are applications that collect before and after snapshots, perform analysis, and report failures; they *snapshot the system* before and after any changes, compare the results, and quickly identify impactful service issues.

JSNAP uses SLAX (originally developed by Juniper) to code and develop the script that collects the data and then verifies that data based on configured criteria. JSNAP works well with almost all Juniper devices.

The next-gen version, JSNAPy is based on open standard Python 2.7 and PyEZ to achieve the same goal as JSNAP with many additional enhancements. This *Day One* book focuses on JSNAPy.

MORE? For information on JSNAP, please refer to an excellent book, *Day One: Using JSNAP to Automate Network Verifications*, by Diogo Montagner, available here: http://www.juniper.net/us/en/training/jnbooks/day-one/automation-series/using-jsnap-automate-network-verifications/.

Why JSNAPy?

JSNAP was written with SLAX, whereas JSNAPy uses Python, which is more user-friendly than SLAX and provides more flexibility. Important reasons that JSNAPy uses Python are its openness and its ability to customize and integrate with any existing automation. For example, if you have an existing operations support system (OSS) already implemented, JSNAPy can easily be integrated as a module to your existing implementations.

By the way, the name JSNAPy means that it's *J*unos *SNAP*shot administrator with *P*ython. Hence, JSNAP plus Python = JSNAPy.

JSNAPy will help you:

- Collect data to check and verify it.
- Automate the verification of data intelligently and quickly.
- Simplify the whole process of data collection and verification.
- Customize and integrate network verification into your existing network systems.

Move From JSNAP to JSNAPy (Tool)

Existing users of JSNAP can stay with JSNAP if they wish, but I suggest moving to JSNAPy.

To make your life easier during this transition, Juniper has provided an built-in tool that converts the original JSNAP test file format into the new JSNAPy test file format, which is from SLAX to YAML.

The tool is called *jsnap2py*, it is shipped with JSNAPy, and jsnap2py will convert the slax configuration file *test_interface.conf* into yaml file *test_interface.yml*.

Let's begin:

```
jsnap2py -i test_interface.conf
```

NOTE If you want to assign a different output file name, then use `-o` option.

```
jsnap2py -i test_interface.conf -o test_interface.yml
```

Here's what to add on the CLI:

```
premesh:jsnapy premesh$ jsnap2py
usage: jsnap2py [-h] [-i INPUT] [-o OUTPUT]
```

This converts JSNAP test to JSNAPy YAML file:

```
optional arguments:
  -h, --help          show this help message and exit
  -i INPUT, --input INPUT
                     JSNAP test file
  -o OUTPUT, --output OUTPUT
                     jsnapy test file
```

And here is the configuration file:

```
premesh:samples premesh$ more trial.conf
do {
   show_chassis_re;
   show_chassis_fpc;
}
show_chassis_re {
    command show chassis routing-engine;
    iterate route-engine {
      id slot;
        no-diff mastership-state {
      info "Checking Routing Engine mastership state.";
      err "The status of Routing Engine %s has changed from %s to %s", $ID.1,$PRE/mastership-
state, $POST/mastership-state;
      }
   }
}
show_chassis_fpc {
    command show chassis fpc;
    iterate fpc {
      id slot;
        no-diff state {
      info "Checking the FPC state.";
      err "The state of FPC %s has changed from %s to %s.", $ID.1, $PRE/state,$POST/state;
      }
   }
}
```

The command with input and output file:

```
premesh:samples premesh$ jsnap2py -i trial.conf -o trial.yml
premesh:samples premesh$
```

Confirming if the file is created or not:

```
premesh:samples premesh$ ls trial*
trial.conf   trial.yml
```

And the new JSNAPy YAML file:

```
premesh:samples premesh$ more trial.yml
show_chassis_fpc:
- command: show chassis fpc
- iterate:
    id: slot
    tests:
    - err: The state of FPC {{id_1}} has changed from {{pre["/state"]}} to {{post["/
state"]}}.
      info: Checking the FPC state.
      no-diff: state
    xpath: fpc
show_chassis_re:
- command: show chassis routing-engine
- iterate:
    id: slot
    tests:
    - err: The status of Routing Engine {{id_1}} has changed from {{pre["/mastership-
state"]}}
        to {{post["/mastership-state"]}}
      info: Checking Routing Engine mastership state.
      no-diff: mastership-state
    xpath: route-engine
tests_include:
- show_chassis_re
- show_chassis_fpc
```

NOTE As you can see, the sequence in which the test file is converted is not as we define it in the JSNAPy YAML file, the `tests_include` is defined at the end. But don't worry, the YAML test file will work as expected as YAML will take care of sequencing.

MORE Want to know more? Refer to: https://github.com/Juniper/JSNAPy/wiki/7.-jsnap2py.

Let's dive deeper into JSNAPy and get to know all the components and how to put them together. Get into your lab and follow along. It's day one and you have a job to do, and by Chapter 4 you'll be able to collect and verify network events using some simple JSNAPy examples.

Chapter 2

JSNAPy Components

This chapter discusses the different components used by JSNAPy. It is divided into four main sections:

- How JSNAPy communicates to your device
- File structure
- JSNAPy core operation
- What's new in JSNAPy

How JSNAPy Communicates to Your Device

JSNAPy communicates with devices to take a snapshot of the required configuration or operational commands by using PyEZ, which uses SSH to establish the connection. It uses NETCONF to retrieve the configuration data information as shown in Figure 2.1.

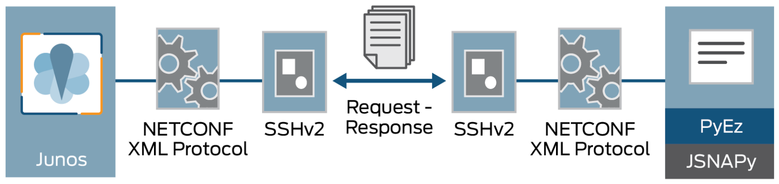


Figure 2.1 How JSNAPy Communicates

JSNAPy makes use of NETCONF XML Protocol (RFC6241) and Secure TCP/IP connections via SSHv2 (RFC6242). See https://tools.ietf.org/html/rfc6242 for more information.

NETCONF

The NETCONF protocol is defined in IETF RFC 6241 (https://tools.ietf.org/html/rfc6241) as a mechanism by which a network device can be managed, configuration data retrieved, and new configuration data uploaded. The NETCONF protocol allows the device to expose a full, formal application programming interface (API). Applications can use this straightforward API to send and receive full and partial configuration data sets.

The NETCONF protocol uses a remote procedure call (RPC) paradigm. A client encodes an RPC in XML and sends it to a server using a secure, connection-oriented session. The server responds with a reply encoded in XML.

Extensible Markup Language (XML)

XML is a markup language that defines a set of rules for encoding documents in a format that is both human- and machine-readable.

The design goals of XML emphasize simplicity, generality, and usability across the Internet. It is a textual data format widely used for the representation of arbitrary data structures. All major routing vendors support XML to configure, retrieve, or modify code on the router.

MORE? Visit https://www.w3.org/XML for more information about XML.

YAML

YAML is a human-readable data serialization language that takes concepts from programming languages such as C, Perl, Python, and Ruby. See RFC0822 (MAIL), RFC1866 (HTML), RFC2045 (MIME), RFC2396 (URI), XML, SAX, SOAP, and JSON.

YAML is an acronym for *YAML Ain't Markup Language*. Early in its development, YAML was said to mean *Yet Another Markup Language* but it was then reinterpreted to distinguish its purpose as data-oriented, rather than document markup.

MORE? Visit http://yaml.org/ for more information about YAML.

Where Do You Get JSNAPy?

Installing JSNAPy with pip

Using pip is the recommended way to install JSNAPy. If you already have pip available, you can simply execute:

```
pip install jsnapy
```

Installing JSNAPy from source

This approach works regardless of the operating system (JSNAPy works well with Windows and UNIX-like systems). You can get the source code either by downloading a source and extracting it, or by cloning the GitHub repository at http://github.com/Juniper/jsnapy.

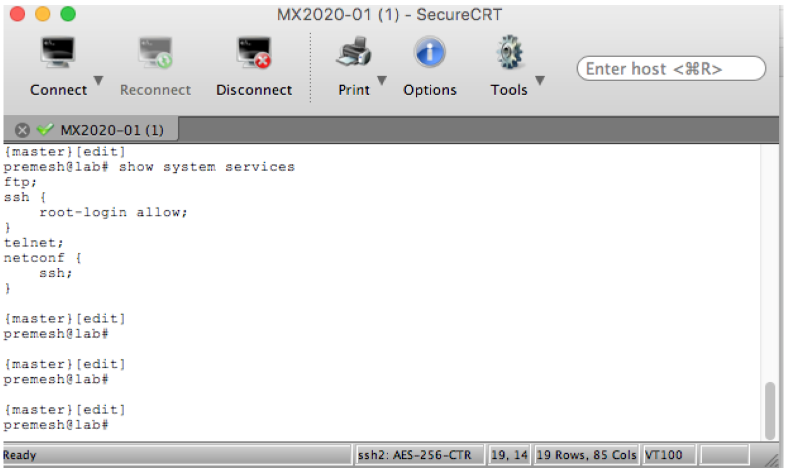
Device Readiness to Support JSNAPy

Your Juniper device should be configured as follows:

- Support SSH
- Support NETCONF over SSH
- Port 22 and 830 should be allowed

NOTE You can use `port` to specify your own port in devices.yml.

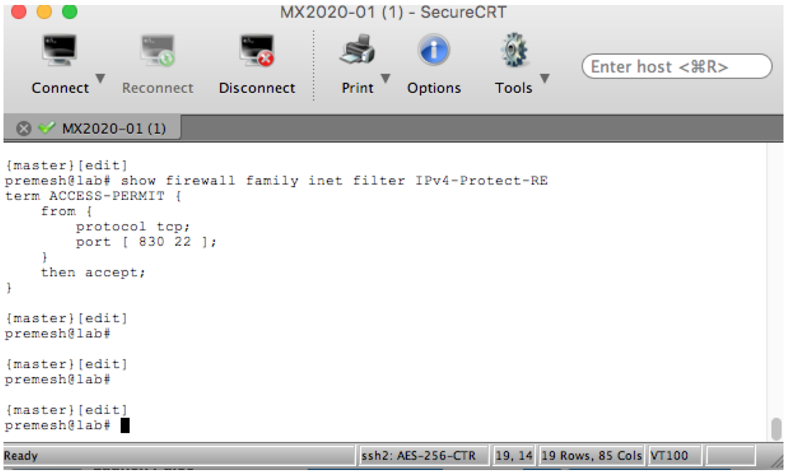
Figures 2.2 and 2.3 show a sample configuration from a Juniper router.



```
{master}[edit]
premesh@lab# show system services
ftp;
ssh {
    root-login allow;
}
telnet;
netconf {
    ssh;
}

{master}[edit]
premesh@lab#

{master}[edit]
premesh@lab#

{master}[edit]
premesh@lab#
```

```
Ready    ssh2: AES-256-CTR    19, 14    19 Rows, 85 Cols    VT100
```

Figure 2.2 Configuring SSH and NETCONF over SSH



```
{master}[edit]
premesh@lab# show firewall family inet filter IPv4-Protect-RE
term ACCESS-PERMIT {
    from {
        protocol tcp;
        port [ 830 22 ];
    }
    then accept;
}

{master}[edit]
premesh@lab#

{master}[edit]
premesh@lab#

{master}[edit]
premesh@lab#
```

Figure 2.3 Allowing Ports 830 and 22

Application Structure

Let's quickly look at the different files and folders used for JSNAPy. The default location for JSNAPy is: /etc/jsnapy/.

NOTE In a forthcoming version of JSNAPy, you will have a modifier that can allow you to override this information in runtime mode – it is the *-folder* option.

The location /etc/jsnapy contains the following folders:

- samples: Folder containing sample examples
- snapshots: Folder containing all snapshots
- testfiles: Folder containing all test cases (.yml)

And /etc/jsnapy contains the following files:

- jsnapy.cfg: It contains default path for snapshot, test file, and config file.
- logging.yml: It stores the default logging capability; can be modified.

The /etc/jsnapy/snapshots contains the following file extensions.

- *.xml: These are the XML replies returned by the devices upon collecting snapshots using snap and snapcheck operations.

And the /etc/jsnapy/testfiles contains the following file extensions.

- *.yml: yml is the language used to define the requirements. There can be two types of file, a *configuration* file or a *test* file. See the distinctions in the table here:

Configuration file	Test file
Define hostname (can be single or multiple).	Define test to be included.
Pointer to the test files.	Pointer to the test module within the file.
Inform if database or mailing service is required.	Provide commands, iterate, and test scenario.
Can be directly called inside module.	Define the info and error messages.

- *.py: This is the python script that can call a JSNAPy config file as a module, so that JSNAPy can be integrated to any existing automation system. We will learn about this in the upcoming chapter, "Working With JSNAPy Towards Automation."

NOTE Details about the content of these files will be explained later in this chapter.

JSNAPy Core Operation

JSNAPy uses a group of YAML scripts to achieve below command line operation:

- *snap:* As the name suggests, this command takes the snapshot of a given command or RPC during the time the script is executed. This is the operator used to collect snapshots before, during, or after the activity.
- *check:* This command compares two snapshots (this book uses pre- and post-activity throughout) based on given test cases. This command needs two copies of the snapshot and then compares the two based on criteria provided (there are a rich group of operators that can be used to compare).
- *snapcheck:* This is the combination of snap and check, taking the best of both worlds and doing it on runtime. This command is very useful to check some predefined conditions on runtime or in offline mode. This command takes snapshots of given commands or RPCs and then compares them against predefined criteria mentioned in test cases.

NOTE Typically, *snapcheck* is used for network audits where you want to check if the state of the device complies with a set of golden rules (built on top of a *golden config*). A golden config is a template configuration version used in compliance management as an ideal configuration against which similar devices can be compared. Any differences found are recorded as compliance evaluation failures by default.

- *diff*: As the name suggests, this is the Diff tool, it is very similar to the UNIX command diff. It basically compares two snapshots word by word without any test operator. The best part is that it can compare snapshots in both XML or text format.

