

NETCONF & YANG API Orchestration Guide

Published
2022-04-28

RELEASE
3.3

Table of Contents

Introduction

Prerequisites and Preparations

Introduction to NETCONF Orchestration API

Supported Features in Paragon Active Assurance

Test and Monitor Templates

Examples of Controlling Paragon Active Assurance via NETCONF & YANG API

Examples: Test Agents

Examples: Inventory Items

Examples: Alarms

Examples: SSH Keys

Examples: Tests

Examples: Monitors

Using Tags

Troubleshooting

Notes on Test Agent Applications and Test Agent Appliances

Appendix: Tree Structure of Full YANG Model

Introduction

IN THIS SECTION

- [Purpose of This Document | 1](#)
- [Conventions | 1](#)
- [Notes on Backward Compatibility | 2](#)

Purpose of This Document

This documentation describes how to integrate Paragon Active Assurance with a network service orchestrator via the Control Center NETCONF & YANG API. Hands-on examples are given of the principal tasks involved, including: creating and deploying Virtual Test Agents, running tests and monitors, and retrieving results from these activities.

In this document, the freely available Python NETCONF client `ncclient` is used in the role of orchestrator.

Conventions

The following abbreviations are used in this document:

Abbreviation	Meaning
CLI	Command Line Interface
EM	Element Manager
ES	Errored Second
MEP	MEG (Maintenance Entity Group) End Point (<i>ITU-T Y.1731 definition</i>) or Maintenance End Point (<i>Cisco definition</i>)

(Continued)

Abbreviation	Meaning
NFV	Network Function Virtualization
NFVO	Network Function Virtualization Orchestrator
NSD	Network Service Descriptor
RPC	Remote Procedure Call
SIP	Session Initiation Protocol
SLA	Service Level Agreement
S-VNFM	Special VNF Manager
VNF	Virtual Network Function
vTA	Virtual Test Agent

Notes on Backward Compatibility

In versions 2.35.4/2.36.0 of the NETCONF & YANG API, the validation of certain requests was made more stringent to adhere to the NETCONF standard. This means that client code based on older versions of this guide might now be rejected.

For example, in previous Python example code, no namespace attribute was provided. The namespace now needs to be supplied in the request XML whenever you want to modify a ConfD resource.

Prerequisites and Preparations

IN THIS SECTION

- [ConfD Installation | 3](#)
- [Verifying That ConfD Is Running | 3](#)
- [Synchronizing the Configuration Database with Control Center | 4](#)
- [Setting Up Multiple NETCONF-controlled Paragon Active Assurance Accounts | 6](#)

ConfD Installation

ConfD (a product from Tail-f) is used as an intermediary between the Paragon Active Assurance system and NETCONF. ConfD connects Paragon Active Assurance configuration and operational data to the NETCONF & YANG API.

ConfD should have been installed along with the Control Center software, as described in the *Installation Guide*.

Verifying That ConfD Is Running

To verify that the ConfD is up and running, run the command

```
ssh -s <username>@localhost -p 830 netconf
```

to check that ConfD responds on port 830. In the command, <username> is as defined by the

```
netconf user create
```

command in the Installation Guide, section *Installing ConfD*. Give the password defined by the same command.

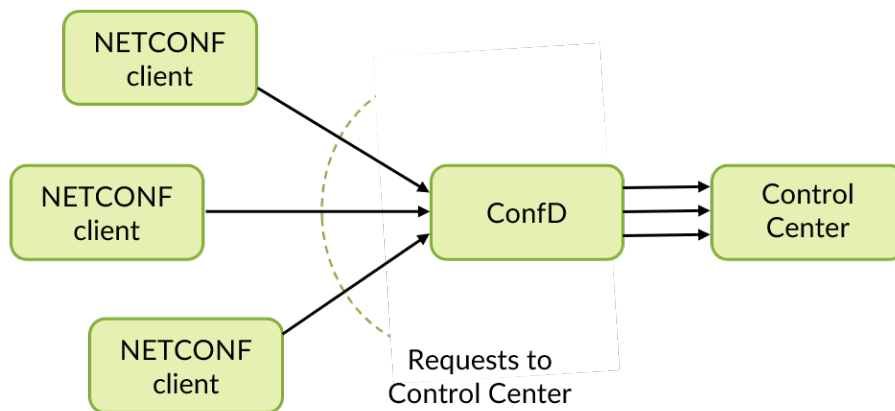
In the output, verify that the Control Center module is included. The output should contain a line like the following:

```
<capability>http://ncc.netrounds.com?module=netrounds-ncc&revision=2017-06-15</capability>
```

Synchronizing the Configuration Database with Control Center

Finally, we need to update the configuration database through NETCONF. We will do so here by means of a Python library called ncclient (NETCONF Client). However, the task could also be accomplished in a different programming language as long as it uses the NETCONF/YANG protocol.

The role of ncclient is to act as a client towards the ConfD server that hosts the NETCONF/YANG API.



It is worth pointing out that ncclient is not related in any way to Control Center (previously "Netrounds Control Center"), although the name happens to begin with "ncc".

Here is how to install ncclient:

- Download the software from <https://github.com/ncclient/ncclient>.
- Run this command: `pip install ncclient`

We can now perform the synchronization as follows. Note carefully that this needs to be done on a separate computer, and not on the Control Center server itself:

```
#
# NOTE:
# This script acts as a client towards ConfD running on the NCC server.
# It will use the NETCONF/YANG API for communication.
```

```

#
# The script is not meant to run on the NCC server!
# Run it on a separate client computer.
#

from ncclient import manager
from ncclient.xml_ import to_ele

# NETCONF server
host = '<host IP>'          # Replace with server IP of Control Center
port = 830
user = '<username>'        # Replace with username defined in "netconf user create"
command
password = '<password>'    # Replace with password defined in "netconf user create"
command

# Product account
netrounds_account = '<account>' # Replace with account name

with manager.connect(host=host, port=port, username=user, password=password,
                    hostkey_verify=False) as m:

    # Update config database
    xml = "<<sync-from-ncc xmlns='http://ncc.netrounds.com' />>>"

    # Convert to ElementTree Element
    elem = to_ele(xml)
    print m.dispatch(elem)

```

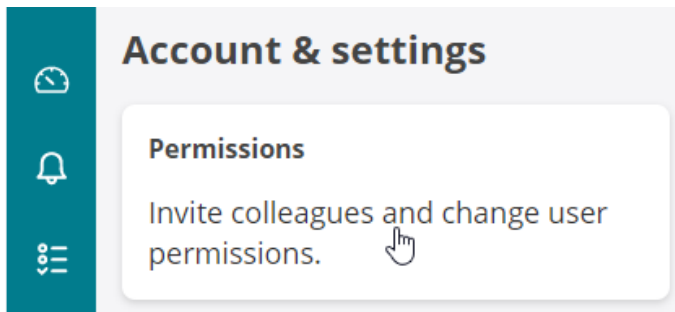
NOTE: This procedure is also required whenever Test Agents have been installed and registered independently of NETCONF. See the note in the section ["Overview of Test Agent Orchestration" on page 17](#) for more information.

Setting Up Multiple NETCONF-controlled Paragon Active Assurance Accounts

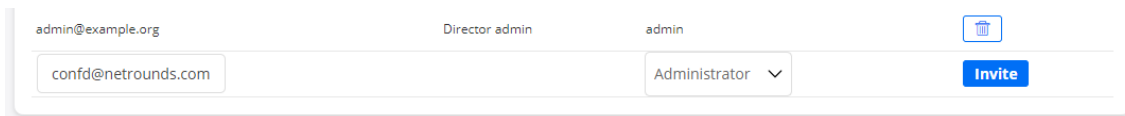
The steps below are required only if you wish to set up further Paragon Active Assurance accounts to be controlled by NETCONF, in addition to the account configured in this way in the *Installation Guide*, section "Installing ConfD".

For each such account, proceed as follows:

- In Control Center, log in to the account and navigate to **Account > Permissions**.



- Add the user "confd@netrounds.com", and grant this ConfD user admin permission in the GUI by clicking the **Invite** button.



- Synchronize the configuration database with Control Center as described in the section ["Synchronizing the Configuration Database with Control Center" on page 4](#).

You should now be able to control multiple Paragon Active Assurance accounts with the same ConfD user.

NOTE: Once you start controlling a Paragon Active Assurance account via ConfD, you must NOT make changes to this account through the web GUI with respect to any Paragon Active Assurance features that are "config" (see the chapter ["Supported Features in Paragon Active Assurance" on page 9](#)). If you do, loss of sync will result.