Deep Dive With JSNAPy Configuration

Let's take a deep dive inside the JSNAPy configuration to understand what's needed for a successful execution of JSNAPy.

A configuration file is a combination of different blocks. Let's discuss each block in detail, and later, when you look at the example, you will see how the different blocks work together.

Hosts Code Block

The hosts code block lists the device credentials for where the test will be executed. You can either define the device (in case there is only one device) or include a .yml file, which has multiple devices segregated as group.

Here is an example of a hosts code block with a single device:

```
hosts:
  - device: 10.10.1.2
    username : demo
    passwd: demo123
```

And here is an example of a hosts code block, which includes a .yml file called `devices` with multiple device credentials:

```
hosts:
  - include: devices.yml
    group: MX
```

Tests Code Block

The tests code block lists the names of each .yml file that has tests that will be executed in the snapshot:

```
tests:
  - test_is_equal.yml
  - chassis_hardware.yml
  - pic_status.yml
```

Database Code Block

Database code blocks have the connectivity details of the database that is used to store the data and compare the info. By default, a text file is created and stored to compare the info:

```
# can use sqlite to store data and compare them
sqlite:
  - store_in_sqlite: True
    check_from_sqlite: True
    database_name: jbb.db
    compare: 1,0
```

Mail Code Block

The `mail` code blocks list the .yml file that has all the credentials to send the mail of JSNAPy results:

```
# can send mail by specifying mail
mail: send_mail.yml
```

Tests_include Code Block

The `tests_include` code block lists the name of the test that will be executed in the snapshot:

```
tests_include:
  - check_core
```

Check_Core - Test Definition Code Block

The `check_core` code block is the actual configuration where you are going to define the tests and their criteria of pass or fail. It is the combination of many components, so let's review in order to understand each one:

- The section name: Unique, user-defined string that should describe the check being performed.
- One command statement: Specifies the Junos OS operational mode command that is executed to collect the data.
- Section declarations: One or more items or iterate content section declarations that define the test cases used to evaluate the data.
- XPath expression: XPath is a syntax for defining parts of an XML document that then uses path expressions to navigate in XML documents. The output collected by a JSNAPy snap is an XML document and XPath defines the scope of the content that will be queried and searched.

- Test operator: Each test case is defined by a test operator followed by any required parameters. You will learn more about test operator later on in this chapter.
- The `err` statement: Defines one or more error statements, which are generated when the content fails the specific test case.
- The `info` statement: Provides information about the test case and expected operating conditions within the test-case code block.

Here is an example of an actual configuration with notes in bold face:

```
check_core:   <<<< Section name
  - command: show system core-dumps routing-engine both <<< command statement
  - iterate:   <<<< section declarations
      xpath: '//multi-routing-engine-results/multi-routing-engine-item/directory-
list'  <<<< XPath expression
      tests:
          - not-exists: ./directory/file-information/file-name  <<<< Test operator
            err: "Test  Failed!!core file exist <{{post['./directory/file-information/
file-name']}}> on {{post['./../re-name']}}"  <<<<  err
            info: "Test Succeeded!!!There is no core file"  <<< info
```

Logging Code Block

Logging is configured under the `logging.yml` file. This file is used to define log file, location, severity (level), and format.

Level can be INFO, ERROR, CRITICAL, or DEBUG. The default logging level is INFO, which is heavily used in this book, but you may notice in some places the author has changed the logging level to DEBUG.

TIP If you just want to print the `err` message, INFO is enough, but if you need the info message also you'll want to change the logging level to DEBUG.

There are a few additional rules for a JSNAPy configuration:

- Indent is very important in YAML file
- Use spaces instead of tabs
- Use hash (#) to insert comments into the configuration file

These are the different code blocks of a JSNAPy configuration. Now let's use the code blocks to learn about the different files that are used to execute JSNAPy.

Writing the Config File

The config file is the combination of the device details and the test file. The config file supports the following code blocks:

- `hosts`: Use a separate .yml file with all the device credentials and import it in the config file with the keyword included.
- `tests`: Specify test files that you want to run where you have written all the conditions. You can also specify the test conditions in a config file but it's not a good practice.
- `sqlite`: This is a free database that can be used to store the snapshots. Use this key if you want to compare and store snapshots in a database. You can use any other database like SQL, or Oracle, just follow the same step as you do in Python for database connectivity.
- `mail`: This is an optional and cool feature. This option will send an email to the predefined user upon the completion of a JSNAPy script with the details of which tests have been executed and the results.

Let's take a look at some real examples.

For connecting with a single device:

```
hosts:
  - device: 10.10.1.24
    username : demo
    passwd: demo
tests:
  - test_no_diff.yml
  - test_bgp_neighbor.yml

# (optional) use only when you want to store and compare
# snapshots from database
sqlite:
  - store_in_sqlite: True
     check_from_sqlite: True
    database_name: jbb.db
    compare: 1,0
# (optional) use when you want to send mail about test results
mail: send_mail.yml
```

For connecting with multiple devices:

```
# for multiple devices with database
hosts:
   - include: devices.yml
     group: MX
tests:
   - test_is_equal.yml
   - test_is_in.yml
# (optional) use only when you want to store and compare snapshots
# from database
sqlite:
   - store_in_sqlite: yes
     check_from_sqlite: yes
     database_name: jbb.db
# (optional) use when you want to send mail about test results
mail: send_mail.yml
```

And, devices.yml:

```
MX:
- 10.20.1.20:
     username: root
     passwd: root123
- 10.21.13.14:
     username: root
     passwd: root123
- 10.20.6.26:
     username: jsnapy
     passwd: jsnapy123
EX:
  - 10.2.15.210:
     username: root
     passwd: root123
QFX:
 - 10.29.1.24:
      username: abc
      passwd: pqr
- 10.29.6.1:
      username: abc
      passwd: pqr123
```

Writing the Test File

The purpose of writing a test file is to specify commands or RPCs of whose snapshot is to be taken, which nodes you want to test, and how to test them. Test files use the following code blocks:

- `tests_include`: this is an optional argument to be used if you want to include only explicit test cases. If you do not include this tag, then by default JSNAPy will run all test cases.
- The `check-test` definition code block: This block defines the test criteria and execution plan. (Not shown in examples.)
- `command/rpc`: You can give either a `command` or `rpc` to perform the testing, but most of the scripts will be a combination of command and RPC.

- format: You can specify output [text,xml] but mostly you'll format as XML, which is also the default, but in the case of diff, you can also use text. Note that for comparing text output, only the diff option is supported.
- args: these are optional configuration items that are exclusively used with RPC only.
  - filter_xml: can provide filtered output
  - Other arguments such as the following examples:

```
1.   - rpc: get-config
     - args:
           filter_xml: configuration/system/login
2.   - rpc: get-interface-information
         format: text
     - args:
           interface-name: em0
           media: True
           detail: True
```

When there is big chunk of data and only a small portion of it is required to identify the issue, use of args eases the work. For example, there can be thousands of interfaces configured in the router but we just need to understand if the management interface is working fine.

NOTE Confused on which to use, the Junos OS command or RPC? I personally think it is better to use the Junos command because it will be easily readable by others, especially those who are not familiar with Junos RPCs.

- item/iterate: You can have multiple iterates or items under one command or rpc and each command can use either the iterate or the item, the difference being that item will execute the test only once whereas iterate will keep executing the test until all the conditions have been matched. Note that you use item only if you want to use the first node in xpath alone, and you use iterate if want to keep iterating for all nodes in XPath.
- tests: Now this is the place or section that specifies test cases. A test file can have multiple test cases inside iterate and item, and a test file can have multiple iterates or items. As described earlier, in code blocks there are just three golden blocks to define inside test, which are:

```
           test-operator <condition>      ---CONDITION
           info <CUSTOMIZED MESSAGE>      --- output if test pass
           err  < CUSTOMIZED MESSAGE >    -- output if test fails.
Example1:
           tests:
                   - is-equal: //minimum-time, 60
                     info: "Test Succeeded!!, minimum-time now is equal to <{{post['//
minimum-time']}}>"
                    err: "Test Failed!!!, minimum-time is not equal to 60, it is
<{{post['//minimum- time']}}>"
```

- Variable definition: Use to compare and print; the value from the output, pre and post, can be used in conjunction with the XPath value. This is very useful when you want to print both values, the before and the after. Note that JSNAPy uses *pre* for first snapshot and *post* for second snapshot irrespective of the name given by the user.

Let's stop here and review some real examples in order to test your comprehension so far.

Sample test file1 test_is_equal.yml

The following test checks to confirm that the `admin-status` of interface ge-1/2/1 is equal to up. If the admin-status is up then the test has passed and it will print an info message. If the admin-status is not equal to up then it will print an err message:

```
test_interfaces_terse:
   - command: show interfaces terse ge*
   - item:
       id: ./name
       xpath: //physical-interface[normalize-space(name) = "ge-1/2/1"]
       tests:
          - is-equal: admin-status, up
            info: "Test Succeeded !! admin-status is equal to:
                   <{{post['admin-status']}}> with oper-status
                   <{{post['oper-status']}}>"
            err: "Test Failed !! admin-status is not equal to up, it
                  is <{{post['admin-status']}}> with oper-status
                  <{{post['oper-status']}}>"
```

Sample test file2

Here, multiple iterators are used inside one command and multiple tests inside one iterator. The example checks various fields of output for the `show bgp neighbor` command and the RPC `get-bgp-neighbor-information`:

```
tests_include:
    - test_command_bgp
    - test_rpc_bgp

test_command_bgp:
    - command: show bgp neighbor
    - iterate:
          xpath: '//bgp-information/bgp-peer/bgp-option-information'
          tests:
             - is-gt: holdtime, 10
               err: "Test Failed!! holdtime is not greater than 10, it
                     is: <{{post['holdtime']}}>"
               info: "Test succeeded!! holdtime is greater than 10, it
                      is: <{{post['holdtime']}}>"
             - is-lt: preference, 200
               err: "Test Failed!! preference is not less than 10,
                     <{{post['preference']}}>"
               info: "Test succeeded!! preference is less than 10,
                      <{{post['preference']}}>"
test_rpc_bgp:
    - rpc: get-bgp-neighbor-information
    - iterate:
          xpath: '//bgp-information/bgp-peer'
          tests:
             - not-equal: last-state,Idle
               err: "Test Failed!! last state is <{{post['last-
                     state']}}>"
               info: "Test succeeded!! last state is not equal to
                      idle, it is: <{{post['last-state']}}>"
             - all-same: flap-count
               err: "Test Failed!!! flap count are not all same!!, it
                     is <{{post['flap-count']}}> "
               info: "Test Succeeded!! flap count are all same, it is
                      now <{{post['flap-count']}}>!!!"
             - is-equal: flap-count, 0
               err: "Test Failed!!! flap count is not equal to 0, it
                     is: <{{post['flap-count']}}> "
               info: "Test Succeeded!! flap count is equal to
                      <{{post['flap-count']}}> !!"
```

How to Use JSNAPy

So far you have learned a lot about what JSNAPy can do, why you need to use it, and the resulting benefits. Now it's time to learn *how* to actually use it. The rest of this chapter will take you on a practical tutorial covering:

- Taking snapshots with JSNAPy
- Checking current configuration using JSNAPy
- Comparing two snapshots
- Writing config files
- Writing test files

Taking Snapshots Using JSNAPy:

As explained previously, JSNAPy snapshots are a collection of outputs from the router that represent its state at a specific point in time. The following steps show you how to write a JSNAPy test file to collect snapshots. You will learn how to write a config (#1) file and test(#2) file:

1. Write the config file containing details about the device's login credentials and the test files to be included. (You already learned about these details of config code blocks in this chapter's *Deep Dive* section :

```
config_snap.yml

hosts:
  - device: 10.209.1.2
    username : demo
    passwd: demo123
tests:
  - test_is_equal.yml
```

2. Write the test file containing the details about the commands, RPCs and test operators to be used. (Again refer to this chapter's *Deep Dive* section):

```
test_is_equal.yml
test_interfaces_terse:
  - command: show interfaces terse xe*
  - item:
      id: ./name
      xpath: //physical-interface[normalize-space(name) = "xe-1/2/3"]
      tests:
        - is-equal: admin-status, up
          info:  "Test Succeeded !! admin-status is equal to <{{post['admin-
status']}}> with oper-status
                      <{{pre['oper-status']}}>"
          err:  "Test Failed !! admin-status is not equal to down, it is <{{post['admin-
status']}}> with
                      oper-status <{{pre['oper-status']}}>"
```

3. Take a snapshot using JSNAPy:

```
premesh# jsnapy --snap pre -f config_snap.yml
Connecting to device 10.209.1.2 ................
Taking snapshot for show interfaces terse lo* ................
sh-3.2#
```

You have learned about `snap` in this chapter's *JSNAPy Core Operation* section. Note that `pre` is the file name that will be used to store the XML output of the command, whereas `-f` is used to define the configuration file called `config_snap.yml`. Here are the different options that can be used with `snap`:

```
premesh:snapshots premesh$ jsnapy --snap
positional arguments:
  pre_snapfile          pre snapshot filename
  post_snapfile         post snapshot filename

optional arguments:
  -h, --help          show this help message and exit
  --snap             take the snapshot for commands specified in test file
  --check             compare pre and post snapshots based on test operators specified in
test file
  --snapcheck           check current snapshot based on test file
  --diff                display the difference between two snapshots
  -V, --version         displays version
  -f FILE, --file FILE  config file to take snapshot
  -t HOSTNAME, --hostname HOSTNAME
                        hostname
  -p PASSWD, --passwd PASSWD
                        password to login
  -l LOGIN, --login LOGIN
                        username to login
  -P PORT, --port PORT  port no to connect to device
  -v, --verbosity       Set verbosity
                        -v: Debug level messages
                        -vv: Info level messages
                        -vvv: Warning level messages
                        -vvvv: Error level messages
                        -vvvvv: Critical level messages
premesh:snapshots premesh$
```

Now the question is, where are these snapshot output files stored?

The answer is that all snapshots are stored in the */etc/jsnapy/snapshots* folder by default. This is configurable in version 1.1 where you can override /etc/jsnapy/jsnapy.cfg.

Snapshots file names are formed by using following values: file_name = `<device_name>_<snap_tag>_<command/RPC>.<xml/text>`:

```
premesh# cd /etc/jsnapy/snapshots/
premesh# ls
 choc-mx480-b_post_show_interfaces_terse_lo_.xml
```

Here is an example of a snapshot, in an XML version of the CLI output, collected from the router using the JSNAPy test file:

```
premesh# cat 10.10.1.2_pre_show_interfaces_terse_lo_.xml
<interface-information style="terse">
<physical-interface>
<name>
lo0
</name>
<admin-status>
up
</admin-status>
```

```
<oper-status>
up
</oper-status>
<logical-interface>
<name>
lo0.0
</name>
<admin-status>
up
</admin-status>
<oper-status>
up
</oper-status>
<filter-information>
</filter-information>
<address-family>
<address-family-name>
inet
</address-family-name>
<interface-address>
<ifa-local>
127.0.0.1
</ifa-local>
<ifa-destination emit="emit">
0/0
</ifa-destination>
</interface-address>
</address-family>
</logical-interface>
</physical-interface>
</interface-information>
```

Checking the Current Configuration Using JSNAPy

Let's learn how to use JSNAPy to check the current configuration, something that can be used for auditing purposes or simply to validate the baseline. Let's use `snapcheck`, the combination of `snap` plus `check`. Snapcheck uses the predefined parameter to verify the collected output.

1. Write the configuration file containing details about the device's login credentials and the test files to be included:

```
config_snapcheck.yml

hosts:
  - device: 10.10.1.2
    username : demo
    passwd: demo123
tests:
  - test_is_equal.yml
```

2. Write the test file containing the details about the commands, RPCs and test operators to be used:

```
test_is_equal.yml
test_interfaces_terse:
  - command: show interfaces terse lo*
  - item:
      id: ./name
      xpath: //physical-interface[normalize-space(name) = "lo0"]
      tests:
        - is-equal: admin-status, up
          info:  "Test Succeeded !! admin-status is equal to <{{post['admin-
status']}}> with oper-status
                      <{{pre['oper-status']}}>"
          err:  "Test Failed !! admin-status is not equal to down, it is <{{post['admin-
status']}}> with
                      oper-status <{{pre['oper-status']}}>"
```

NOTE `Normalize-space` is used to collapse whitespace in a string name.

3. Now, check the current configuration using `snapcheck`:

```
premesh# jsnapy --snapcheck pre -f config_snapcheck.yml
Connecting to device 10.10.16.204 ......................
Taking snapshot for show interfaces terse lo* ..........
******Performing test on Device: 10.10.16.204**********
Tests Included: test_interfaces_terse
********Command is show interfaces terse lo*************
--------Performing is-equal Test Operation--------------
Test Succeeded !! admin_status is equal to <up> with oper_status <up>
Final result of is-equal: PASSED
------------------- Final Result!! ---------------------
Total No of tests passed: 1
Total No of tests failed: 0
Overall Tests passed!!!
```

Compare Two Snapshots Using Check (Test Operator is Defined)

Now let's compare the specific values from the output (snapshot) taken during two *different* time periods, using `check` to compare the pre-defined criteria with the value of the pre- and post-files.

1. Write the configuration file containing details about the device's login credentials and the test files that are to be included:

```
config_check.yml

hosts:
  - device: 10.209.1.2
    username : demo
    passwd: demo123
tests:
  - test_no_diff.yml
```

2. Write the test file containing the details about the commands, RPCs and test operators to be used:

```
test_no_diff.yml
test_command_version:
  - command: show interfaces terse
  - iterate:
      xpath: physical-interface
      id: './name'
      tests:
        - no-diff: oper-status       # element in which test is performed
          err: "Test Failed!! oper-status  got changed, before it was <{{pre['oper-
status']}}>, now it is <{{post['oper-
status']}}> with name <{{id_0}}> and admin status <{{post['admin-status']}}>"
          info: "Test succeeded!! oper-
status is same with value, before it is <{{pre['oper-status']}}> now it is <{{post['oper-
status']}}> with admin status <{{post['admin-status']}}> "
```

3. Take the pre-snapshot using the `--snap` command:

```
premesh# jsnapy --snap pre -f config_check.yml
Connecting to device 10.209.1.2 ................
Taking snapshot for show interfaces terse lo* ................
premesh#
```

4. Take the post snapshot using the `--snap` command:

```
premesh# jsnapy --snap post -f config_check.yml
Connecting to device 10.209.1.2 ................
Taking snapshot for show interfaces terse lo* ................
premesh#
```

5. Check pre- and post-snapshots using the `--check` command:

```
premesh# jsnapy --check pre post -f config_check.yml
**********Performing test on Device: 10.209.16.206******************
Tests Included: test_command_version
***********Command is show interfaces terse*************************
----------------------Performing no-diff Test Operation----------------------
Test Failed!! oper_status  got changed, before it was <['up']>, now it is <['down']> with
name <['demux0']> and admin status <up>
Final result of no-diff: FAILED
---------------------------- Final Result!! -------------------------------
Total No of tests passed: 0
Total No of tests failed: 1
Overall Tests failed!!!
```

Compare Two Snapshots Using Diff (Test Operators Not Required)

In this example, JSNAPy uses the UNIX `diff` operator to compare two snapshots, so it doesn't need any test operators.

1. Write the configuration file containing the details about the device's logging credentials and the test files to be included:

```
Config_diff.yml
hosts:
  - device: 10.209.1.2
    username : demo
    passwd: demo123
tests:
  - test_no_diff.yml
```

2. Write the test file containing the details about the command /RPC and the test operators to be used:

```
test_no_diff.yml
test_command_version:
  - command: show interfaces terse
```

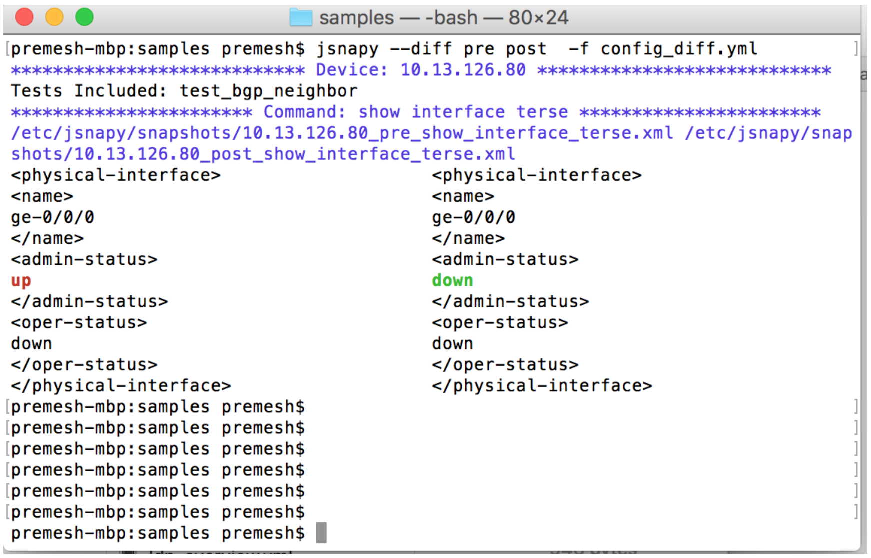
3. Take the pre- snapshot using the `--snap` command:

```
premesh# jsnapy --snap pre -f config_single_snapcheck.yml
Connecting to device 10.209.1.2 ................
Taking snapshot for show interfaces terse lo* ................
premesh#
```

4. Take the post-snapshot using the `--snap` command:

```
premesh# jsnapy --snap post  -f config_single_snapcheck.yml
Connecting to device 10.209.1.2 ................
Taking snapshot for show interfaces terse lo* ................
premesh#
```

5. Compare the two snapshots word by word as shown in Figure 2.4, without any test operator.



```
premesh-mbp:samples premesh$ jsnapy --diff pre post  -f config_diff.yml
**************************** Device: 10.13.126.80 ****************************
Tests Included: test_bgp_neighbor
*********************** Command: show interface terse ***********************
/etc/jsnapy/snapshots/10.13.126.80_pre_show_interface_terse.xml /etc/jsnapy/snap
shots/10.13.126.80_post_show_interface_terse.xml
<physical-interface>                    <physical-interface>
<name>                                  <name>
ge-0/0/0                                ge-0/0/0
</name>                                 </name>
<admin-status>                          <admin-status>
up                                      down
</admin-status>                         </admin-status>
<oper-status>                           <oper-status>
down                                    down
</oper-status>                          </oper-status>
</physical-interface>                   </physical-interface>
premesh-mbp:samples premesh$
premesh-mbp:samples premesh$
premesh-mbp:samples premesh$
premesh-mbp:samples premesh$
premesh-mbp:samples premesh$
premesh-mbp:samples premesh$
premesh-mbp:samples premesh$
```

Figure 2.4 Comparing Two Snapshots

CAUTION When using XML output, only the XML element contents are compared in a string fashion.

Simple, yes? The selection of the words is so user friendly that a non-developer can pick it up quite easily. Let's move on and spend a little time studying the operators.

Chapter 3

JSNAPy Operators

JSNAPy has a wide variety of operators, including but not limited to numeric and string. Throughout this chapter you will learn about operators starting with the basic description of all operators and then work through examples and use cases for each of them.

Supported Test Operators

So far you have learned about the test file and you have seen some of the operators used, but now it's time to learn what operators can be used inside the test file. Please review the collection of lists detailed in Tables 3.1 through 3.4.

NOTE For comparing current configuration against some predefined criteria (only one snapshot is required):

Table 3.1 Execute Tests Over Elements with Numeric or String Values

OPERATOR	DESCRIPTION
`all-same`	Checks if all content values for the specified element are the same. It can also be used to compare all content values against another specified elements.
`is-equal`	Checks if the value (integer or string) of the specified element matches a given value.
`not-equal`	Checks if the value (integer or string) of the specified element does not match a given value.

`exists`	Verifies the existence of an XML element in the snapshot.
`not-exists`	Verifies XML element should not be present in snapshot.
`contains`	Determines if an XML element string value contains the provided test-string value.

Table 3.2 Execute Tests Over Elements with Numeric Values

OPERATOR	DESCRIPTION
`is-gt`	Checks if the value of a specified element is greater than a given numeric value.
`is-lt`	Checks if the value of a specified element is less than a given numeric value.
`in-range`	Checks if the value of a specified element is within the given numeric range.
`not-range`	Checks if the value of a specified element is outside of a given numeric range.

Table 3.3 Execute Tests over Elements with String Values

OPERATOR	DESCRIPTION
`contains`	Determines if an XML element string value contains the provided test-string value.
`is-in`	Checks if the specified element string value is included in a given list of strings.
`not-in`	Checks if the specified element string value is NOT included in a given list of strings.

Table 3.4 Test Operators to Compare Values of Two Snapshots

OPERATOR	DESCRIPTION
`no-diff`	Compares data elements present in both snapshots, and verifies if their value is the same.
`list-not-less`	Checks if the item is present in the first snapshot but not present in the second snapshot.
`list-not-more:`	Checks if the item is present in the second snapshot but not present in the first snapshot.
`delta`	Compares the change in value of an element to a delta. The delta can be specified as: 1. an absolute percentage 2. positive, negative percentage 3. an absolute fixed value 4. a positive negative fixed value

NOTE JSNAPy 1.1 will have support for `no-diff-in` type of operators that don't fail the test if the XML Path specified does not exist in both pre- and post- snapshots. This is explained in the details in the next section.

Okay, now you have a list of operators that can be used by JSNAPy. Refer back to them when needed, or print out the tables for reference as you see them in operation and determine the places where they can be best used.

NOTE This book uses only one protocol for all examples – OSPF – in order to explain the operator without digging into the complexity of networking. The complete JSNAPy configuration is provided in the Appendix.

Name: all-same

Description: Check if all content values for the specified element are the same. It can also be used to compare all content values against another specified element.

Example: `test_neighbor_state.yml`

```
ospf-neighbor:
- command: show ospf neighbor
- iterate:
    id: interface-name
    xpath: ospf-neighbor
    tests:
    - all-same: ospf-neighbor-state
      err: 'All OSPF neighbor on {{post["interface-name"]}} to
            {{post["neighbor-address"]}} are not in same state, state change
            from  {{pre["ospf-neighbor-state"]}} to
            {{post["ospf-neighbor-state"]}}'
      info: 'No state change for OSPF neighbors'
```

CLI

```
# State during the pre snapshot
premesh@lab> show ospf neighbor
Address          Interface              State     ID               Pri  Dead
10.0.31.213      ae0.0                  Full      10.0.31.240      128    37
169.254.61.61    ae1.0                  Full      10.0.31.232      128    31

# State during the post snapshot
premesh@lab> show ospf neighbor
Address          Interface              State     ID               Pri  Dead
10.0.31.213      ae0.0                  Full      10.0.31.240      128    31
169.254.61.61    ae1.0                  Exchange 10.0.31.232      128    35
```

JSNAPy Output

```
Tests Included: ospf-neighbor
************************ Command: show ospf neighbor ************************
OSPF neighbor on ae1.0 to 169.254.61.61 state change from  Full to Exchange
FAIL | Value of all "ospf-neighbor-state" at xpath "ospf-
neighbor" is not same [ 1 matched / 1 failed ]
------------------------------- Final Result!! -------------------------------
Total No of tests passed: 0
Total No of tests failed: 1
Overall Tests failed!!!
```

Use case: Note that `all-same` is a very important operator for any check, and can be used for almost anything when you need to confirm whether before and after are the same. For example: IGP peering, OS version, network state, etc.

Name: is-equal

Description: Check if the value (integer or string) of the specified element matches a given value.