Introduction to NETCONF Orchestration API

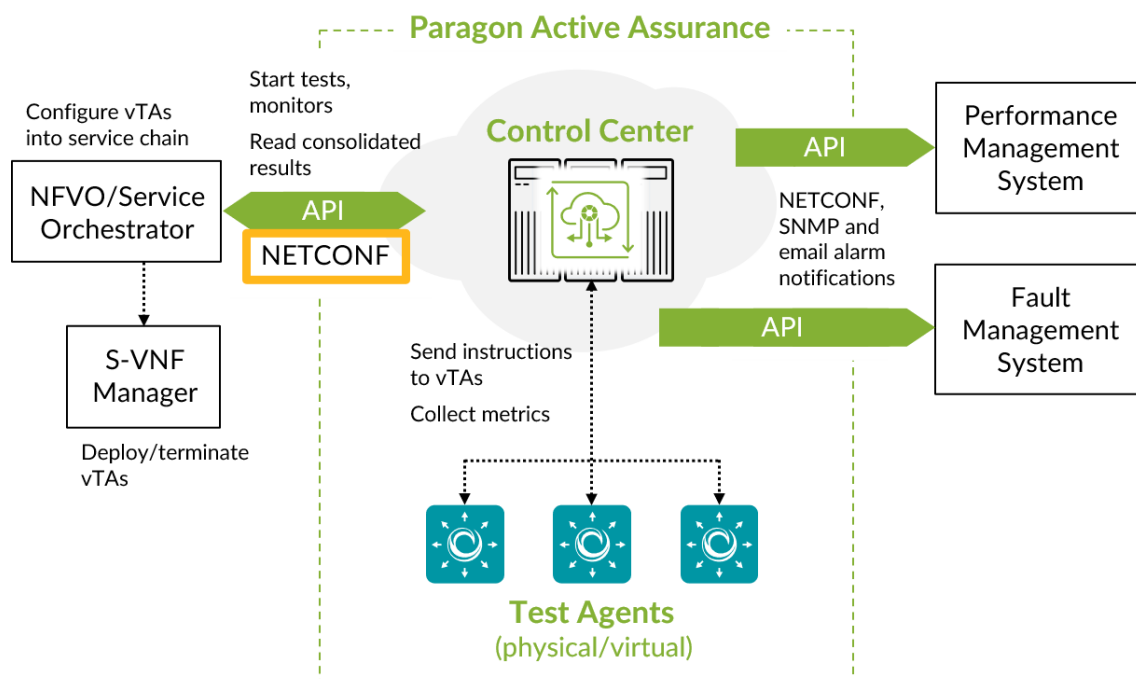
IN THIS SECTION

- Overview | 7
- Definitions of Concepts in Paragon Active Assurance | 8
- Workflow for Automation | 9

Overview

A third-party NFVO or service orchestrator is typically the component which initiates test and monitoring sessions using the Control Center API. This orchestrator also retrieves the aggregated measurement results from the Test Agent activities. Performance KPIs may be retrieved by third-party Performance Management Systems, while events – once triggered by threshold violations set in the Control Center – can be sent to third-party Fault Management systems.

To summarize, the figure below shows how Paragon Active Assurance interacts with other third-party systems in the OSS landscape.



- **NFVO/Service Orchestrator:** Instructs the VNF Manager to deploy the vTAs and configure Paragon Active Assurance into the service chain. Once the service has been activated, the orchestrator uses the API towards Control Center to trigger service activation tests and retrieve pass/fail results. If the tests are passed, the orchestrator will use the API towards Control Center to start active monitoring of the service. KPIs from the monitoring are retrieved continuously either by the orchestrator or by a separate Performance Management platform.
- **Control Center:** Deploys, scales, and terminates the vTA as instructed by the NFVO or service orchestrator.
- **Performance Management system or Service Quality Management system:** Reads KPIs from active monitoring via the Control Center API.
- **Fault Management system:** Receives NETCONF, SNMP, or email notifications from Control Center if SLAs are violated.

Definitions of Concepts in Paragon Active Assurance

- **Test Agents:** The components that perform measurements (for tests as well as monitors) in a Paragon Active Assurance system. Test Agents consist of software with the ability to generate, receive, and analyze real network traffic.
 - The kind of Test Agent discussed in this document is the *Virtual Test Agent (vTA)*, a virtual network function (VNF) deployed on a hypervisor. Other types of Test Agent also exist.
- There are two basic types of measurement in Paragon Active Assurance, *tests* and *monitors*.
 - **Test:** A test consists of one or several *steps*, each of which has a *specified, finite duration*. Steps are executed sequentially. Each step may entail running multiple *tasks* concurrently.
 - **Monitor:** A monitor does not have a specified duration but executes *indefinitely*. Like a step in a test, a monitor may execute multiple concurrent tasks.
- **Template:** When Paragon Active Assurance is controlled by an orchestrator, tests and monitors are always executed by means of templates in which the test or monitor is defined. Parameter settings can be passed as inputs to the template at runtime.

Workflow for Automation

Design Time

At design time, you prepare measurements by creating templates for tests and monitors in Paragon Active Assurance. How to do that is covered in the chapter ["Test and Monitor Templates" on page 15](#).

Runtime

At runtime, you set up your devices and perform the actual measurements.

- An overview of all examples given is found in the chapter ["Examples of Controlling Paragon Active Assurance via NETCONF & YANG API" on page 15](#).
- How to deploy and configure Test Agents is gone through in the chapter ["Examples: Test Agents" on page 16](#).
- How to import inventory items such as TWAMP reflectors and IPTV channels is gone through in the chapter ["Examples: Inventory Items" on page 29](#).
- How to configure alarms is explained in the chapter ["Examples: Alarms" on page 35](#).
- How to run tests and monitors by executing Paragon Active Assurance templates through NETCONF is described in the chapters ["Examples: Tests" on page 43](#) and ["Examples: Monitors" on page 54](#).

Supported Features in Paragon Active Assurance

IN THIS SECTION

- [Explanation of YANG Constructs | 10](#)
- [Tables of Paragon Active Assurance Features Available for Orchestration | 10](#)
- [Supported NETCONF Capabilities | 14](#)

All test and monitor types in Paragon Active Assurance can be created and executed through the use of templates. How to do this is covered in the in-app help under "Tests and monitors" > "Creating templates".

Creation of Paragon Active Assurance accounts is currently not supported; however, one or several predefined accounts will have been set up for the user.

The tables below detail what features in Paragon Active Assurance are available in this release, and how these features are represented in YANG.

Explanation of YANG Constructs

For convenience, definitions are given here of the YANG constructs referred to in the feature table.

- **Config** (config=true): Configuration data, required to transform a system from one state to another.
- **State** (config=false): State data: additional data on a system that is not configuration data, such as read-only status information and collected statistics.
- **RPC**: A Remote Procedure Call, as used within the NETCONF protocol.
- **Notification**: Event notifications sent from a NETCONF server to a NETCONF client.

Tables of Paragon Active Assurance Features Available for Orchestration

Resource: Monitoring

YANG path:/accounts/account/monitors

Feature	Subfeature	YANG construct
Create/modify/delete monitor	Based on monitor template	Config
Start/stop monitor	-	Config
Monitor templates	List existing monitor templates with inputs	State
NETCONF notifications	Alarm state changed	Notification
Monitor results	SLA/ES counter for top level (%)	State
	SLA/ES counter for task level (%)	State

Note

Unlike tests (compare **Resource: Tests** below), monitors are not started with an RPC but rather by committing the monitor configuration.

Resource: Tests

YANG path: /accounts/account/tests

Feature	Subfeature	YANG construct
Start test	Based on test template	RPC
Manage tests	List tests with status	State
Test templates	List existing test templates with inputs	State
NETCONF notifications	Test status changed	Notification
Test results	Get test step status (pass, fail, error, ...)	State

Resource: Test Agents

YANG paths:

- /accounts/account/test-agents (Config)
- /accounts/account/registered-test-agents (State)

Test Agents under /accounts/account/test-agents are the ones that are config in an account. Only these Test Agents can be configured and used in tests and monitors via NETCONF by the orchestrator.

After you have configured a Test Agent and it has registered to the account, the Test Agent will appear under /accounts/account/registered-test-agents. You can find all registered Test Agents using a "get" command in NETCONF (compare the chapter Examples: Test Agents).

Under /accounts/account/registered-test-agents you may also find Test Agents that have not yet been configured. Any such Test Agents must be configured before they can be used.

In an orchestration scenario, it is generally recommended that you do all configuration of your Paragon Active Assurance account through NETCONF. This ensures that test-agents and registered-test-agents do not diverge.