Example: ospf-neighbor.yml

```
ospf-neighbor:
- command: show ospf neighbor
- iterate:
    id: interface-name
    xpath: ospf-neighbor
    tests:
       - is-equal: ospf-neighbor-state, Full
         err: OSPF neighbor on {{post["interface-name"]}} to
              {{post["neighbor-address"]}} is not up, rather
              {{post["ospf-neighbor-state"]}}
         info: All OSPF neighbors are up
```

CLI

```
# State during the pre snapshot
premesh@lab> show ospf neighbor
Address          Interface               State     ID               Pri  Dead
10.0.31.213      ae0.0                   Full      10.0.31.240      128    37
169.254.61.61    ae1.0                   Full      10.0.31.232      128    31

# State during the post snapshot
premesh@lab> show ospf neighbor
Address          Interface               State     ID               Pri  Dead
10.0.31.213      ae0.0                   Full      10.0.31.240      128    31
169.254.61.61    ae1.0                   Exchange 10.0.31.232      128    35
```

JSNAPy output

```
Tests Included: ospf-neighbor
************************ Command: show ospf neighbor ************************
OSPF neighbor on ae1.0 to 169.254.61.61 is not up, rather Exchange
FAIL | All "ospf-neighbor-state" is not equal to "Full" [ 1 matched / 1 failed ]
------------------------------- Final Result!! -------------------------------
Total No of tests passed: 0
Total No of tests failed: 1
Overall Tests failed!!!
```

Use case: Note that `is-equal` is a valuable audit operator, checking the value against predefined values and the use cases could be protocol status, interface description, route community, etc.

Name: not-equal

Description: Check if the value (integer or string) of the specified element does not match a given value.

Example: `test_ospf_neighbor.yml`

```
ospf-neighbor:
- command: show ospf neighbor
- iterate:
    id: interface-name
    xpath: ospf-neighbor
    tests:
    - not-equal: ospf-neighbor-state, Full
      err: 'All OSPF neighbors state is Full'
      info: 'OSPF neighbor on {{post["interface-name"]}} to {{post["neighbor-
address"]}} is not Full, rather {{post["ospf-neighbor-state"]}}'
```

CLI

```
# State during the pre snapshot
premesh@lab> show ospf neighbor
Address          Interface              State     ID               Pri  Dead
10.0.31.213      ae0.0                  Full      10.0.31.240      128    37
169.254.61.61    ae1.0                  Full      10.0.31.232      128    31

# State during the post snapshot
premesh@lab> show ospf neighbor
Address          Interface              State     ID               Pri  Dead
10.0.31.213      ae0.0                  Full      10.0.31.240      128    31
169.254.61.61    ae1.0                  Exchange 10.0.31.232       128    35
```

JSNAPy output

```
Tests Included: ospf-neighbor
************************ Command: show ospf neighbor ************************
---------------------Performing not-equal Test Operation---------------------
All OSPF neighbors are up
OSPF neighbor on ae1.0 to 169.254.61.61 is not up, rather Exchange
FAIL | All "ospf-neighbor-state" is equal to "Full" [ 1 matched / 1 failed ]
------------------------------- Final Result!! -------------------------------
Total No of tests passed: 0
Total No of tests failed: 1
Overall Tests failed!!!
```

Use case: Note that `not-equal` is the opposite of `is-equal`, but like `is-equal`, `not-equal` checks the value against a predefined value. The use case could be for protocol status, interface description, route community, etc.

Name: exists

Description: verifies the existence of an element in the snapshot.

Example: `test_ospf_authentication.yml`

```
ospf-md5:
- command: show configuration protocols ospf | display inheritance |
           display xml
- iterate:
    xpath: protocols/ospf/area/interface
    id: name
    tests:
    - exists: authentication
      err: ' OSPF interface {{post["name"]}} is configured without
             authentication.'
      info: All OSPF interface has authentication enabled
```

CLI

```
premesh@lab> show configuration protocols ospf
area 0.0.0.0 {
    interface lo0.0;
    interface ae0.0 {
        authentication {
            md5 1 key "$9$5znCO1hKMXtuMX7-2gTz36tuBIEyev"; ## SECRET-DATA
        }
    }
    interface ae1.0 {
        authentication {
            md5 1 key "$9$IYWhyKX7V4aUM8aUjH5TRhSrM8xNdsgo"; ## SECRET-DATA
        }
    }
    interface ge-1/2/3.0 {
        authentication {
            md5 1 key "$9$H.fz9AOhSe36SevW-dk.P536Ctu1Ec"; ## SECRET-DATA
        }
    }
}
```

JSNAPy Output

```
Tests Included: ospf-md5
***************** Command: show configuration protocols ospf *****************
 OSPF interface lo0.0 is configured without authentication.
FAIL | All "authentication" do not exists at xpath "protocols/ospf/area/
interface" [ 3 matched / 1 failed ]
------------------------------- Final Result!! -------------------------------
Total No of tests passed: 0
Total No of tests failed: 1
Overall Tests failed!!!
```

Use case: Note that `exists` best suits the use case where you need to check and verify whether or not specified information is present, or to check for configuration audits like authentication, NTP, or SYSLOG.

Name: not-exists

Description: Verifies that a given element should not be present in the snapshot.

Example: ospf-md5.yml

```
ospf-md5:
- command: show configuration protocols ospf | display inheritance |
           display xml
- iterate:
    id: name
    xpath: protocols/ospf/area/interface
    tests:
    - not-exists: authentication
      err: 'OSPF interface {{post["name"]}} has authentication
            enabled'
      info: 'OSPF interface {{post["name"]}} is configured without
            authentication.'
```

CLI

```
premesh@lab> show configuration protocols ospf
area 0.0.0.0 {
    interface lo0.0;
    interface ae0.0 {
        authentication {
            md5 1 key "$9$5znCO1hKMXtuMX7-2gTz36tuBIEyev"; ## SECRET-DATA
        }
    }
    interface ae1.0 {
        authentication {
            md5 1 key "$9$IYWhyKX7V4aUM8aUjH5TRhSrM8xNdsgo"; ## SECRET-DATA
        }
    }
    interface ge-1/2/3.0 {
        authentication {
            md5 1 key "$9$H.fz9A0hSe36SevW-dk.P536Ctu1Ec"; ## SECRET-DATA
        }
    }
}
```

JSNAPy Output

```
Tests Included: ospf-md5
***************** Command: show configuration protocols ospf *****************
---------------------Performing not-exists Test Operation---------------------
OSPF interface lo0.0 is configured without authentication.
OSPF interface ae0.0 has authentication enabled
OSPF interface ae1.0 has authentication enabled
OSPF interface ge-1/2/3.0 has authentication enabled
FAIL |  "authentication" exists at xpath "protocols/ospf/area/
interface" [ 1 matched / 3 failed ]
------------------------------- Final Result!! -------------------------------
Total No of tests passed: 0
Total No of tests failed: 1
Overall Tests failed!!!
```

Use case: Note that `not-exists` best suits the use cases where you need to check and verify whether or not information is present, or to check for configuration audits like authentication, NTP, or SYSLOG.

Name: contains

Description: Checks if the given value (integer or string) of the specified element is present in a snapshot.

Example: `test_interface_name.yml`

```
ospf-md5:
- command: show configuration protocols ospf | display inheritance | display
           xml
- iterate:
    id: name
    xpath: protocols/ospf/area/interface
    tests:
    - contains: name, -
      err: ' OSPF interface {{post["name"]}} seems not a physical interface.'
      info: All OSPF interface is physical interface.
```

CLI

```
premesh@lab> show configuration protocols ospf
area 0.0.0.0 {
    interface lo0.0;
    interface ae0.0 {
        authentication {
            md5 1 key "$9$5znCO1hKMXtuMX7-2gTz36tuBIEyev"; ## SECRET-DATA
        }
    }
    interface ae1.0 {
        authentication {
            md5 1 key "$9$IYWhyKX7V4aUM8aUjH5TRhSrM8xNdsgo"; ## SECRET-DATA
        }
    }
    interface ge-1/2/3.0 {
        authentication {
            md5 1 key "$9$H.fz9A0hSe36SevW-dk.P536Ctu1Ec"; ## SECRET-DATA
        }
    }
}
```

JSNAPy Output

```
Tests Included: ospf-md5
***************** Command: show configuration protocols ospf *****************
 OSPF interface lo0.0 seems not a physical interface.
 OSPF interface ae0.0 seems not a physical interface.
 OSPF interface ae1.0 seems not a physical interface.
FAIL | All "name" do not contains -" [ 1 matched / 3 failed ]
------------------------------- Final Result!! -------------------------------
Total No of tests passed: 0
Total No of tests failed: 1
Overall Tests failed!!!
```

Use case: The best use case for `contains` is to check whether or not a specific value is present, for example, if you wanted to check for a community in a specific route.

Name: is-gt

Description: Check if the value of a specified element is greater than a given numeric value.

Example: `Check_ospf_neighbor.yml`

```
ospf_interface:
- command: show ospf interface
- iterate:
    xpath: ospf-interface[contains(interface-name, "*e*")]
    tests:
    - is-gt: neighbor-count, 0
      err: OSPF interface {{post["interface-name"]}} does not have any
           neighbors
      info: OSPF interfaces must have at least 1 neighbor
```

CLI

```
premesh@lab> show ospf interface
Interface           State   Area            DR ID           BDR ID          Nbrs
ae0.0               PtToPt  0.0.0.0         0.0.0.0         0.0.0.0         1
ae1.0               PtToPt  0.0.0.0         0.0.0.0         0.0.0.0         1
lo0.0               DRother 0.0.0.0         0.0.0.0         0.0.0.0         0
```

JSNAPy Output

```
Tests Included: ospf_interface
************************ Command: show ospf interface ************************
OSPF interface lo0.0 does not have any neighbors
Pass | All "neighbor-count" is greater than  "0" [ 2 matched / 1 Passed ]
------------------------------- Final Result!! -------------------------------
Total No of tests passed: 1
Total No of tests failed: 0
Overall Tests failed!!!
```

Use case: Note that `is-gt` best suits a situation where you want to make sure the value is less than the specific value. A best use case would be checking to ensure that maximum capacity, such as the number of MACs or the number of IP routes, is not exceeded.

Name: is-lt

Description: Check if the value of a specified element is less than a given numeric value.

Example: `check_neighbor_count.yml`

```
ospf_interface:
- command: show ospf interface
- iterate:
    xpath: ospf-interface
    tests:
    - is-lt: neighbor-count, 1
      err: 'OSPF interfaces must have at least 1 neighbor'
      info: 'OSPF interface {{post["interface-name"]}} does not have any
            neighbors'
```

CLI

```
premesh@lab> show ospf interface
Interface           State   Area            DR ID           BDR ID          Nbrs
ae0.0               PtToPt  0.0.0.0         0.0.0.0         0.0.0.0         1
ae1.0               PtToPt  0.0.0.0         0.0.0.0         0.0.0.0         1
lo0.0               DRother 0.0.0.0         0.0.0.0         0.0.0.0         0
```

JSNAPy Output

```
Tests Included: ospf_interface
************************ Command: show ospf interface ************************
OSPF interface lo0.0 does not have any neighbors
Pass | All "neighbor-count" is greater than “0" [ 2 matched / 1 Passed]
------------------------------- Final Result!! -------------------------------
Total No of tests passed: 1
Total No of tests failed: 0
Overall Tests failed!!!
```

Use case: Note that `is-lt` best suits a situation when you want to make sure the value is less than the specific value. A good use case would be checking to ensure that the maximum capacity, such as the number of MACs or number of IP routes, is *not* reached.

Name: in-range

Description: Check if the value of a specified element is in a given numeric range.

Example: `test_link_count.yml`

```
OSPF-DB:
- command: show ospf database detail
- iterate:
    xpath: '/ospf-database-information/ospf-database/ospf-router-lsa'
    id: ../advertising-router
    tests:
    - in-range: link-count, 5, 10
      err: Router {{post["../advertising-router"]}} has {{post["link-
           count"]}} links
      info: OSPF router links [5..10]
```

CLI

```
premesh@lab> show ospf database detail | match "Router|link count"
Router   10.0.27.254      10.0.27.254      0x800001f5  1396  0x22 0xc619   84
  bits 0x0, link count 18
Router   10.0.31.232      10.0.31.232      0x80000448  1413  0x22 0x7a0    84
  bits 0x0, link count 5
Router   10.0.31.240      10.0.31.240      0x8000044a  1400  0x22 0x3aa1   84
  bits 0x0, link count 5
Router  *10.0.31.241      10.0.31.241      0x80000446  1287  0x22 0x6c81   84
  bits 0x0, link count 5
```

JSNAPy Output

```
Tests Included: OSPF-DB
********************** Command: show ospf database detail **********************
Router 10.0.27.254 has 18 links
FAIL | All "link-count" is not in range:  "5.000000 - 10.000000" [ 3 matched / 1 failed ]
------------------------------- Final Result!! -------------------------------
Total No of tests passed: 0
Total No of tests failed: 1
Overall Tests failed!!!
```

Use case: Note that `in-range` is best suited for conditions where the absolute value is not compulsory, but the result is in a range the test deems passable. Think about active routes or routes learned from protocol.

Name: not-range

Description: Check if the value of a specified element is outside of a given numeric range.

Example: `check_link_count.yml`

```
OSPF-DB:
- command: show ospf database detail
- iterate:
    xpath: '/ospf-database-information/ospf-database/ospf-router-lsa'
    id: ../advertising-router
    tests:
    - not-range: link-count, 6, 10
      err: Router {{post["../advertising-router"]}} has
           {{post["link-count"]}} links.
      info: OSPF router links not[6..10]
```

CLI

```
premesh@lab> show ospf database detail | match "Router|link count"
Router   10.0.27.254      10.0.27.254      0x800001f5  1396  0x22 0xc619  84
  bits 0x0, link count 8
Router   10.0.31.232      10.0.31.232      0x80000448  1413  0x22 0x7a0   84
  bits 0x0, link count 5
Router   10.0.31.240      10.0.31.240      0x8000044a  1400  0x22 0x3aa1  84
  bits 0x0, link count 5
Router  *10.0.31.241      10.0.31.241      0x80000446  1287  0x22 0x6c81  84
  bits 0x0, link count 5
```

JSNAPy Output

```
Tests Included: OSPF-DB
********************* Command: show ospf database detail *********************
Router 10.0.27.254 has 8 links
FAIL | All "link-count" is in range:  "6.000000 - 10.000000" [ 3 matched / 1 failed ]
------------------------------- Final Result!! -------------------------------
Total No of tests passed: 0
Total No of tests failed: 1
Overall Tests failed!!!
```

Use case: Note that `not-range` is best suited for the condition where an absolute value is not compulsory but the result is in a range the test deems passable. It is good for active routes or for routes learned from a protocol.

Name: is-in

Description: Checks to see if the specified element string value is included in a given list of strings.

Example: `check_ospf_neighbor_state.yml`

```
ospf-neighbor:
- command: show ospf neighbor
- iterate:
       xpath: ospf-neighbor
       id: interface-name
       tests:
           -  is-in: ospf-neighbor-state, Up, Full
              err: 'OSPF neighbor on {{post["interface-name"]}} to {{post["neighbor-
address"]}} is neither up nor Full, rather {{post["ospf-neighbor-state"]}}'
              info: 'All OSPF neighbors are {{post["ospf-neighbor-state"]}}'
```

NOTE One of the values used in the example (Up) is not a valid state for OSPF, but this book uses OSPF to show off JSNAPy, and here the concept of the JSNAPy operator is being demonstrated.

CLI

```
premesh@lab> show ospf neighbor
Address          Interface              State     ID               Pri  Dead
10.0.31.213      ae0.0                  Full      10.0.31.240      128    32
169.254.61.61    ae1.0                  Exchange 10.0.31.232       128    37
```

JSNAPy Output

```
Tests Included: ospf-neighbor
************************ Command: show ospf neighbor ************************
OSPF neighbor on ae1.0 to 169.254.61.61 is not up, rather Exchange
FAIL | All "ospf-neighbor-state" is not in list ['Up', 'Full'] [ 1 matched / 1 failed ]
------------------------------- Final Result!! -------------------------------
Total No of tests passed: 0
Total No of tests failed: 1
Overall Tests failed!!!
```

Use case: The best use case for `is-in` is auditing the configuration for whether the defined information is included or not included. Examples would be checking IGP /BGP peering status or checking to see if a particular interface is configured in MPLS, LDP, or RSVP.

Name: not-in

Description: Checks if the specified element string value is *not* included in a given list of strings.

Example: `check_ospf_neighbor_state.yml`

```
ospf-neighbor:
- command: show ospf neighbor
- iterate:
     xpath: ospf-neighbor
     id: interface-name
     tests:
    - not-in: ospf-neighbor-state, Up, Full
      info: 'OSPF neighbor on {{post["interface-name"]}} to {{post["neighbor-
address"]}} is neither up nor Full, rather {{post["ospf-neighbor-state"]}}'
      err: 'All OSPF neighbors are up'
```

CLI

```
premesh@lab> show ospf neighbor
Address          Interface              State     ID               Pri  Dead
10.0.31.213      ae0.0                  Full      10.0.31.240      128    32
169.254.61.61    ae1.0                  Exchange  10.0.31.232      128    37
```

JSNAPy Output

```
Tests Included: ospf-neighbor
************************ Command: show ospf neighbor ************************
-----------------------Performing not-in Test Operation-----------------------
All OSPF neighbors are up
OSPF neighbor on ae1.0 to 169.254.61.61 is not up, rather Exchange
FAIL | "ospf-neighbor-state" is in list ['Up', 'Full'] [ 1 matched / 1 failed ]
------------------------------- Final Result!! -------------------------------
Total No of tests passed: 0
Total No of tests failed: 1
Overall Tests failed!!!
```

NOTE The output has logging enabled as a DEBUG, the idea being to print the informational message as well.

Use case: Note that `not-in` is the exact opposite of `is-in`, the best use case being to audit the configuration with a check to see if the defined information is included or not included, for example, determining whether IGP /BGP peering status is or is not full.

Name: no-diff

Description: No-diff compares the data elements present in both pre- and post-snapshots and verifies whether or not their value is the same.

```
Example:  check_ospf_neighbor_address.yml
ospf-neighbor:
- command: show ospf neighbor
- iterate:
    xpath: ospf-neighbor
    id: interface-name
    tests:
    - no-diff: neighbor-address
      err: '{{id_0}} was going to {{pre["neighbor-
address"]}}, now going to {{post["neighbor-address"]}}'
      info: 'OSPF neighbor address is same in both pre and post snapshots'
```

CLI

Before:

```
premesh@lab> show ospf neighbor
Address          Interface              State     ID               Pri  Dead
10.0.31.213      ae0.0                  Full      10.0.31.240      128    32
169.254.21.62    ae1.0                  Full      10.0.31.232      128    37
```

After:

```
10.0.31.213      ae0.0                  Full      10.0.31.240      128    32
169.254.61.61    ae1.0                  Full      10.0.31.232      128    37
```

JSNAPy Output

```
Tests Included: ospf-neighbor
************************ Command: show ospf neighbor ************************
   was going to ['169.254.21.62'], now going to ['169.254.61.61']
FAIL | All "neighbor-address" is not same in pre and post snapshot [ 1 matched / 1 failed ]
------------------------------- Final Result!! -------------------------------
Total No of tests passed: 0
Total No of tests failed: 1
Overall Tests failed!!!
```

Use case: Note that `no-diff` best suits situations when you want to see what has changed between the pre- and the post-snapshot. A use case can be comparing what IP the protocol is peering with, or to check the status of peering, such as if the peering for BPG, IGP is up or not.

Name: list-not-less

Description: Check if information is present in the first snapshot but not present in the second snapshot.

Example: `check_ospf_interface_name.yml`

```
ospf-neighbor:
- command: show ospf neighbor
- iterate:
    xpath: ospf-neighbor
    id: interface-name
    tests:
      - list-not-less:
        err: 'OSPF interface gone missing: {{pre["interface-
name"]}} going to {{pre["neighbor-address"]}}'
        info: 'All OSPF interface-name are present in both pre and post snapshot'
```

CLI

Before:

```
premesh@lab> show ospf neighbor
Address          Interface              State     ID               Pri  Dead
10.0.31.213      ae0.0                  Full      10.0.31.240      128    32
169.254.61.61    ae1.0                  Full      10.0.31.232      128    37
```

After:

```
premesh@lab> show ospf neighbor
Address          Interface              State     ID               Pri  Dead
10.0.31.213      ae0.0                  Full      10.0.31.240      128    32
```

JSNAPy Output

```
Tests Included: ospf-neighbor
************************ Command: show ospf neighbor ************************
OSPF interface list check
ID gone missing !!
ID list ' {'id_0': ['ae1.0']} ' is not present in post snapshots
OSPF interface gone missing: ae0.0 going to 10.0.31.213
FAIL | All "no node" in pre snapshot is not present in post snapshot [ 1 matched / 1 failed ]
------------------------------- Final Result!! -------------------------------
Total No of tests passed: 0
Total No of tests failed: 1
Overall Tests failed!!!
```

Use case: Use `list-not-less` to find if something is missing after an activity, such as the removal of an interface from IGP (OSPF/ISIS) or BGP, or determining if the interfaces have gone missing.

Name: list-not-more

Description: The opposite of `list-not-less`, it checks to see if an item is present in the second snapshot but not in the first snapshot.

Example: ospf-neighbor .yml

```
ospf-neighbor:
- command: show ospf neighbor
- iterate:
    id: interface-name
    xpath: ospf-neighbor
    tests:
    - list-not-more:
      err: 'New OSPF interface added: {{post["interface-name"]}} going to
            {{post["neighbor-address"]}}'
      info: 'All OSPF interface-name are present in both pre and post
            snapshot'
```

CLI

Before:

```
premesh@lab> show ospf neighbor
Address          Interface              State     ID               Pri  Dead
10.0.31.213      ae0.0                  Full      10.0.31.240      128    32
```

After:

```
premesh@lab> show ospf neighbor
Address          Interface              State     ID               Pri  Dead
10.0.31.213      ae0.0                  Full      10.0.31.240      128    32
169.254.61.61    ae1.0                  Full      10.0.31.232      128    37
```

JSNAPy Output

```
Tests Included: ospf-neighbor
************************ Command: show ospf neighbor ************************
ID gone missing!!

ID list ' {'id_0': ['ae1.0']} ' is not present in pre snapshots
New OSPF interface added: ae0.0 going to 10.0.31.213
FAIL | All "no node" in post snapshot is not present in pre snapshot [ 1 matched / 1 failed ]
------------------------------- Final Result!! -------------------------------
Total No of tests passed: 0
Total No of tests failed: 1
Overall Tests failed!!!
```

Use case: Note that `list-not-more` is very useful for finding any new additions after an activity. Examples of the best use cases are determining the addition of any core files or any additional chassis alarm logs, checking to see if cards or interfaces have been added, adjacencies, etc.

Name: delta

Description: Compare the change in value of an element to a delta.

Requirement: Must be present in both snapshots.

The delta can be specified as an absolute percentage, a positive or negative percentage, an absolute fixed value, or a positive or negative fixed value.

Example: OSPF-DB.yml

```
OSPF-DB:
- command: show ospf database detail
  - iterate:
      id: '../lsa-id'
      xpath: //ospf-database/ospf-router-lsa
      tests:
        - delta: link-count, 20%
          info: "Test Succeeded!! OSPF router LSA link for <{{post['lsa-
id']}}> has changed within delta, before it was <{{pre['link-
count']}}>, now it is <{{post['link-count']}}> "
          err: "Test Failed!!! OSPF router LSA link for <{{post['../lsa-
id']}}> has changed more than delta, before it was {{pre['link-
count']}}, now it is {{post['link-count']}} "
```

CLI

```
premesh@lab> show ospf database detail | match "Router|link count"
Router   10.0.27.254      10.0.27.254      0x800001dc   169  0x22 0xf8ff  84
  bits 0x0, link count 55 <25>
Router   10.0.31.232      10.0.31.232      0x8000042f   186  0x22 0x3987  84
  bits 0x0, link count 5
Router   10.0.31.240      10.0.31.240      0x80000431   173  0x22 0x6c88  84
  bits 0x0, link count 5
Router  *10.0.31.241      10.0.31.241      0x8000042d    60  0x22 0x9e68  84
  bits 0x0, link count 2 <5>
```

JSNAPy Output

```
********************* Command: show ospf database detail *********************
Test Failed!!! OSPF router LSA link for <10.0.27.254> has changed more than delta, before it
was 55.0, now it is 25.0
Test Failed!!! OSPF router LSA link for <10.0.31.241> has changed more than delta, before it
was 2.0, now it is 5.0
FAIL | All "link-count" is not with in delta difference of 20% [ 2 matched / 2 failed ]
------------------------------- Final Result!! -------------------------------
Total No of tests passed: 0
Total No of tests failed: 1
Overall Tests failed!!!
```

Use case: Note that `delta` is a great feature when a test does not denote failure if the change is in an acceptable range. A good use case could be the IP route, where the test is supposed to pass even if there is a little difference in the value before and after.

NOTE Let's explain some advance features, introduced in JSNAPy 1.1, such as *chained operators* and *ignore-null attribute*.

Name: chained operators

Description: As the name suggests, it provides the ability to have complex operations with the help from logical operators like AND, OR, and NOT.

Example: test_command_version.yml

```
test_command_version:
  - command: show bgp neighbor
  - iterate:
      xpath: '/bgp-information/bgp-peer'
      id: peer-address
      tests:
        - OR:
           - AND:
                 - not-equal: last-state,Active       # element in which test is performed
                   err: "Test Failed!! last state is <{{post['last-state']}}>"
               info: "Test succeeded!! last state is not equal to idle, it is: <{{post['last-
state']}}>"

              - NOT:
                - all-same: flap-count
                  err: "Test Failed!!! flap count are not all same!!, it is <{{post['flap-
count']}}> "
                  info: "Test Succeeded!! flap count are all same, it is now <{{post['flap-
count']}}>!!!"

         - is-equal: flap-count, 0
           err: "Test Failed!!! flap count is not equal to 0, it is: <{{post['flap-count']}}>
"
           info: "Test Succeeded!! flap count is equal to <{{post['flap-count']}}> !!" it was
{{pre['link-count']}}, now it is {{post['link-count']}} "
```

In this example OR, AND, and NOT are used.

- For OR: IF one condition is Pass, Then Pass, Else, Fail
- For AND: All conditions should be Pass, Then Pass, Else, Fail
- For NOT: If condition Pass, Then Fail, Else, Pass

In the above example OR (AND (not-equal && NOT(all-same)) is-equal).

JSNAPy Output

```
Connecting to device 10.13.10.10 ................
Taking snapshot of COMMAND: show task replication
**************************** Device: 10.13.10.10 ****************************
Tests Included: task-replication
************************ Command: show task replication ************************
SKIPPING!! Node <task-protocol-replication-state> not found at xpath <.> for IDs: {'name':
None}
-------------------------------- Final Result!! --------------------------------
task-replication : Skipped
Total No of tests passed: 0
Total No of tests failed: 0
None of the test cases executed !!!
```

Use case: Chained operators are important and necessary to build complex test cases with the use of logical operators. There are many conditions where execution of one test is dependent on the result of another test, such as testing the number of times a link flaps if BGP is established and LogUpDown is enabled, or, checking the number of routes received if BGP is up.

Name: ignore-null (Optional Attribute)

Description: The `ignore-null` attribute can be used with any operator for passing the test in the absence of nodes for which the XPath is provided. One thing to note is that instead of classifying this as error, in the absence of nodes the program would just skip the test, not pass it, just skip it. This is similar to the old behavior in JSNAP.

Example: task-replication.yml

```
tests_include:
  - task-replication

task-replication:
- command: show task replication
- iterate:
    id: name
    xpath: .
    tests:
      - is-equal: task-protocol-replication-state, Complete
        ignore-null: True
        err: 'Task replication is pending, please look at show task replication output'
        info: 'Task replication successful.'
```

JSNAPy Output

```
Connecting to device 10.10.10.104 ................
Taking snapshot of COMMAND: show task replication
*************************** Device: 10.10.10.104 ***************************
Tests Included: task-replication
*********************** Command: show task replication ***********************
SKIPPING!! Node <task-protocol-replication-
state> not found at xpath <.> for IDs: {'name': None}
------------------------------- Final Result!! -------------------------------
task-replication : Skipped
Total No of tests passed: 0
Total No of tests failed: 0
None of the test cases executed !!!
```

NOTE Tested on a router that did not have task replication enabled.

Use case: This optional attribute changes the behavior of failing the test if the node is absent in XPath (under the following conditions):

- One must have expected some nodes to be there, and if not found, the program notifies the user about the condition.