Feature	Subfeature	YANG construct
"Pre-create" Test Agent on server	–	Config
Configure offline Test Agent	(Control Center pushes config to Test Agent when it comes online)	Config
Use existing/externally configured Test Agents	Use in test/monitor	Config
	Configure interfaces	Config
	Get status	State
Configure Test Agent (Test Appliance only)	Configure NTP	Config
	Configure bridges	Config
	Configure VLAN interfaces	Config
	Configure SSH keys	Config
	IPv6	Config
Utils	Reboot	RPC
	Update	RPC
NETCONF notifications	Online status changed	Notification
Status	Get system status (uptime, memory usage, load average, version)	State

Resource: Inventory

YANG path: /accounts/account/twamp-reflectors

Feature	Subfeature	YANG construct
Manage TWAMP reflector inventory	Create/modify/delete TWAMP reflectors	Config

YANG path: /accounts/account/y1731-meps

Feature	Subfeature	YANG construct
Manage Y.1731 MEP inventory	Create/modify/delete MEPs	Config

YANG path: /accounts/account/iptv-channels

Feature	Subfeature	YANG construct
Manage IPTV channel inventory	Create/modify/delete IPTV channels	Config

YANG path: /accounts/account/ping-hosts

Feature	Subfeature	YANG construct
Manage Ping host inventory	Create/modify/delete Ping hosts	Config

YANG path: /accounts/account/sip-accounts

Feature	Subfeature	YANG construct
Manage SIP account inventory	Create/modify/delete SIP accounts	Config

Resource: Alarms

YANG path: /accounts/account/alarm-templates

Feature	Subfeature	YANG construct
Configure alarm templates	Create/modify/delete alarm templates	Config

YANG path: /accounts/account/alarm-email-lists

Feature	Subfeature	YANG construct
Configure alarm email lists	Create/modify/delete alarm email lists	Config

YANG path: /accounts/account/snmp-managers

Feature	Subfeature	YANG construct
Configure SNMP managers	Create/modify/delete SNMP managers	Config

Resource: Global Statistics

YANG path: /statistics

Feature	Subfeature	YANG construct
Retrieve statistics for entire Control Center	--	Config

Resource: Account Statistics

YANG path: /accounts/account/statistics

Feature	Subfeature	YANG construct
Retrieve statistics for specific account	--	Config

Supported NETCONF Capabilities

The table below points to the IETF RFCs describing the NETCONF capabilities used for the purpose of Paragon Active Assurance orchestration.

- `ietf-netconf.yang`
 - IETF RFC 6241, Network Configuration Protocol (NETCONF), <https://tools.ietf.org/html/rfc6241>
 - The only supported error handling method is rollback-on-error.

- The only supported data store is writable-running.
- `ietf-netconf-notifications.yang`
- IETF RFC 5277, NETCONF Event Notifications, <https://tools.ietf.org/html/rfc5277>

Test and Monitor Templates

Templates for test and monitor types need to be set up manually through the Paragon Active Assurance front-end user interface. How to do this is covered in the in-app help under "Tests and monitors" > "Creating templates".

Examples of Controlling Paragon Active Assurance via NETCONF & YANG API

IN THIS SECTION

- [Tools Used in Examples | 15](#)
- [Overview of Key Tasks Performed | 16](#)

In the chapters that follow, it is assumed that suitable test and monitor templates have been defined according to the instructions given in the chapter "[Test and Monitor Templates](#)" on page 15.

Tools Used in Examples

All the examples in the subsequent chapters have been constructed using the following freely available tools:

- **Pyang:** Used to visualize and browse the YANG models.
 - Available at <https://github.com/mbj4668/pyang> (clone from git and run `python setup.py install`).

- **Python NETCONF client "ncclient":** Used to communicate with Control Center using NETCONF.
 - Available at <https://github.com/ncclient/ncclient> (run `pip install ncclient`).

The `netrounds-ncc.yang` data model is found in `/opt/netrounds-confd` once ConfD has been installed according to the *Installation Guide*).

Overview of Key Tasks Performed

(Some further tasks are also exemplified in what follows.)

- "Creating and deploying a new Test Agent" on page 16
- "Creating inventory items (e.g. reflectors)" on page 29
- "Setting up alarm templates and where to send alarms" on page 35
- "Creating and running a test" on page 45
- "Retrieving test results" on page 50
- "Starting a monitor (includes setup of alarms)" on page 60
- "Retrieving SLA status for a monitor" on page 67
- "Working with tags" on page 71

Examples: Test Agents

IN THIS SECTION

- Overview of Test Agent Orchestration | 17
- Creating and Deploying a New Test Agent | 17
- Listing the Test Agents in Your Paragon Active Assurance Account | 25
- Deleting a Test Agent | 27
- NETCONF Notifications | 28

Overview of Test Agent Orchestration

Test Agents in Paragon Active Assurance are considered as "configuration" in the context of orchestration. This means that creation, control, and deletion of Test Agents should be done via the orchestrator and NETCONF rather than via the Paragon Active Assurance GUI.



IMPORTANT: If a Test Agent is installed by a technician and registered to Control Center without first being created via the NETCONF & YANG API, the Test Agent will not exist in the configuration database, and the system will get out of sync. For ConfD to become aware of the Test Agent in this case, it will be necessary to perform a new synchronization with Control Center, as detailed in the section ["Synchronizing the Configuration Database with Control Center" on page 4](#).

Orchestration of Virtual Test Agents (vTAs) should therefore rather be done in the following steps:

1. **Create the Virtual Test Agent**, including its interface configuration, **using the NETCONF & YANG interface to Control Center**. The name of the Test Agent will be its unique key.
2. **Deploy the vTA on a virtualization platform**. Follow the instructions in the online help under Test Agents > Installation. The basic interface configuration that allows the vTA to connect to Control Center, as well as credentials for authentication, is provided into the vTA using cloud-init user data. Once the vTA has booted, it will automatically connect to Control Center using an encrypted OpenVPN connection. A NETCONF notification is sent since the value of the vTA's test-agent-status-change parameter has now changed to "online".

NOTE: Since the name of the vTA is its identifier in Control Center, this name must be the same as that defined in Control Center in ["step 1" on page 17](#).

3. Once the vTA has connected and authenticated to Control Center, **the interface configuration is pushed to the vTA**. This is the interface configuration provided in ["step 1" on page 17](#) when the vTA was created in Control Center.
4. After the vTA has served its purpose, **delete the vTA**.

Creating and Deploying a New Test Agent

We first need to create a Test Agent using the NETCONF & YANG interface to Control Center. When a Test Agent is created in this way, no synchronization with Control Center is needed.

The YANG model for a Test Agent is as depicted below. It is obtained as output from the command

```
pyang -f tree netrounds-ncc.yang
```

The full YANG model is given in ["Appendix: Tree Structure of Full YANG Model" on page 81](#), which also contains a legend explaining the conventions used in this and other YANG model illustrations in the present document.

```

+---rw test-agents
| +---rw test-agent* [name]
|   +---rw name          string
|   +---rw description?  string
|   +---rw tags
|   | +---rw tag* [name]
|   |   +---rw name      -> ../../../../tags/tag/name
|   +---ro state
|   | +---ro status?      test-agent-status-t
|   | +---ro version?     string
|   | +---ro uptime?      int64
|   | +---ro memory-usage? percent-t
|   | +---ro load-average
|   |   +---ro last-1-minute?  decimal64
|   |   +---ro last-5-minutes? decimal64
|   |   +---ro last-15-minutes? decimal64
|   +---rw ntp
|   | +---rw interface?    string
|   | +---rw servers* [address]
|   | | +---rw address     union
|   | +---rw enable-ipv6?  boolean
|   +---rw management
|   | +---rw interface?    string
|   | +---rw use-public-address? boolean
|   +---rw interfaces
|   | +---rw interface* [name]
|   |   +---rw name          string
|   |   +---rw description?  string
|   |   +---rw interface-type? interface-type-t
|   |   +---rw mac?          yang:mac-address
|   |   +---rw mtu?          uint16
|   |   +---rw speed?        speed-t
|   |   +---rw bridge

```

```

| | | +--rw members
| | |   +--rw member* [interface]
| | |     +--rw interface -> ../../../../interface/name
| | | +--rw vlan
| | |   +--rw id          vlan-id-t
| | |   +--rw parent      -> ../../../../interface/name
| | | +--rw wifi
| | |   +--rw ssid?       string
| | |   +--rw bssid?      yang:mac-address
| | |   +--rw country?    string
| | |   +--rw ht          boolean
| | |   +--rw ht40        boolean
| | |   +--rw vht         boolean
| | |   +--rw sgi         boolean
| | |   +--rw ldpc        boolean
| | |   +--rw ht-mcs?     string
| | |   +--rw vht-max-mcs uint8
| | |   +--rw vht-max-mimo uint8
| | |   +--rw freq-2-4    boolean
| | |   +--rw freq-5      boolean
| | |   +--rw auth
| | |     +--rw (auth-type)?
| | |       +---:(personal)
| | |         +--rw personal!
| | |           +--rw cipher      cipher-auth-t
| | |           +--rw password?  string
| | |       +---:(eap-tls)
| | |         +--rw eap-tls!
| | |           +--rw cipher      cipher-auth-t
| | |           +--rw ca-cert?    string
| | |           +--rw client-cert? string
| | |           +--rw identity?   string
| | |           +--rw private-key? string
| | |       +---:(eap-ttls)
| | |         +--rw eap-ttls!
| | |           +--rw cipher      cipher-auth-t
| | |           +--rw ca-cert?    string
| | |           +--rw identity?   string
| | |           +--rw anonymous-identity? string
| | |           +--rw password?  string
| | |       +---:(peap)
| | |         +--rw peap!
| | |           +--rw cipher      cipher-auth-t