- No operations can be performed on a missing node, so skip the test altogether.

Summary

There’s a lot of information in this chapter so digest it in the best way you can, and let’s move on to examine a working example I have tested and implemented in many large production networks.

Chapter 4

Working Examples

Okay, now it's time to see JSNAPy perform its magic. This chapter lets you see what JSNAPy can do using two commonly used scenarios in the networking world. The first shows a string comparison and the second uses a Method of Procedure (MoP) that the author has implemented in many major networks.

JSNAPy String Comparison

You should now know that JSNAPy has a feature that can compare two snapshots, node by node, using the `--diff` command, without specifying any test case. Let's identify the key differences for the `show configuration` command between pre- and post-snapshots.

By now, you should be familiar with the procedure: save the show configuration with the display set in two different files (pre and post) and use JSNAPy `--diff` with the file name to identify the differences between configurations. The differences are highlighted in `bold` in the following output.

NOTE The difference in the config is `deactivate routing-instances vrf`.

```
premesh@automation$ jsnapy --diff snapshots/config_100.100.100.1_set_pre  snapshots/
config_100.100.100.1_set_post
snapshots/config_100.100.100.1_set_pre
snapshots/config_100.100.100.1_set_post
---
661 set routing-instances vrf instance-type vrf
660 set routing-instances vrf instance-type vrf
   662 set routing-instances vrf interface irb.0                           661 set routing-
instances vrf interface irb.0
   663 set routing-instances vrf route-distinguisher 100:12               662 set routing-
instances vrf route-distinguisher 100:12
   664 set routing-instances vrf vrf-target target:100:4                  663 set routing-
instances vrf vrf-target target:100:4
   665 set routing-instances vrf vrf-table-label                          664 set routing-
instances vrf vrf-table-label
                                                             665 deactivate routing-instances vrf
automation@automation: $
```

The difference here is just a string comparison.

JSNAPy MoP

This is a MoP developed to help the author with verifications while performing Junos OS upgrades for customers. Table 4.1 lists the procedure and is followed by the JSNAPy configuration.

The MoP is divided into three parts:

1. Collect the data during pre- and post-activity and then compare later to detect any differences.

2. Collect the data during pre-activity and check for any discrepancies such as chassis alarm, core file, etc.

3 Collect the data during post-activity and check for any discrepancies such as chassis alarm, core file, etc. In addition, compare the pre- and post-activity.

Table 4.1 Method of Procedure (MoP)

Category	CLI	Check	Output
Hardware status	`show chassis hardware models`	Check serial numbers of all hardware.	Notify previous and new serial numbers for inventory purpose.
	`show chassis fpc pic-status`	Check status of all hardware (FPC and PIC).	Notify of FPCs or PICs that are not online.
	`show chassis environment`	Check status of all hardware environments.	Notify if any hardware status has changed.
	`show chassis routing-engine`	Check mastership status of both REs.	Notify if any RE slot mastership status has changed.
	`show version invoke-on all-routing-engines`	Check the Junos version on both REs.	Notify if any of the REs has different Junos version pre- and post-activity.
System Status	`show chassis alarms`	Check if any new alarms are active.	Notify if any new alarms are active.
	`show system core-dumps routing-engine both`	Check if there are any core files on any of the REs.	Notify if there are any new core files.
Interfaces Status	`show arp no-resolve`	Check if any change in the ARP, newly added or previously MAC deleted. Also check if there is any interface or IP change for any MAC-address.	Notify if any new MAC was added or earlier MAC deleted. Also when any interface or IP change for any MAC-address occurred.
	`show interface terse`	Check if any oper-status has changed.	Notify if nterface, oper-status, or admin-status has changed.
	`show interfaces extensive`	NO check.	Display admin-status, oper-status , input-bytes, and output-bytes for logical and physical interfaces.
Route Summary	`show route summary`	Check if any route table has discrepancy in number of routes.	Notify if number of routes has changed by 20%.
	`show route forwarding-table summary`	Check if any forwarding table has discrepancy in number of routes.	Notify if number of route in forwarding table has changed by 20%.
BFD	`show bfd session detail`	Check if any BFD session is down.	Notify if BFD session is down.

Category	CLI	Check	Output
OSPF	`show ospf interface`	Check if any OSPF interface has 0 neighbor (interface which contains "-").	Notify if any OSPF interface has 0 neighbor.
	`show ospf neighbor`	Check if any OSPF neighbor is missing.	Notify if any OSPF neighbor is missing.
	`show ospf database detail`	Database detail. Check if any OSPF router link count is in predefined range.	Notify if any OSPF router link count is out of range.
BGP	`show bgp summary`	Check if any BGP neighbor is missing. Plus if all established plus active prefix delta is more than 20% .	Notify if any BGP neighbor is missing. Or it is not established or delta is more than 20%.
LDP	`show ldp interface`	Check if any LDP interface has 0 neighbor .	Notify if any LDP interface has 0 neighbor.
	`show ldp neighbor`	Check if any LDP neighbor is missing.	Notify if any LDP neighbor is missing.
RSVP	`show rsvp interface`	Check if any RSVP interface is missing. Plus if any interface is not in UP state.	Notify if any RSVP interface is missing.
	`show rsvp session`	Check if any RSVP session is missing.	Notify if any RSVP session is missing.
MPLS	`show mpls lsp detail`	Check if any MPLS LSP is not in UP state, also that no LSP is missing.	Notify if any MPLS LSP session is not up.
Layer 2 Circuits	`show l2circuit connections`	Check if any l2circuit is not in UP state, also that no l2circuit is missing.	Notify if any l2circuit session is not up.

config_mop.yml

```
# for one device, can be given like this:
hosts:
   - device: 100.100.100.1
     username : lab
     passwd: lab123
tests:
  - chassis_hardware.yml
  - pic_status.yml
  - interface_statistic.yml
  - chassis_environment.yml
  - junos_routing-engine.yml
  - junos_version.yml
  - test_alarm_check.yml
  - check_core.yml
  - test_arp.yml
  - test_interface_operstate.yml
  - test_route_summary.yml
  - test_forwarding_summary.yml
  - test_bfd_session.yml
  - ospf-neighbor.yml
  - test_ospf_interface.yml
  - test_bgp-summary.yml
  - test_ospf_db.yml
  - test_rsvp.yml
  - test_rsvp_interface.yml
  - test_ldp_interface.yml
  - ldp-neighbor.yml
  - test_mpls_lsp.yml
  - l2circuit.yml


# can use sqlite to store data and compare them
#sqlite:
#  - store_in_sqlite: True
#    check_from_sqlite: True
#    database_name: jbb.db
#    compare: 1,0

# can send mail by specifying mail
 mail: send_mail.yml
```

chassis_hardware.yml

```
tests_include:
  - chassis_hardware

chassis_hardware:
  - command: show chassis hardware
  - iterate:
      xpath: '//chassis/chassis-module'
      id: './name'
      tests:
        - no-diff: serial-number
          err: "Test Failed!! serial-number got changed, before it was <{{pre['serial-
number']}}>, now it is <{{post['serial-number']}}> with name <{{id_0}}>"
          info: "Test Succeeded!! serial-number are all same for name <{{id_0}}>!!!"
  - iterate:
      xpath: '//chassis/chassis-module/chassis-sub-module'
      id: './name'
      tests:
        - no-diff: serial-number
          err: "Test Failed!! serial-number got changed, before it was <{{pre['serial-
number']}}>, now it is <{{post['serial-number']}}> with name <{{id_0}}>"
          info: "Test Succeeded!! serial-number are all same for name <{{pre['../name']}}>
<{{id_0}}>!!!"
```

pic_status.yml

```
tests_include:
  - check_chassis_fpc

check_chassis_fpc:
  - command: show chassis fpc pic-status
  - iterate:
      xpath: //fpc
      id: ./slot
      tests:
        - all-same: state
          err: "Test Failed!!! state of \"{{post['description']}}\" at fpc {{post['slot']}}
are not all same!!, it is <{{post['state']}}> "
          info: "Test Succeeded!! state of \"{{post['description']}}\" at fpc {{post['slot']}}
are all same, it is now <{{post['state']}}>!!!"
  - iterate:
      xpath: //fpc/pic
      id: ./pic-slot
      tests:
        - all-same: pic-state
          err: "Test Failed!!! state of \"{{post['pic-type']}}\" at fpc {{post['../slot']}}
pic {{post['pic-slot']}} are not all same!!, it is <{{post['pic-state']}}> "
          info: "Test Succeeded!! state of \"{{post['pic-type']}}\" at fpc {{post['../slot']}}
pic {{post['pic-slot']}} are all same, it is now <{{post['pic-state']}}>!!!"
```

interface_statistic.yml

```
tests_include:
  - check_show_interfaces

check_show_interfaces:
  - command: show interfaces extensive
  - iterate:
      xpath: //physical-interface[contains(name, "-")]
      tests:
        - is-in: oper-status, up
          err: "{{post['name']}} \t {{post['oper-status']}} \t {{post['admin-status']}} \t
{{post['traffic-statistics/input-bps']}} \t {{post['traffic-statistics/output-bps']}}"
          info: "Test Failed!!! Physical operational status is not up, inetrface
<{{post['logical-interface/name']}}> is: <{{post['oper-status']}}> with admin status
<{{post['admin-status']}}> "
  - iterate:
      xpath: //physical-interface/logical-interface[contains(name, "-")]
      tests:
        - is-in: snmp-index, up
          err: "{{post['name']}} \t {{post['../oper-status']}}  \t {{post['../admin-status']}}
```

```
\t {{post['transit-traffic-statistics/input-bps']}} \t {{post['transit-traffic-statistics/
output-bps']}}"
         info: "Test Failed!!! logical operational status is not up, inetrface
<{{post['name']}}> is: <{{post['oper-status']}}> with admin status <{{post['admin-
status']}}> "
```

chassis_environment.yml

```
tests_include:
  - check_environment

check_environment:
  - command: show chassis environment
  - iterate:
      xpath: /environment-information/environment-item
      id: ./name
      tests:
        - no-diff: status
          err: "Test Failed!!! states of \"{{pre['name']}}\" are not all same!!, earlier it was
<{{pre['status']}}> and now its <{{post['status']}}>"
          info: "Test Succeeded!! states of \"{{post['name']}}\"  are all same, it is now
<{{post['status']}}>!!!"
```

junos_routing-engine.yml

```
tests_include:
  - test_routing-engine

test_routing-engine:
  - command: show chassis routing-engine
  - iterate:
      xpath: //route-engine-information/route-engine
      id: 'slot'
      tests:
        - no-diff: mastership-state
          info: "Test Succeeded!! RE SLOT {{post['slot']}} has mastership-state
\"{{post['mastership-state']}}\" "
         err: "Test Failed!!! Junos version does not match earlier on RE SLOT {{post['slot']}}
it was \"{{pre['mastership-state']}}\" and now it is \"{{post['mastership-state']}}\""
```

junos_version.yml

```
tests_include:
  - test_version_check

test_version_check:
  - command: show version invoke-on all-routing-engines
  - iterate:
      xpath: //software-information
      id: 'junos-version'
      tests:
        - no-diff: junos-version
          info: "Test Succeeded!! {{post['host-name']}} has Junos version  \"{{post['junos-
version']}}\" "
          err: "Test Failed!!! Junos version does not match earlier it was \"{{pre['junos-
version']}}\" and now it is \"{{post['junos-version']}}\""
```

test_alarm_check.yml

```
tests_include:
  - alarm-checks
alarm-checks:
  - command: show chassis alarms
  - iterate:
      xpath: '/alarm-information'
      tests:
          - list-not-more: './alarm-detail/alarm-description'
            info: "Test Succeeded!! chassis alarms <{{pre['alarm-description']}}> already 
exist"
           err: "Test Failed!!!There is new chassis alarms <{{post['alarm-detail/alarm-
description']}}> "
```

check_core.yml

```
tests_include:
  - check_core
check_core:
  - command: show system core-dumps routing-engine both
  - iterate:
      xpath: '//multi-routing-engine-results/multi-routing-engine-item/directory-list'
      tests:
          - not-exists: ./directory/file-information/file-name
            err: "Test Failed!!core file exist <{{post['./directory/file-information/file-
name']}}> on {{post['./../re-name']}}"
            info: "Test Succeeded!!!There is no core file"
```

test_arp.yml

```
tests_include:
  - test_arp_mac
  - test_arp_mac2

test_arp_mac:
  - command: show arp no-resolve
  - iterate:
      xpath: '//arp-table-information/arp-table-entry'
      id: './mac-address'
      tests:
        - no-diff: ip-address      # element in which test is performed
          err: "Test Failed!! ip-address got changed for mac-address <{{id_0}}>, before it was 
<{{pre['ip-address']}}>, now it is <{{post['ip-address']}}>"
          info: "ip-address is same for mac  <{{id_0}}>"

        - no-diff: interface-name      # element in which test is performed
          err: "Test Failed!! interface-name got changed for mac-address <{{id_0}}>, before it 
was <{{pre['interface-name']}}>, now it is <{{post['interface-name']}}>"
          info: "interface-name is same for mac <{{id_0}}>"

test_arp_mac2:
  - command: show arp no-resolve
  - iterate:
      xpath: '//arp-table-information/arp-table-entry'
      id: './interface-name'
      tests:
```

```
      - list-not-less: mac-address
        err: "name list changed, mac-address: <{{pre['mac-address']}}> with interface-name 
<{{id_0}}> is not present in post-snap"
        info: "name list is same, mac-address is same for  <{{id_0}}>"

      - list-not-more: mac-address
        err: "<{{post['mac-address']}}> mac-address with interface-name <{{id_0}}> is not 
present in pre snapshot"
        info: "mac-address is same for  <{{id_0}}>"
```

test_interface_operstate.yml

```
tests_include:
  - test_interface_operstate

# use '/' in your test cases apart from xpath if u want to search all elements irrespective of 
hierarchy, ex: in id if u use /name instead of name
# then it will search in all the names in given xpath irrespective of their position
# for simple, one test using command

test_interface_operstate:
  - command: show interfaces terse
  - iterate:
     xpath: physical-interface[contains(name, "-")]
     id: './name'
     tests:
      - no-diff: oper-status      # element in which test is performed
        err: "Test Failed!! oper-status  got changed for <{{post['name']}}>, before it was 
<{{pre['oper-status']}}>, now it is <{{post['oper-status']}}> with name <{{id_0}}> and admin 
status <{{post['admin-status']}}>"
       info: "Test succeeded!! oper-status is same for <{{post['name']}}> with value, before 
it is <{{pre['oper-status']}}> now it is <{{post['oper-status']}}> with admin status 
<{{post['admin-status']}}> "
```