```

```

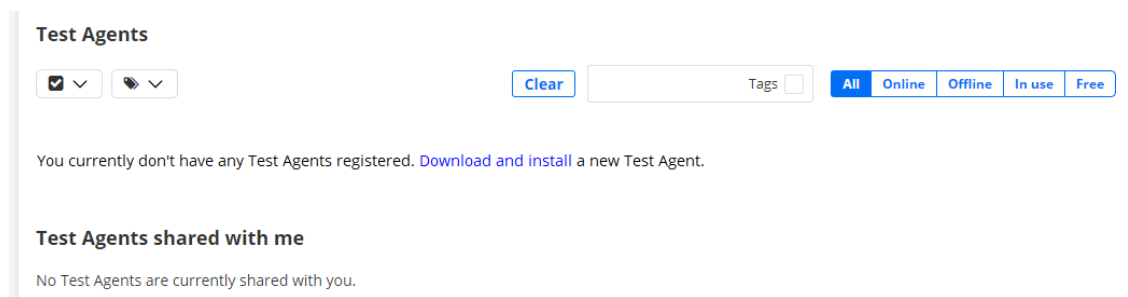
| | | +--rw ca-cert?      string
| | | +--rw identity?    string
| | | +--rw anonymous-identity? string
| | | +--rw password?    string
| | +--rw mobile
| | | +--rw apn?          string
| | | +--rw pdp_type      string
| | | +--rw rat_band      mobile-rat-band-t
| | | +--rw rat_mode      mobile-rat-mode-t
| | +--rw interface-config
| | | +--rw ipv4-address
| | | | +--rw (address-type)?
| | | | | +--:(static)
| | | | | | +--rw static!
| | | | | | +--rw address      tailf:ipv4-address-and-prefix-length
| | | | | | +--rw routes* [network]
| | | | | | | +--rw network    tailf:ipv4-address-and-prefix-length
| | | | | | | +--rw gateway    inet:ipv4-address
| | | | | | +--rw dns-servers* inet:ipv4-address
| | | | | +--:(dhcp)
| | | | | +--rw dhcp!
| | | | | +--rw vendor-id?    string
| | +--rw ipv6-address
| | | +--rw (address-type)?
| | | | +--:(static)
| | | | | +--rw static!
| | | | | | +--rw address      tailf:ipv6-address-and-prefix-length
| | | | | | +--rw routes* [network]
| | | | | | | +--rw network    tailf:ipv6-address-and-prefix-length
| | | | | | | +--rw gateway    inet:ipv6-address
| | | | | | +--rw dns-servers* inet:ipv6-address
| | | | | +--:(dhcp)
| | | | | +--rw dhcp!
| | | | | +--rw vendor-id?    string
| | | | +--:(slaac)
| | | | +--rw slaac!
| | | | +--rw dns-servers*    inet:ipv6-address
| +--rw ssh-keys
| | +--rw ssh-key* [name]
| | | +--rw name          string
| | | +--rw ssh-key-value string
| | +--ro test-agent?     -> /accounts/account/test-agents/test-agent/name

```

We proceed in the following steps, which are detailed in the following:

1. At the outset, the Paragon Active Assurance account "demo" has no Test Agents in its inventory.
2. A Test Agent called "vta1" is created using ncclient. At this stage, no real Test Agent exists yet (that is, it has not yet been started).
3. The Test Agent is deployed in OpenStack. (Deployment on that platform is chosen here as one possibility among others.)
4. The Test Agent connects to the Control Center account "demo" and is now ready for use.

Step 1: At the outset, there are no Test Agents in the account "demo". See the screenshot below from the Control Center GUI.



Step 2: A Test Agent is created in Control Center using the Python NETCONF client "ncclient". Below is ncclient code for creating a Test Agent having one physical interface with a DHCP address:

```
import argparse

from ncclient import manager

parser = argparse.ArgumentParser(description='Test creating Test Agent')
parser.add_argument('--host', help='The hostname where ConfD is found', required=True)
parser.add_argument('--port', help='The port to connect to ConfD', required=True)
parser.add_argument('--username', help='The username to connect to ConfD', required=True)
parser.add_argument('--password', help='Password to the ConfD account', required=True)
parser.add_argument('--netrounds-account', help='The NCC account short name', required=True)
parser.add_argument('--test-agent-name', help='Name of Test Agent', required=True)
args = parser.parse_args()

with manager.connect(host=args.host, port=args.port, username=args.username,
                    password=args.password, hostkey_verify=False) as m:

    # Create Test Agent in Control Center
    xml = """<config xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
```

```

<accounts xmlns="http://ncc.netrounds.com">
  <account>
    <name>demo</name>
    <test-agents>
      <test-agent>
        <name>vta1</name>
        <description/>
        <ntp>
          <interface>eth0</interface>
          <server>time.google.com</server>
        </ntp>
        <management>
          <interface>eth0</interface>
        </management>
        <interfaces>
          <interface>
            <name>eth0</name>
            <description/>
            <interface-type>physical</interface-type>
            <mac>fa:16:3e:ec:b3:a8</mac>
            <mtu>1400</mtu>
            <speed>auto</speed>
            <interface-config>
              <ipv4-address>
                <dhcp/>
              </ipv4-address>
              <ipv6-address></ipv6-address>
            </interface-config>
          </interface>
        </interfaces>
        <tags>
          <tag>
            <name>Tag 1</name>
          </tag>
        </tags>
      </test-agent>
    </test-agents>
  </account>
</accounts>
</config>
"".format(
account=args.netrounds_account,
testagent=args.test_agent_name

```



```
)

print m.edit_config(target='running', config=xml)
```

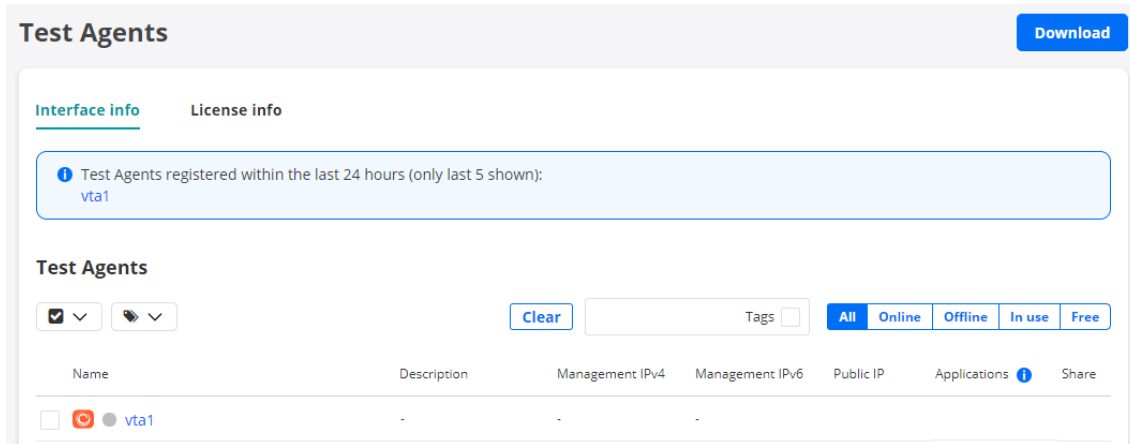
NOTE: The code preceding with `manager.connect(...)` is omitted from subsequent example code snippets.

An NTP server is configured on `eth0`, and `eth0` is also the management interface (that is, the interface that connects to Control Center).

A **Test Agent Application** does not currently allow configuring interfaces. For this reason, from version 2.34.0 onward, it is possible to omit the interface configuration in the YANG schema. The corresponding XML is therefore radically simplified in this case:

```
# Create Test Agent Application in Control Center
xml = """<config xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <accounts xmlns="http://ncc.netrounds.com">
        <account>
            <name>demo</name>
            <test-agents>
                <test-agent>
                    <name>vta2</name>
                    <description/>
                    <tags>
                        <tag>
                            <name>Tag 2</name>
                        </tag>
                    </tags>
                </test-agent>
            </test-agents>
        </account>
    </accounts>
</config>"""
```

Once the Test Agent has been created, it exists in the configuration database and in Control Center. See the screenshot below of the Test Agent inventory, showing the Test Agent "vta1":



Step 3: It is now time to deploy the Test Agent "vta1" in OpenStack.

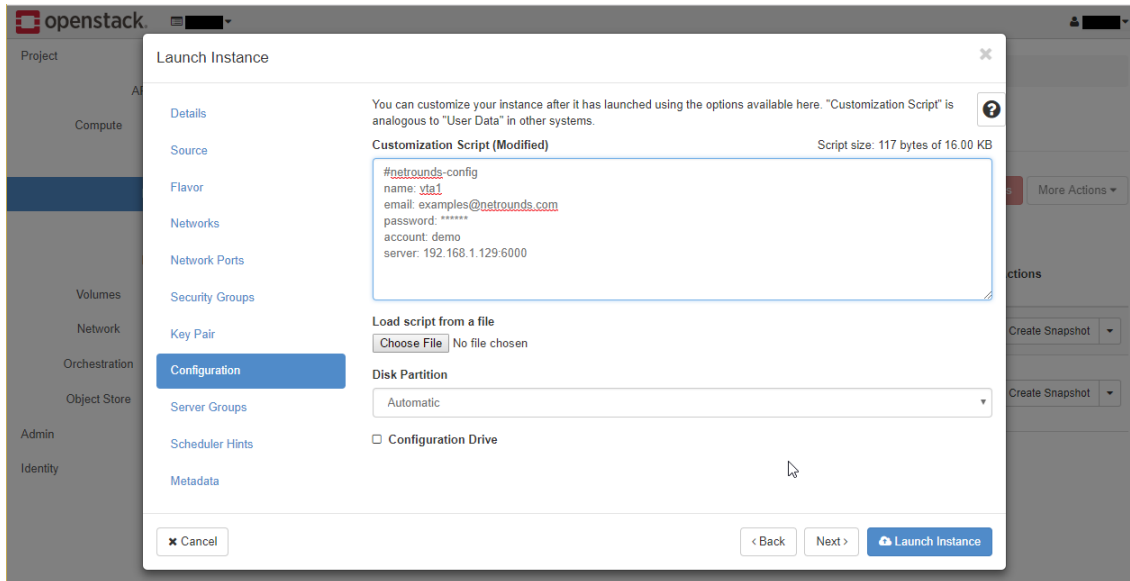
The Test Agent will use cloud-init user data to retrieve the information on how to connect to Control Center. Specifically, the user data text file has the following contents (**Note** that the `#cloud-config` and `netrounds_test_agent` lines must be present, and that the remaining lines must be indented):

```
#cloud-config
netrounds_test_agent:
  name: MyvTA                # Name of vTA to appear in Control Center inventory
  email: john.doe@example.com # Email you use when logging in to the system
  password: secret           # Login password
  account: theaccount         # Account name
  server: <login-server>:<port> # Control Center host and port (default == SaaS)

                                # Note: With an IPv6 server address the whole string
                                # including port must be in double quotes
```

For further information, please refer to the document *How to Deploy Virtual Test Agents in OpenStack*.

Once the Test Agent has been deployed and has connected to Control Center, the configuration will be pushed from Control Center to the Test Agent.



Step 4: The Test Agent is now online in Control Center and has obtained its configuration. The Test Agent is ready for use in tests and monitoring. See these sections:

- ["Starting a Test" on page 45](#)
- ["Starting a Monitor" on page 60](#)

Listing the Test Agents in Your Paragon Active Assurance Account

Below is example ncclient Python code for listing the Test Agents in a Paragon Active Assurance account:

```
import argparse

from ncclient import manager
from ncclient.xml_ import to_ele, to_xml

# (server and account details omitted)

with manager.connect(host=args.host, port=args.port, username=args.username,
                    password=args.password, hostkey_verify=False) as m:

    # Get config subtree
    # Get Test Agents from account
    filter = """<accounts xmlns="http://ncc.netrounds.com">
```

```

        <account>
            <name>{account}</name>
            <test-agents></test-agents>
        </account>
    </accounts>""".format(account=args.netrounds_account)
    ele = to_ele(m.get_config(source='running', filter=('subtree', xml)).data_xml)
    print to_xml(ele, encoding='UTF-8', pretty_print=True)

```

Running this code gives output like that below:

```

<?xml version="1.0" ?>
<data xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <accounts xmlns="http://ncc.netrounds.com">
    <account>
      <name xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">demo</name>
      <test-agents>
        <test-agent>
          <name>vta1</name>
          <description/>
          <ntp>
            <interface>eth0</interface>
            <server>time.google.com</server>
          </ntp>
          <management>
            <interface>eth0</interface>
          </management>
          <interfaces>
            <interface>
              <name>eth0</name>
              <description/>
              <interface-type>physical</interface-type>
              <mac>fa:16:3e:ec:b3:a8</mac>
              <mtu>1400</mtu>
              <speed>auto</speed>
              <interface-config>
                <ipv4-address>
                  <dhcp/>
                </ipv4-address>
                <ipv6-address></ipv6-address>
              </interface-config>
            </interface>

```

```

        </interfaces>
    </test-agent>
</test-agents>
</account>
</accounts>
</data>

```

Deleting a Test Agent

After a test has completed, it might be relevant in some use cases to delete the Test Agent.

Below is a code snippet showing how to do this with ncclient:

```

with manager.connect(host=args.host, port=args.port, username=args.username,
password=args.password,
                    hostkey_verify=False) as m:
    # Get config subtree

    # Delete Test Agent from account
    xml = """<config xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
        <accounts xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0" xmlns="http://
ncc.netrounds.com">
            <account>
                <name>demo</name>
                <test-agents>
                    <test-agent nc:operation="delete">
                        <name>{test_agent}</name>
                    </test-agent>
                </test-agents>
            </account>
        </accounts>
    </config>""".format(test_agent="vta2")

    test_agents = m.edit_config(target='running', config=xml)

```

NETCONF Notifications

Below, we present a simple example script for listening to all incoming NETCONF notifications from Control Center. These notifications are sent whenever certain events take place, such as a Test Agent going offline or a user-initiated test being completed. Based on the information carried in the notifications, users can assign automated follow-up actions in the orchestrator.

```
with manager.connect(host=args.host, port=args.port, username=args.username,
                    password=args.password,
                    hostkey_verify=False) as m:

    data = m.create_subscription(stream_name='ncc')
    # This will put ncclient in a state where it listens for notifications,
    # triggered for instance by a Test Agent being unplugged.

    # The example code below simply reads one notification. What to do with it is left
    # to the creativity of the user.
    while True:
        notification = m.take_notification()
        print notification._raw
```

When the above script is executed, ncclient will present the received notification in structured XML. See the example output below, which shows a Test Agent going offline unexpectedly.

```
<ncclient.operations.notify.Notification object at 0x7f30242b6f10>
<?xml version="1.0" encoding="UTF-8"?>
<notification xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0"><eventTime>
2017-02-03T15:09:55.939156+00:00</eventTime>
<test-agent-status-change xmlns='http://ncc.netrounds.com'>
  <account>demo</account>
  <test-agent>HW1</test-agent>
  <status>offline</status>
</test-agent-status-change>
</notification>
```

Examples: Inventory Items

IN THIS SECTION

- [Creating a TWAMP Reflector | 29](#)
- [Creating a Y.1731 MEP | 30](#)
- [Creating an IPTV Channel | 31](#)
- [Creating a Ping Host | 31](#)
- [Creating a SIP Account | 32](#)
- [Retrieving Inventory Items | 33](#)

Creating (importing) and managing inventory items such as TWAMP reflectors and Y.1731 MEPs is done in a similar way as for Test Agents. Below is XML and NETCONF code for defining such entities in Paragon Active Assurance through the NETCONF & YANG API and for retrieving lists of the items defined.

Creating a TWAMP Reflector

```
with manager.connect(host=args.host, port=args.port, username=args.username,
                    password=args.password, hostkey_verify=False) as m:

    # Create TWAMP reflector in account
    xml = """<config xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
        <accounts xmlns="http://ncc.netrounds.com">
            <account>
                <name>{account}</name>
                <twamp-reflectors>
                    <twamp-reflector>
                        <name>{name}</name>
                        <address>{address}</address>
                        <port>{port}</port>
                        <control-port>{control}</control-port>
                    </twamp-reflector>
                </twamp-reflectors>
            </account>
        </accounts>
    </config>"""
```

```

        </account>
    </accounts>
</config>""".format(account=args.netrounds_account,
                      name='My TWAMP Reflector',
                      address='192.168.6.113',
                      port=21000,
                      control=862)

monitor = m.edit_config(target='running', config=xml)

```

Creating a Y.1731 MEP

```

with manager.connect(host=args.host, port=args.port, username=args.username,
                    password=args.password, hostkey_verify=False) as m:

    # Create Y.1731 MEP in account
    xml = """<config xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
        <accounts xmlns="http://ncc.netrounds.com">
            <account>
                <name>{account}</name>
                <y1731-meps>
                    <y1731-mep>
                        <name>{y1731_mep}</name>
                        <mac>{mac}</mac>
                        <meg-level>{meg_level}</meg-level>
                    </y1731-mep>
                </y1731-meps>
            </account>
        </accounts>
    </config>""".format(account=args.netrounds_account,
                        y1731_mep='Y1731',
                        mac='08:00:27:aa:95:d1',
                        meg_level=2)

    monitor = m.edit_config(target='running', config=xml)

```


Creating an IPTV Channel

```

with manager.connect(host=args.host, port=args.port, username=args.username,
                    password=args.password, hostkey_verify=False) as m:

    # Create IPTV channel in account
    xml = """<config xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
        <accounts xmlns="http://ncc.netrounds.com">
            <account>
                <name>(account)</name>
                <iptv-channels>
                    <iptv-channel>
                        <name>{iptv_channel_name}</name>
                        <ip>{ip}</ip>
                        <source>{source}</source>
                        <port>{port}</port>
                        <pnum>{pnum}</pnum>
                    </iptv-channel>
                </iptv-channels>
            </account>
        </accounts>
    </config>""".format(account=args.netrounds_account,
                        iptv_channel_name='My TV Channel',
                        ip='226.226.226.0',
                        source='226.226.226.0',
                        port=123,
                        pnum=123)

    monitor = m.edit_config(target='running', config=xml)

```

Creating a Ping Host

```

with manager.connect(host=args.host, port=args.port, username=args.username,
                    password=args.password, hostkey_verify=False) as m:

    # Create Ping host in account
    xml = """<config xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
        <accounts xmlns="http://ncc.netrounds.com">