test_route_summary.yml

```
tests_include:
  - check_route_summary

check_route_summary:
  - command: show route summary
  - iterate:
     xpath: //route-summary-information/route-table
     id: ./table-name
     tests:
      - delta: active-route-count, 20%
        info: "Test Succeeded!! Active route for routing table <{{post['table-name']}}> has 
changed within delta 20%, before it was <{{pre['active-route-count']}}>, now it is 
<{{post['active-route-count']}}> "
        err: "Test Failed!!! Active route for routing table <{{post['table-name']}}> has 
changed more than delta 20%, before it was <{{pre['active-route-count']}}>, now it is 
<{{post['active-route-count']}}> "
```

test_forwarding_summary.yml

```
tests_include:
  - check_forwarding_summary

check_forwarding_summary:
  - command: show route forwarding-table summary
  - iterate:
     xpath: //forwarding-table-information/route-table/route-table-summary
     id: ./../table-name
     tests:
       - delta: route-count, 20%
         info: "Test Succeeded!! Active route for forwarding table <{{post['../table-name']}}> 
and type <{{post['route-table-type']}}> has changed within delta 20%, before it was 
<{{pre['route-count']}}>, now it is <{{post['route-count']}}> "
         err: "Test Failed!!! Active route for forwarding table <{{post['../table-name']}}> 
and type <{{post['route-table-type']}}> has changed more than delta 20%, before it was 
<{{pre['route-count']}}>, now it is <{{post['route-count']}}> "
```

test_bfd_session.yml

```
tests_include:
  - test_bfd_session

test_bfd_session:
  - command: show bfd session extensive
  - iterate:
     xpath: '/bfd-session-information/bfd-session'
     tests:
       - no-diff: session-state      # element in which test is performed
         err: "Test Failed!! state is <{{post['session-state']}}> now but earlier was 
<{{pre['session-state']}}> for neighbor <{{post['session-neighbor']}}> and client 
<{{post['bfd-client/client-name']}}>"
         info: "Test succeeded!! state has not changed for neighbor <{{post['session-
neighbor']}}> and client <{{post['bfd-client/client-name']}}>"
```

ospf-neighbor.yml

```
tests_include:
  - ospf-neighbor

ospf-neighbor:
- command: show ospf neighbor
- iterate:
    id: interface-name
    tests:
     - err: 'OSPF interface gone missing: {{pre["interface-name"]}} going to {{pre["neighbor-
address"]}}'
       info: OSPF interface list check
       list-not-less:
     - err: 'New OSPF interface added: {{post["interface-name"]}} going to {{post["neighbor-
address"]}}'
       info: OSPF interface list check
       list-not-more:
     - err: '   was going to {{pre["neighbor-address"]}}, now going to {{post["neighbor-
address"]}}'
       info: OSPF neighbor change check
```

```
      no-diff: neighbor-address
    - err: OSPF neighbor on {{post["interface-name"]}} to {{post["neighbor-address"]}} is not
up, rather {{post["ospf-neighbor-state"]}}
      info: All OSPF neighbors are up
      is-equal: ospf-neighbor-state, Full
    - err: 'All OSPF neighbors are up'
      info: 'OSPF neighbor on {{post["interface-name"]}} to {{post["neighbor-address"]}} is
not up, rather {{post["ospf-neighbor-state"]}}'
      not-equal: ospf-neighbor-state, Full
    - err: 'OSPF neighbor on {{post["interface-name"]}} to {{post["neighbor-address"]}} state
change from  {{pre["ospf-neighbor-state"]}} to {{post["ospf-neighbor-state"]}}'
      info: 'No state change for OSPF neighbors'
      all-same: ospf-neighbor-state
    - err: 'OSPF neighbor on {{post["interface-name"]}} to {{post["neighbor-address"]}} is
not up, rather {{post["ospf-neighbor-state"]}}'
      info: 'All OSPF neighbors are up'
      is-in: ospf-neighbor-state, Up, Full
    - info: 'OSPF neighbor on {{post["interface-name"]}} to {{post["neighbor-address"]}} is
not up, rather {{post["ospf-neighbor-state"]}}'
      err: 'All OSPF neighbors are up'
      not-in: ospf-neighbor-state, Up, Full
   xpath: ospf-neighbor
```

test_ospf_interface.yml

```
tests_include:
  - ospf_interface
ospf_interface:
- command: show ospf interface
- iterate:
   tests:
   - err: OSPF interface {{post["interface-name"]}} does not have any neighbors
     info: OSPF interfaces must have at least 1 neighbor
     is-gt: neighbor-count, 0
   - err: 'OSPF interfaces must have at least 1 neighbor, {{post["interface-name"]}} has
atleast 1 neighbor'
     info: 'OSPF interface {{post["interface-name"]}} does not have any neighbors'
     is-lt: neighbor-count, 1
   xpath: ospf-interface[interface-name != "lo0.0"]
```

test_bgp-summary.yml

```
tests_include:
  - bgp-summary
bgp-summary:
- command: show bgp summary
- iterate:
   id: peer-address
   tests:
   - err: 'BGP peer AS: {{post["peer-as"]}}, NEI: {{post["peer-address"]}} is not Estab,
rather {{post["peer-state"]}}'
     info: All BGP Peers are 'Established'
     is-equal: peer-state, Established
   - err: 'BGP RIB: ''{{post["peer-address"]}}'' went away, oh no!'
     info: BGP list did not loose peers
     list-not-less: null
```

```
    - err: 'BGP RIB: ''{{post["peer-address"]}}'' is added !'
      info: BGP list add new peers
      list-not-more: null
    xpath: bgp-peer
- iterate:
    id: './bgp-rib/name'
    tests:
    - delta: .//bgp-rib/active-prefix-count, 20%
      err: ' ERROR: The number of active prefix of the BGP Table {{post["bgp-rib/name"]}}have 
changed more than 20%. [Before = {{pre["bgp-rib/active-prefix-count"]}} / After = 
{{post["bgp-rib/active-prefix-count"]}}]'
      info: 'Checking BGP peer active prefix count (tolerance 20%) {{post["bgp-rib/active-
prefix-count"]}}'
xpath: '/bgp-information'
```

test_ospf_db.yml

```
tests_include:
           - OSPF-DB
OSPF-DB:
- command: show ospf database detail
- iterate:
    id: ../advertising-router
    tests:
    - err: Router {{post["../advertising-router"]}} has {{post["link-count"]}} links
      in-range: link-count, 5, 10
      info: OSPF router links [5..10]
    - err: Router {{post["../advertising-router"]}} has {{post["link-count"]}} links
      info: OSPF router links not[5..10]
      not-range: link-count, 5, 10
    - err: 'Router {{post["../advertising-router"]}} has changed to {{post["link-count"]}} 
links earlier it was {{pre["link-count"]}} '
      info: 'OSPF router links not changed significantly for {{post["../advertising-
router"]}}'
      delta: link-count, 20%
    xpath: '/ospf-database-information/ospf-database/ospf-router-lsa'
```

test_rsvp.yml

```
tests_include:
- rsvp
rsvp:
- command: show rsvp session
- iterate:
    tests:
    - err: ' RSVP session with name {{post["name"]}} to {{post["destination-address"]}}
        has LSP state {{post["lsp-state"]}}.'
      info: RSVP LSP state is [Up | NotInService]
      is-in: lsp-state, Up, NotInService
    xpath: rsvp-session-data/rsvp-session
```

test_rsvp_interface.yml

```
tests_include:
  - rsvp-interface

rsvp-interface:
- command: show rsvp interface
- iterate:
    id: interface-name
    tests:
    - err: 'RSVP interface gone missing: {{pre["interface-name"]}}'
      info: RSVP interface list check
      list-not-less:
    - err: RSVP neighbor on {{post["interface-name"]}} is not up, rather {{post["rsvp-
status"]}}
      info: All RSVP neighbors are up
      is-equal: rsvp-status, Up
    xpath: rsvp-interface
```

test_ldp_interface.yml

```
tests_include:
  - ldp_interface

ldp_interface:
- command: show ldp interface
- iterate:
    tests:
    - err: ldp interface {{post["interface-name"]}} does not have any neighbors
      info: ldp interfaces must have at least 1 neighbor
      is-gt: ldp-neighbor-count, 0
    xpath: ldp-interface
```

ldp-neighbor.yml

```
tests_include:
  - ldp-neighbor

ldp-neighbor:
- command: show ldp neighbor
- iterate:
    id: interface-name
    tests:
    - err: 'ldp interface gone missing: {{pre["interface-name"]}} going to {{pre["ldp-
neighbor-address"]}}'
      info: ldp interface list check
      list-not-less:
    - err: '   was going to {{pre["neighbor-address"]}}, now going to {{post["ldp-neighbor-
address"]}}'
      info: ldp neighbor change check
      no-diff: neighbor-address
    xpath: ldp-neighbor
```

test_mpls_lsp.yml

```
tests_include:
- mpls_lsp
mpls_lsp:
- command: show mpls lsp detail
- iterate:
    tests:
    - err: ' MPLS LSP  with name {{post["name"]}} to {{post["destination-address"]}}
        has LSP state {{post["lsp-state"]}}.'
```

```
     info: MPLS LSP state is Up
     is-in: lsp-state, Up
   - err: 'MPLS LSP gone missing: with name {{post["name"]}} to {{post["destination-
address"]}}'
     info: MPLS LSP list check
     list-not-less:
   xpath: rsvp-session-data/rsvp-session/mpls-lsp
```

l2circuit.yml

```
tests_include:
  - l2circuit

l2circuit:
- command: show l2circuit connections
- iterate:
   id: connection-id
   tests:
   - err: 'l2circuit interface gone missing: {{pre["connection-id"]}}'
     info: l2circuit interface list check
     list-not-less:
   - err: l2circuit neighbor on {{post["connection-id"]}} is not up, rather
{{post["connection-status"]}}
     info: All l2circuit neighbors are up
     is-equal: connection-status, Up
   xpath: l2circuit-neighbor/connection
```

Chapter 5

Working Towards Automation With JSNAPy

Hopefully this book has shown you the benefits of JSNAPy and you now know that JSNAPy is a great tool for data collection and verification.

But can data collection and verification work as a standalone tool? Yes, it can, depending on the size of your network, but how can JSNAPy become part of your existing automation system?

This final chapter showcases how JSNAPy can be integrated with existing automation or even become the first step towards a large network automation plan.

Embedding JSNAPy as a Python Module

A Python module is created and installed as soon as you install JSNAPy in your server. This module works exactly the same way standalone JSNAPy works, but with additional features such as calling the JSNAPy as a runtime or directly supplying the JSNAPy output to Python.

In this example, snapcheck is used through the Python module and stores the results as values that can be used as variables for other checks, or you can just print the output in a user-defined format:

```
premesh:testfiles premesh$ more module_snapcheck.py
### performing function similar to --snapcheck option in command line ######
from jnpr.jsnapy import SnapAdmin
from pprint import pprint
from jnpr.junos import Device
```

```
js = SnapAdmin()

config_file = "/etc/jsnapy/testfiles/config_single_snapcheck.yml"
snapvalue = js.snapcheck(config_file, "snap")

for snapcheck in snapvalue:
    print "\n -----------snapcheck----------"
    print "Tested on", snapcheck.device
    print "Final result: ", snapcheck.result
    print "Total passed: ", snapcheck.no_passed
    print "Total failed:", snapcheck.no_failed
    pprint(dict(snapcheck.test_details))
```