```

```

        <account>
            <name>{account}</name>
            <ping-hosts>
                <ping-host>
                    <name>{name}</name>
                    <host>{host}</host>
                    <gps-lat>{gps_lat}</gps-lat>
                    <gps-long>{gps_long}</gps-long>
                </ping-host>
            </ping-hosts>
        </account>
    </accounts>
</config>"".format(account=args.netrounds_account,
                    name='My Ping Host',
                    host='192.168.6.113',
                    gps_lat=22.11,
                    gps_long=-40.40)

```

Creating a SIP Account

```

with manager.connect(host=args.host, port=args.port, username=args.username,
                    password=args.password, hostkey_verify=False) as m:

```

```

# Create SIP account in product account
xml = """<config xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <accounts xmlns="http://ncc.netrounds.com">
        <account>
            <name>{account}</name>
            <sip-accounts>
                <sip-account>
                    <sip-address>{sip_address}</sip-address>
                    <password>{password}</password>
                    <registrar>{registrar}</registrar>
                    <proxy>{proxy}</proxy>
                    <user-auth>{user_auth}</user-auth>
                    <uri-rewrite>{uri_rewrite}</uri-rewrite>
                </sip-account>
            </sip-accounts>
        </account>
    </accounts>
</config>

```

```

        </accounts>
    </config>""".format(account=args.netrounds_account,
                        sip_address='user_name@example.com',
                        password='my_password',
                        registrar='8.8.8.8',
                        proxy='example.com',
                        user_auth='my_auth_id',
                        uri_rewrite='abc')

    monitor = m.edit_config(target='running', config=xml)

```

Retrieving Inventory Items

Below is Python code for retrieving all inventory items defined in an account. (All types of inventory items are fetched in one go here in order to avoid some repetition in the document. Naturally, any subset of inventory items can be fetched by leaving out some of the lines under account below.)

```

with manager.connect(host=args.host, port=args.port, username=args.username,
                    password=args.password, hostkey_verify = False) as m:

    # Get inventory items from account
    xml = """<accounts xmlns="http://ncc.netrounds.com">
        <account>
            <name>{account}</name>
            <twamp-reflectors></twamp-reflectors>
            <y1731-meps></y1731-meps>
            <iptv-channels></iptv-channels>
            <sip-accounts></sip-accounts>
        </account>
    </accounts>""".format(account=args.netrounds_account)

    # Convert to ElementTree object
    ele = to_ele(m.get_config(source='running', filter=('subtree', xml)).data_xml)

    # Convert back to XML string and print
    print to_xml(ele, encoding='UTF-8', pretty_print=True)

```

Running this code gives output like that below:

```
<?xml version="1.0" ?>
<data xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <accounts xmlns="http://ncc.netrounds.com">
    <account>
      <name xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">demo</name>
      <twamp-reflectors>
        <twamp-reflector>
          <name>My TWAMP Reflector</name>
          <address>192.168.6.113</address>
          <port>21000</port>
          <control-port>862</control-port>
        </twamp-reflector>
      </twamp-reflectors>
      <y1731-meps>
        <y1731-mep>
          <name>Y1731</name>
          <mac>08:00:27:aa:95:d1</mac>
          <meg-level>2</meg-level>
        </y1731-mep>
      </y1731-meps>
      <iptv-channels>
        <iptv-channel>
          <name>My TV Channel</name>
          <ip>226.226.226.0</ip>
          <source>226.226.226.0</source>
          <port>123</port>
          <pnum>123</pnum>
        </iptv-channel>
      </iptv-channels>
      <sip-accounts>
        <sip-account>
          <sip-address>user_name@example.com</sip-address>
          <password>my_password</password>
          <registrar>8.8.8.8</registrar>
          <proxy>example.com</proxy>
          <user-auth>my_auth_id</user-auth>
          <uri-rewrite>abc</uri-rewrite>
        </sip-account>
      </sip-accounts>
    </account>
  </accounts>
</data>
```



```

        <addresses>{addresses1}</addresses>
        <addresses>{addresses2}</addresses>
    </alarm-email-list>
</alarm-email-lists>
</account>
</accounts>
</config>""".format(account=args.netrounds_account,
                    name='email list',
                    addresses1='thranduil@example.com',
                    addresses2='nimrodel@example.com')

print m.edit_config(target='running', config=xml)

```

Retrieving All Alarm Email Lists

```

with manager.connect(host=args.host, port=args.port, username=args.username,
                    password=args.password, hostkey_verify=False) as m:

    # Get email lists in account
    xml = """<accounts xmlns="http://ncc.netrounds.com">
        <account>
            <name>{account}</name>
            <alarm-email-lists></alarm-email-lists>
        </account>
    </accounts>""".format(account=args.netrounds_account)

    # Convert to ElementTree object
    ele = to_ele(m.get_config(source='running', filter=('subtree', xml)).data_xml)

    # Convert back to XML string and print
    print to_xml(ele, encoding='UTF-8', pretty_print=True)

```

SNMP Managers

Creating an SNMP Manager

```

with manager.connect(host=args.host, port=args.port, username=args.username,
                    password=args.password, hostkey_verify=False) as m:

```

```

# Create SNMP manager in account (version 2c)
xml = """<config xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <accounts xmlns="http://ncc.netrounds.com">
        <account>
            <name>{account}</name>
            <snmp-managers>
                <snmp-manager>
                    <name>{name}</name>
                    <ip>{ip}</ip>
                    <version>{version}</version>
                    <community>{community}</community>
                </snmp-manager>
            </snmp-managers>
        </account>
    </accounts>
</config>""".format(account=args.netrounds_account,
                      version='2c',
                      ip='8.8.8.8',
                      name='snmp manager',
                      community='my community string')

# Note: The user can also create a version 3 SNMP manager with the following parameters:

# <engine-id>{engine_id}</engine-id>
# <user-name>{user_name}</user-name>
# <security>{security}</security>
# <auth-password>{auth_password}</auth-password>
# <priv-password>{priv_password}</priv-password>

print m.edit_config(target='running', config=xml)

```

Retrieving All SNMP Managers

```

with manager.connect(host=args.host, port=args.port, username=args.username,
                    password=args.password, hostkey_verify=False) as m:

    # Get SNMP managers in account
    xml = """<accounts xmlns="http://ncc.netrounds.com">
        <account>
            <name>{account}</name>

```

```

        <snmp-managers></snmp-managers>
    </account>
</accounts>"".format(account=args.netrounds_account)

# Convert to ElementTree object
ele = to_ele(m.get_config(source='running', filter=('subtree', xml)).data_xml)

# Convert back to XML string and print
print to_xml(ele, encoding='UTF-8', pretty_print=True)

```

Alarm Templates

Creating an Alarm Template

```

# Note: This example requires existing an SNMP manager and alarm email list in order to work

with manager.connect(host=args.host, port=args.port, username=args.username,
password=args.password, hostkey_verify=False) as m:

    # Create alarm template in account
    xml = "<config xmlns='urn:ietf:params:xml:ns:netconf:base:1.0'>
        <accounts xmlns='http://ncc.netrounds.com'>
            <account>
                <name>{account}</name>
                <alarm-templates>
                    <alarm-template>
                        <name>{name}</name>
                        <email>{email}</email>
                        <snmp>{snmp}</snmp>
                        <action>{action}</action>
                        <window-size>{window_size}</window-size>
                        <interval>{interval}</interval>
                        <send-only-once>{send_only_once}</send-only-once>
                        <snmp-trap-per-stream>{snmp_trap_per_stream}</snmp-trap-per-
stream>

                        <thr-es-critical>{thr_es_critical}</thr-es-critical>
                        <thr-es-critical-clear>{thr_es_critical_clear}</thr-es-critical-
clear>

                        <thr-es-major>{thr_es_major}</thr-es-major>

```



```

        <thr-es-major-clear>{thr_es_major_clear}</thr-es-major-clear>
        <thr-es-minor>{thr_es_minor}</thr-es-minor>
        <thr-es-minor-clear>{thr_es_minor_clear}</thr-es-minor-clear>
        <thr-es-warning>{thr_es_warning}</thr-es-warning>
        <thr-es-warning-clear>{thr_es_warning_clear}</thr-es-warning-
clear>

        </alarm-template>
    </alarm-templates>
</account>
</accounts>
</config>""".format(account=args.netrounds_account,
                    name='template',
                    email='email list',
                    snmp='snmp manager',
                    action='some action string',
                    window_size=60,
                    interval=300,
                    send_only_once='false',
                    snmp_trap_per_stream='false',
                    thr_es_critical=9,
                    thr_es_critical_clear=8,
                    thr_es_major=7,
                    thr_es_major_clear=6,
                    thr_es_minor=5,
                    thr_es_minor_clear=4,
                    thr_es_warning=3,
                    thr_es_warning_clear=2)

# Note: The user can also provide additional parameters:

# <no-data-severity>{no_data_severity}</no-data-severity>
# <no-data-timeout>{no_data_timeout}</no-data-timeout>

print m.edit_config(target='running', config=xml)

```

Retrieving All Alarm Templates

```

with manager.connect(host=args.host, port=args.port, username=args.username,
                    password=args.password, hostkey_verify=False) as m:

    # Get alarm templates in account

```

```

xml = """<accounts xmlns="http://ncc.netrounds.com">
    <account>
        <name>{account}</name>
        <alarm-templates></alarm-templates>
    </account>
</accounts>""".format(account=args.netrounds_account)

# Convert to ElementTree object
ele = to_ele(m.get_config(source='running', filter=('subtree', xml)).data_xml)

# Convert back to XML string and print
print to_xml(ele, encoding='UTF-8', pretty_print=True)

```

Examples: SSH Keys

IN THIS SECTION

- [Adding an SSH Key | 41](#)
- [Deleting an SSH Key | 42](#)

You can add SSH public keys to a Test Agent via the NETCONF & YANG API. Using the corresponding private key you can then log in to the Test Agent via SSH.

The full list of available operations on SSH keys is as follows:

- Add an SSH key
- Modify an SSH key
- Inspect an SSH key
- List SSH keys
- Delete an SSH key.

Below, the add and delete operations are exemplified.

Adding an SSH Key

Here is how to create a new SSH key.

```
with manager.connect(host=args.host, port=args.port, username=args.username,
                    password=args.password, hostkey_verify=False) as m:

    xml = """
    <config xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
      <accounts xmlns="http://ncc.netrounds.com">
        <account>
          <name>{account}</name>
          <test-agents>
            <test-agent>
              <name>{test_agent}</name>
              <ssh-keys>
                <ssh-key>
                  <name>{key_name}</name>
                  <ssh-key-value>{key_value}</ssh-key-value>
                </ssh-key>
              </ssh-key>
            </test-agent>
          </test-agents>
        </account>
      </accounts>
    </config>""".format(account=args.netrounds_account,
                        key_name=args.key_name,
                        key_value=args.key_value,
                        test_agent=args.test_agent)

    monitor = m.edit_config(target='running', config=xml, default_operation='merge')

    print "ok" if monitor.ok else monitor.errors
```

Deleting an SSH Key

If you want to delete an SSH key, use the following command:

```
with manager.connect(host=args.host, port=args.port, username=args.username,
                    password=args.password, hostkey_verify=False) as m:

    xml = """
    <config xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
      <accounts xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0" xmlns="http://
ncc.netrounds.com">
        <account>
          <name>{account}</name>
          <test-agents>
            <test-agent>
              <name>{test_agent}</name>
              <ssh-keys>
                <ssh-key nc:operation="delete">
                  <name>{key_name}</name>
                </ssh-key>
              </ssh-key>
            </test-agent>
          </test-agents>
        </account>
      </accounts>
    </config>""".format(account=args.netrounds_account,
                        key_name=args.key_name,
                        test_agent=args.test_agent)

    monitor = m.edit_config(target='running', config=xml, default_operation='merge')

    print "ok" if monitor.ok else monitor.errors
```

Examples: Tests

IN THIS SECTION

- [YANG Model Paths for Tests | 43](#)
- [Prerequisites for Test Orchestration | 44](#)
- [Starting a Test | 45](#)
- [Retrieving Test Results | 50](#)
- [Exporting and Importing Test Templates | 52](#)

It is assumed here that Test Agents (as many as are required for the tests) have been created according to the section ["Creating and Deploying a New Test Agent" on page 17](#).

YANG Model Paths for Tests

Item	YANG model path: /accounts/account/tests ...
tests	/.
test[id]	/test
id	/test/id
name	/test/name
status	/test/status
start-time	/test/start-time
end-time	/test/end-time
report-url	/test/report-url

(Continued)

Item	YANG model path: /accounts/account/tests ...
steps	/test/steps
step[id]	/test/steps/step
name	/test/steps/step/name
id	/test/steps/step/id
start-time	/test/steps/step/start-time
end-time	/test/steps/step/end-time
status	/test/steps/step/status
status-message	/test/steps/step/status-message
templates	/templates
template[name]	/templates/template
name	/templates/template/name
description	/templates/template/description
parameters	/templates/template/parameters
parameter[key]	/templates/template/parameters/parameter
key	/templates/template/parameters/parameter/key
type	/templates/template/parameters/parameter/type

Prerequisites for Test Orchestration

- In order to start a test through NETCONF using ncclient, it is required to first build a test template using the Control Center GUI as detailed in the in-app help under "Tests and monitors" > "Creating

templates". All fields specified in that template as "Template input" will be required as parameters in the XML when orchestrating the initiation of the test template.

- Running tests in Paragon Active Assurance is considered as "state" in the context of orchestration. State data is non-writable data that is not stored in the configuration database, as opposed to the configuration data mentioned in the section ["Overview of Test Agent Orchestration" on page 17](#). This basically means that changes to tests or templates in the Control Center GUI will not cause any sync-related issues between Control Center and the configuration database.
- To get report-url right in test reports, you need to make sure the Control Center URL is correctly configured. This is done in the file `/opt/netrounds-confd/settings.py`. By default the Control Center host name is retrieved using `socket.gethostname()`: see below. If this does not yield the correct result, you need to set the host name (or the entire URL) manually in this file.

```
# URL of Control Center without trailing slash.
# This is for example used in test report-url.
HOSTNAME = socket.gethostname()
NETROUNDS_URL = 'https://%s' % HOSTNAME
```

Starting a Test

As described in the section ["Creating and Deploying a New Test Agent" on page 17](#), run the command

```
pyang -f tree netrounds-ncc.yang
```

from the directory `/opt/netrounds-confd/` in order to output the YANG model. In this model, the RPC for starting a test using ncclient looks as follows:

```

+---x start-test
| +---w input
| | +---w name          string
| | +---w account      -> /accounts/account/name
| | +---w template     string
| | +---w parameters
| |   +---w parameter* [key]
| |     +---w key              string
| |     +---w (value-type)
| |       +---:(integer)
| |       | +---w integer?      int64

```

```

| |      +---:(float)
| |      | +---w float?          decimal64
| |      +---:(string)
| |      | +---w string?         string
| |      +---:(test-agent-interfaces)
| |      | +---w test-agent-interfaces
| |      |   +---w test-agent-interface* [account test-agent interface]
| |      |     +---w account      -> ../../../../account
| |      |     +---w test-agent   -> deref ../../../../account)/../test-agents/test-
agent/name
| |      |       +---w interface  -> deref ../../test-agent)/../interfaces/interface/name
| |      |       +---w ip-version? inet:ip-version
| |      +---:(twamp-reflectors)
| |      | +---w twamp-reflectors
| |      |   +---w twamp-reflector* [name]
| |      |     +---w name        -> deref ../../../../account)/../twamp-reflectors/twamp-
reflector/name
| |      +---:(y1731-meps)
| |      | +---w y1731-meps
| |      |   +---w y1731-mep* [name]
| |      |     +---w name        -> deref ../../../../account)/../y1731-meps/y1731-mep/
name
| |      +---:(iptv-channels)
| |      | +---w iptv-channels
| |      |   +---w iptv-channel* [name]
| |      |     +---w name        -> deref ../../../../account)/../iptv-channels/iptv-
channel/name
| |      +---:(sip-accounts)
| |      | +---w sip-accounts
| |      |   +---w sip-account* [account test-agent interface sip-address]
| |      |     +---w account      -> ../../../../account
| |      |     +---w test-agent   -> deref ../../../../account)/../test-agents/
test-agent/name
| |      |       +---w interface  -> deref ../../test-agent)/../interfaces/interface/name
| |      |       +---w sip-address -> deref ../../../../account)/../sip-accounts/
sip-account/sip-address
| +---ro output
|   +---ro result      test-status-t
|   +---ro id?         uint32
|   +---ro error-text? string

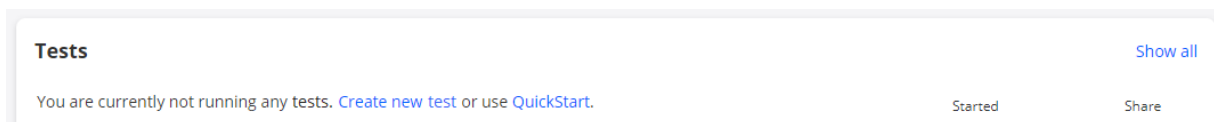
```

For explanations, see the section "[Legend](#)" on page 81 in the Appendix.

The following steps are shown below:

1. Test Agents have been registered to the Paragon Active Assurance account, but no tests have yet been started.
2. The required input parameters are identified in the test template that will be run.
3. A 60 second HTTP test is started using ncclient.

Step 1: At the outset, no tests have been initiated in the Paragon Active Assurance account. See the screenshot below from the Control Center GUI.



Step 2: The template we will use to initiate the test in this example is an HTTP test template. It has two mandatory input fields (Clients and URL) which we have specified as such when building the template in the Control Center GUI.

New test sequence

Name: Description:

Step 1

HTTP

Add Step

Step 1

HTTP

Step 1

General

Clients ⓘ

Select interfaces

URL ⓘ

We will define these parameters (among others) in the XML configuration communicated to the configuration database by our NETCONF manager (ncclient).

Step 3: The HTTP test is initiated using ncclient.

Below is example code where the required configuration information and parameters are specified for the HTTP test template. Depending on how the template has been built, the details here may vary.