Where is the module stored by default? Note that the answer to this question maybe different if a specific path is given during installation, but the default is: `~/JSNAPy/JSNAPy/lib/jnpr`.

```
premesh:testfiles premesh$ python module_snapcheck.py
Connecting to device 10.10.10.35 ...............
Taking snapshot of COMMAND: show version
Taking snapshot of COMMAND: show chassis fpc
**************************** Device: 10.10.10.35 ****************************
Tests Included: test_version_check
*************************** Command: show version ***************************
PASS | All "//package-information/name" exists at xpath "//software-
information" [ 39 matched ]
**************************** Device: 10.10.10.35 ****************************
Tests Included: check_chassis_fpc
************************* Command: show chassis fpc *************************
PASS | All "cpu-total" is greater than 2" [ 1 matched ]
------------------------------- Final Result!! -------------------------------
Total No of tests passed: 2
Total No of tests failed: 0
Overall Tests passed!!!

 -----------snapcheck----------
Tested on 10.10.10.35
Final result:  Passed
Total passed:  2
Total failed: 0
{'show chassis fpc': [{'count': {'fail': 0, 'pass': 1},
                       'expected_node_value': 2.0,
                       'failed': [],
                       'node_name': 'cpu-total',
                       'passed': [{'actual_node_value': '11',
                                   'id': {'./memory-dram-size': '2048'},
                                   'post': {'cpu-total': '11',
                                            'memory-heap-utilization': '43'},
                                   'pre': {'cpu-total': '11'}}],
                       'result': True,
                       'testoperation': 'is-gt',
                       'xpath': '//fpc[normalize-space(slot) = "0"]'}],
 'show version': [{'count': {'fail': 0, 'pass': 39},
                   'failed': [],
                   'node_name': '//package-information/name',
                   'passed': [{'actual_node_value': 'os-kernel',
                               'id': {'host-name': 'WF-CSIM-MX960-1-re0'},
                               'post': {'//package-information/name': 'os-kernel'},
                               'pre': {'//package-information/name': 'os-kernel'}},
                              {'actual_node_value': 'os-libs',
                               'id': {'host-name': 'WF-CSIM-MX960-1-re0'},
                               'post': {'//package-information/name': 'os-libs'},
                               'pre': {'//package-information/name': 'os-libs'}},
                              {'actual_node_value': 'os-runtime',
                               'id': {'host-name': 'WF-CSIM-MX960-1-re0'},
                               'post': {'//package-information/name': 'os-runtime'},
                               'pre': {'//package-information/name': 'os-runtime'}},
                              {'actual_node_value': 'zoneinfo',
                               'id': {'host-name': 'WF-CSIM-MX960-1-re0'},
                               'post': {'//package-information/name': 'zoneinfo'},
                               'pre': {'//package-information/name': 'zoneinfo'}},
                              {'actual_node_value': 'os-libs-compat32',
                               'id': {'host-name': 'WF-CSIM-MX960-1-re0'},
                               'post': {'//package-information/name': 'os-libs-compat32'},
                               'pre': {'//package-information/name': 'os-libs-compat32'}},
                              {'actual_node_value': 'os-compat32',
                               'id': {'host-name': 'WF-CSIM-MX960-1-re0'},
                               'post': {'//package-information/name': 'os-compat32'},
                               'pre': {'//package-information/name': 'os-compat32'}},
                              {'actual_node_value': 'py-base',
                               'id': {'host-name': 'WF-CSIM-MX960-1-re0'},
                               'post': {'//package-information/name': 'py-base'},
                               'pre': {'//package-information/name': 'py-base'}},
                              {'actual_node_value': 'os-crypto',
                               'id': {'host-name': 'WF-CSIM-MX960-1-re0'},
                               'post': {'//package-information/name': 'os-crypto'},
                               'pre': {'//package-information/name': 'os-crypto'}},
                              {'actual_node_value': 'netstack',
                               'id': {'host-name': 'WF-CSIM-MX960-1-re0'},
                               'post': {'//package-information/name': 'netstack'},
                               'pre': {'//package-information/name': 'netstack'}},
                              {'actual_node_value': 'junos-libs-compat32',
                               'id': {'host-name': 'WF-CSIM-MX960-1-re0'},
                               'post': {'//package-information/name': 'junos-libs-
compat32'},
                               'pre': {'//package-information/name': 'junos-libs-
compat32'}},
                              {'actual_node_value': 'junos-runtime',
                               'id': {'host-name': 'WF-CSIM-MX960-1-re0'},
                               'post': {'//package-information/name': 'junos-runtime'},
                               'pre': {'//package-information/name': 'junos-runtime'}},
                              {'actual_node_value': 'junos-platform',
                               'id': {'host-name': 'WF-CSIM-MX960-1-re0'},
                               'post': {'//package-information/name': 'junos-platform'},
                               'pre': {'//package-information/name': 'junos-platform'}},
                              {'actual_node_value': 'junos-modules',
                               'id': {'host-name': 'WF-CSIM-MX960-1-re0'},
                               'post': {'//package-information/name': 'junos-modules'},
                               'pre': {'//package-information/name': 'junos-modules'}},
                              {'actual_node_value': 'junos-libs',
                               'id': {'host-name': 'WF-CSIM-MX960-1-re0'},
                               'post': {'//package-information/name': 'junos-libs'},
                               'pre': {'//package-information/name': 'junos-libs'}},
                              {'actual_node_value': 'junos-dp-crypto-support-platform',
                               'id': {'host-name': 'WF-CSIM-MX960-1-re0'},
                               'post': {'//package-information/name': 'junos-dp-crypto-
support-platform'},
                               'pre': {'//package-information/name': 'junos-dp-crypto-
support-platform'}},
{'actual_node_value': 'junos-daemons',
 'id': {'host-name': 'WF-CSIM-MX960-1-re0'},
 'post': {'//package-information/name': 'junos-daemons'},
 'pre': {'//package-information/name': 'junos-daemons'}},
{'actual_node_value': 'jservices-voice',
 'id': {'host-name': 'WF-CSIM-MX960-1-re0'},
 'post': {'//package-information/name': 'jservices-voice'},
 'pre': {'//package-information/name': 'jservices-voice'}},
{'actual_node_value': 'jservices-ssl',
 'id': {'host-name': 'WF-CSIM-MX960-1-re0'},
 'post': {'//package-information/name': 'jservices-ssl'},
 'pre': {'//package-information/name': 'jservices-ssl'}},
{'actual_node_value': 'jservices-sfw',
 'id': {'host-name': 'WF-CSIM-MX960-1-re0'},
 'post': {'//package-information/name': 'jservices-sfw'},
 'pre': {'//package-information/name': 'jservices-sfw'}},
{'actual_node_value': 'jservices-rpm',
 'id': {'host-name': 'WF-CSIM-MX960-1-re0'},
 'post': {'//package-information/name': 'jservices-rpm'},
 'pre': {'//package-information/name': 'jservices-rpm'}},
{'actual_node_value': 'jservices-ptsp',
 'id': {'host-name': 'WF-CSIM-MX960-1-re0'},
 'post': {'//package-information/name': 'jservices-ptsp'},
 'pre': {'//package-information/name': 'jservices-ptsp'}},
{'actual_node_value': 'jservices-nat',
 'id': {'host-name': 'WF-CSIM-MX960-1-re0'},
 'post': {'//package-information/name': 'jservices-nat'},
 'pre': {'//package-information/name': 'jservices-nat'}},
{'actual_node_value': 'jservices-mss',
 'id': {'host-name': 'WF-CSIM-MX960-1-re0'},
 'post': {'//package-information/name': 'jservices-mss'},
 'pre': {'//package-information/name': 'jservices-mss'}},
{'actual_node_value': 'jservices-mobile',
 'id': {'host-name': 'WF-CSIM-MX960-1-re0'},
 'post': {'//package-information/name': 'jservices-mobile'},
 'pre': {'//package-information/name': 'jservices-mobile'}},
{'actual_node_value': 'jservices-llpdf',
 'id': {'host-name': 'WF-CSIM-MX960-1-re0'},
 'post': {'//package-information/name': 'jservices-llpdf'},
 'pre': {'//package-information/name': 'jservices-llpdf'}},
{'actual_node_value': 'jservices-jflow',
 'id': {'host-name': 'WF-CSIM-MX960-1-re0'},
 'post': {'//package-information/name': 'jservices-jflow'},
 'pre': {'//package-information/name': 'jservices-jflow'}},
{'actual_node_value': 'jservices-ipsec',
 'id': {'host-name': 'WF-CSIM-MX960-1-re0'},
 'post': {'//package-information/name': 'jservices-ipsec'},
 'pre': {'//package-information/name': 'jservices-ipsec'}},
{'actual_node_value': 'jservices-idp',
 'id': {'host-name': 'WF-CSIM-MX960-1-re0'},
 'post': {'//package-information/name': 'jservices-idp'},
 'pre': {'//package-information/name': 'jservices-idp'}},
{'actual_node_value': 'jservices-hcm',
 'id': {'host-name': 'WF-CSIM-MX960-1-re0'},
 'post': {'//package-information/name': 'jservices-hcm'},
 'pre': {'//package-information/name': 'jservices-hcm'}},
{'actual_node_value': 'jservices-crypto-base',
 'id': {'host-name': 'WF-CSIM-MX960-1-re0'},
 'post': {'//package-information/name': 'jservices-crypto-
base'},
                                   'pre': {'//package-information/name': 'jservices-crypto-
base'}},
                                  {'actual_node_value': 'jservices-cpcd',
                                   'id': {'host-name': 'WF-CSIM-MX960-1-re0'},
                                   'post': {'//package-information/name': 'jservices-cpcd'},
                                   'pre': {'//package-information/name': 'jservices-cpcd'}},
                                  {'actual_node_value': 'jservices-bgf',
                                   'id': {'host-name': 'WF-CSIM-MX960-1-re0'},
                                   'post': {'//package-information/name': 'jservices-bgf'},
                                   'pre': {'//package-information/name': 'jservices-bgf'}},
                                  {'actual_node_value': 'jservices-appid',
                                   'id': {'host-name': 'WF-CSIM-MX960-1-re0'},
                                   'post': {'//package-information/name': 'jservices-appid'},
                                   'pre': {'//package-information/name': 'jservices-appid'}},
                                  {'actual_node_value': 'jservices-alg',
                                   'id': {'host-name': 'WF-CSIM-MX960-1-re0'},
                                   'post': {'//package-information/name': 'jservices-alg'},
                                   'pre': {'//package-information/name': 'jservices-alg'}},
                                  {'actual_node_value': 'jservices-aacl',
                                   'id': {'host-name': 'WF-CSIM-MX960-1-re0'},
                                   'post': {'//package-information/name': 'jservices-aacl'},
                                   'pre': {'//package-information/name': 'jservices-aacl'}},
                                  {'actual_node_value': 'jpfe-platform',
                                   'id': {'host-name': 'WF-CSIM-MX960-1-re0'},
                                   'post': {'//package-information/name': 'jpfe-platform'},
                                   'pre': {'//package-information/name': 'jpfe-platform'}},
                                  {'actual_node_value': 'jpfe-common',
                                   'id': {'host-name': 'WF-CSIM-MX960-1-re0'},
                                   'post': {'//package-information/name': 'jpfe-common'},
                                   'pre': {'//package-information/name': 'jpfe-common'}},
                                  {'actual_node_value': 'jdocs',
                                   'id': {'host-name': 'WF-CSIM-MX960-1-re0'},
                                   'post': {'//package-information/name': 'jdocs'},
                                   'pre': {'//package-information/name': 'jdocs'}},
                                  {'actual_node_value': 'fips-mode',
                                   'id': {'host-name': 'WF-CSIM-MX960-1-re0'},
                                   'post': {'//package-information/name': 'fips-mode'},
                                   'pre': {'//package-information/name': 'fips-mode'}}],
                     'result': True,
                     'testoperation': 'exists',
                     'xpath': '//software-information'}]}
premesh:testfiles premesh$
```

Here's a small trick. What if you want to check the XPath but also want to ignore some particular value? You can do this with code "`!=`".

Suppose you want to check the number for peering for the OSPF interface, but you know lo0 will always fail, as it does not have peering. You want to check all interfaces except lo0.0, like this:

```
xpath: ospf-interface[interface-name != "lo0.0"]
```

Summary

That's it. That's JSNAPy. Join us on GitHub and learn a lot more: https://github.com/Juniper/jsnapy.

Appendix

OSPF Test Case

Here is the book's OSPF test case, including all the operators:

```
test_ospf_all_operator.yml
tests_include:
  - ospf-neighbor
  - ospf_interface
  - OSPF-DB
  - ospf-md5


ospf-md5:
- command: show configuration protocols ospf | display inheritance | display xml
- iterate:
    id: name
    tests:
    - err: ' OSPF interface {{post["name"]}} is configured without authentication.'
      exists: authentication
      info: All OSPF interface has authentication enabled
    - err: 'OSPF interface {{post["name"]}} has authentication enabled'
      not-exists: authentication
      info: 'OSPF interface {{post["name"]}} is configured without authentication.'
    - err: ' OSPF interface {{post["name"]}} seems not a physical interface.'
      contains: name, -
      info: All OSPF interface is physical interface.
    xpath: protocols/ospf/area/interface


ospf-neighbor:
- command: show ospf neighbor
- iterate:
    id: interface-name
    tests:
     - err: 'OSPF interface gone missing: {{pre["interface-
name"]}} going to {{pre["neighbor-address"]}}'
       info: OSPF interface list check
       list-not-less:
     - err: 'New OSPF interface added: {{post["interface-
name"]}} going to {{post["neighbor-address"]}}'
       info: OSPF interface list check
       list-not-more:
     - err: '   was going to {{pre["neighbor-address"]}}, now going to {{post["neighbor-
address"]}}'
       info: OSPF neighbor change check
       no-diff: neighbor-address
     - err: OSPF neighbor on {{post["interface-name"]}} to {{post["neighbor-
address"]}} is not up, rather {{post["ospf-neighbor-state"]}}
       info: All OSPF neighbors are up
       is-equal: ospf-neighbor-state, Full
     - err: 'All OSPF neighbors are up'
       info: 'OSPF neighbor on {{post["interface-name"]}} to {{post["neighbor-
address"]}} is not up, rather {{post["ospf-neighbor-state"]}}'
       not-equal: ospf-neighbor-state, Full
     - err: 'OSPF neighbor on {{post["interface-name"]}} to {{post["neighbor-
address"]}} state change from  {{pre["ospf-neighbor-state"]}} to {{post["ospf-neighbor-
state"]}}'
       info: 'No state change for OSPF neighbors'
       all-same: ospf-neighbor-state
     - err: 'OSPF neighbor on {{post["interface-name"]}} to {{post["neighbor-
address"]}} is not up, rather {{post["ospf-neighbor-state"]}}'
       info: 'All OSPF neighbors are up'
       is-in: ospf-neighbor-state, Up, Full
     - info: 'OSPF neighbor on {{post["interface-name"]}} to {{post["neighbor-
address"]}} is not up, rather {{post["ospf-neighbor-state"]}}'
       err: 'All OSPF neighbors are up'
       not-in: ospf-neighbor-state, Up, Full
    xpath: ospf-neighbor

ospf_interface:
- command: show ospf interface
- iterate:
    tests:
    - err: OSPF interface {{post["interface-name"]}} does not have any neighbors
      info: OSPF interfaces must have at least 1 neighbor
      is-gt: neighbor-count, 0
    - err: 'OSPF interfaces must have at least 1 neighbor, {{post["interface-
name"]}} has atleast 1 neighbor'
      info: 'OSPF interface {{post["interface-name"]}} does not have any neighbors'
      is-lt: neighbor-count, 1
    xpath: ospf-interface[interface-name != "lo0.0"]

OSPF-DB:
- command: show ospf database detail
- iterate:
    id: ../advertising-router
    tests:
    - err: Router {{post["../advertising-router"]}} has {{post["link-count"]}} links
      in-range: link-count, 5, 10
      info: OSPF router links [5..10]
    - err: Router {{post["../advertising-router"]}} has {{post["link-count"]}} links
      info: OSPF router links not[5..10]
      not-range: link-count, 5, 10
    - err: 'Router {{post["../advertising-router"]}} has changed to {{post["link-
count"]}} links earlier it was {{pre["link-count"]}} '
      info: 'OSPF router links not changed significantly for {{post["../advertising-
router"]}}'
      delta: link-count, 20%
    xpath: '/ospf-database-information/ospf-database/ospf-router-lsa'
```