For each parameter, the <key> attribute needs to be supplied. The key is identical to the parameter's Variable name in Control Center. You can inspect variable names as follows:

- Click **Tests** on the side bar and select **New Test Sequence**.
- Click **My Templates**.
- Click the **Edit** link below the template of interest.
- Click the **Edit input** button in the top right corner.

In our example, and by default, the variable names are simply lowercase versions of the display names seen in Control Center ("url" vs. "URL", etc.). However, in the Control Center GUI, you can rename the variables to whatever you like.

Besides the key, each parameter needs to have its *type* specified: for example, <string> for the URL. Please note that you need to review the complete YANG model in order to obtain full information on types. For Test Agent interfaces the type has a more complex structure, as evidenced under <clients> in the code below.

```
with manager.connect(host=args.host, port=args.port, username=args.username,
                    password=args.password, hostkey_verify=False) as m:

    # Run RPC "start-test"

    # A template named "HTTP_test" and having two inputs is assumed to exist
    xml = "<start-test xmlns='http://ncc.netrounds.com'>
        <name>Netconf Initiated HTTP test</name>
        <account>{account}</account>
        <template>HTTP_test</template>
        <parameters>
            <parameter>
                <key>clients</key>
                <test-agent-interfaces>
                    <test-agent-interface>
                        <account>{account}</account>
                        <test-agent>TA1</test-agent>
                        <interface>eth0</interface>
                    </test-agent-interface>
                </test-agent-interfaces>
            </parameter>
            <parameter>
                <key>url</key>
                <string>http://www.google.se</string>
            </parameter>
        </parameters>
    </start-test>".format(account=args.netrounds_account)

    # Convert to ElementTree element
    elem = to_ele(xml)

    print m.dispatch(elem)
```

We can now run the script using ncclient. Assuming all is correct, the test will be initiated and its execution displayed in Control Center:



If the test is successfully started, Control Center will respond with the test ID. In this example, the test ID is 3:

```
<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="urn:uuid:f08efc54-2086-4d09-af71-5b65fcecabe1"
xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"><id xmlns='http://ncc.netrounds.com'>3</id>
</rpc-reply>
```

The test ID can also be found in the URL for the test in the Control Center GUI. In this example, that URL is <https://host/demo/testing/3/>.

Retrieving Test Results

The most straightforward way to retrieve test results is by pointing to the test ID.

Below is Python code for getting the results from the above HTTP test with ID = 3:

```
with manager.connect(host=args.host, port=args.port, username=args.username,
                    password=args.password, hostkey_verify=False) as m:
```

```
# Get results from test in account
xml = """<accounts xmlns="http://ncc.netrounds.com">
    <account>
        <name>{account}</name>
        <tests>
            <test>
                <id>{id}</id>
            </test>
        </tests>
    </account>
</accounts>""".format(account=args.netrounds_account,
                        id=3)

# Convert to ElementTree object
ele = to_ele(m.get(filter=('subtree', xml)).data_xml)

# Convert back to XML string and print
print to_xml(ele, encoding='UTF-8', pretty_print=True)
```

The output will look something like this:

```
<?xml version="1.0" ?>
  <data xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
    <accounts xmlns="http://ncc.netrounds.com">
      <account>
        <name xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">demo</name>
        <tests>
          <test>
            <id xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">3</id>
            <name>HTTP</name>
            <status>passed</status>
            <end-time>2017-01-26T22:23:53+00:00</end-time>
            <report-url>https://localhost//demo/testing/3/</report-url>
            <steps>
              <step>
                <id>4</id>
                <name>Step 1</name>
                <start-time>2017-01-26T22:22:51+00:00</start-time>
                <end-time>2017-01-26T22:23:51+00:00</end-time>
                <status>passed</status>
```

```

                <status-message/>
            </step>
        </steps>
    </test>
</tests>
</account>
</accounts>
</data>

```

Exporting and Importing Test Templates

Test templates can be exported in JSON format and reimported in that format into Control Center. This is useful if you want to use test templates in a different installation of Control Center. (The initial creation of the templates is best handled through the Control Center GUI.)

Below is code for performing the export and import.

Exporting Test Templates

```

with manager.connect(host=args.host, port=args.port, username=args.username,
                    password=args.password, hostkey_verify=False) as m:

    # Export selected templates from Control Center
    # In order to export all templates, remove the <templates></templates> node
    export_xml = """
    <export-test-templates xmlns="http://ncc.netrounds.com">
        <account>{account}</account>
        <templates>
            <template>
                <name>template_udp</name>
            </template>
            <template>
                <name>template_tcp</name>
            </template>
        </templates>
    </export-test-templates>""".format(account=args.netrounds_account)

    elem = to_ele(export_xml)
    response = m.dispatch(elem)

```

```
# Get json config from response
root = ET.fromstring(response._raw)
json_config = root[0].text
print json_config
```

The template is contained in the `json_config` object.

Importing Test Templates

A JSON config object holding test templates can be reimported into Control Center as follows.

```
with manager.connect(host=args.host, port=args.port, username=args.username,
                    password=args.password, hostkey_verify=False) as m:

    json_string = """
        {
            "templates":[
                {
                    "input":[
                        {
                            "require_bridge":false,
                            # ...
                            # Insert rest of JSON string here
                            # ...
                        }
                    ],
                    "version": "2.29",
                    "export_date": "2019-05-29T12:39:31.172594Z"
                }
            ]
        }
    """

    # Import templates back into Control Center
    import_xml = """
        <import-test-templates xmlns="http://ncc.netrounds.com">
            <account>{account}</account>
            <overwrite>false</overwrite>
            <data>{data}</data>
        </import-test-templates>""".format(
            account=args.netrounds_account,
            data=json_string)
```

```
elem = to_ele(import_xml)
response = m.dispatch(elem)

print response
```

Examples: Monitors

IN THIS SECTION

- [YANG Model Paths for Monitors | 54](#)
- [Prerequisites for Monitor Orchestration | 59](#)
- [Getting Input Parameters from Monitor Templates | 59](#)
- [Starting a Monitor | 60](#)
- [Starting a Monitor with an Alarm | 62](#)
- [Retrieving Running Monitors | 65](#)
- [Retrieving SLA Status for a Monitor | 67](#)
- [NETCONF Notifications | 69](#)
- [Exporting and Importing Monitor Templates | 69](#)
- [Exporting Monitor Templates | 69](#)
- [Importing Monitor Templates | 70](#)

This section assumes that Test Agents (as many as are required by the monitors) have been created according to the section ["Creating and Deploying a New Test Agent" on page 17](#).

YANG Model Paths for Monitors

Item	YANG model path: /accounts/account/monitors ...
monitors	/.

(Continued)

Item	YANG model path: /accounts/account/monitors ...
monitor[name]	/monitor
name	/monitor/name
description	/monitor/description
started	/monitor/started
template	/monitor/template
alarm-configs	/monitor/alarm-configs

Item	YANG model path: /accounts/account/monitors/monitor/alarm-configs ...
alarm-config[identifier]	/alarm-config
identifier	/alarm-config/identifier
template	/alarm-config/template
email	/alarm-config/email
snmp	/alarm-config/snmp
thr-es-critical	/alarm-config/thr-es-critical
thr-es-critical-clear	/alarm-config/thr-es-critical-clear
thr-es-major	/alarm-config/thr-es-major
thr-es-major-clear	/alarm-config/thr-es-major-clear
thr-es-minor	/alarm-config/thr-es-minor
thr-es-minor-clear	/alarm-config/thr-es-minor-clear
thr-es-warning	/alarm-config/thr-es-warning

(Continued)

Item	YANG model path: /accounts/account/monitors/monitor/alarm-configs ...
thr-es-warning-clear	/alarm-config/thr-es-warning-clear
no-data-severity	/alarm-config/no-data-severity
no-data-timeout	/alarm-config/no-data-timeout
action	/alarm-config/action
window-size	/alarm-config/window-size
interval	/alarm-config/interval
send-only-once	/alarm-config/send-only-once
snmp-trap-per-stream	/alarm-config/snmp-trap-per-stream

Item	YANG model path: /accounts/account/monitors ...
parameters	/monitor/parameters

Item	YANG model path: /accounts/account/monitors/monitor/parameters ...
parameter[key]	/parameter
key	/parameter/key
(value-type)	/parameter
:(integer)	/parameter
integer	/parameter/integer
:(float)	/parameter
float	/parameter/float
:(string)	/parameter

(Continued)

Item	YANG model path: /accounts/account/monitors/monitor/parameters ...
string	/parameter/string
:(test-agent-interfaces)	/parameter
test-agent-interfaces	/parameter/test-agent-interfaces
test-agent-interface["1" on page 58]	/parameter/test-agent-interfaces/
account	/parameter/test-agent-interfaces/test-agent-interface/account
test-agent	/parameter/test-agent-interfaces/test-agent-interface/test-agent
interface	/parameter/test-agent-interfaces/test-agent-interface/interface
ip-version	/parameter/test-agent-interfaces/test-agent-interface/ip-version
:(twamp-reflectors)	/parameter
twamp-reflectors	/parameter/twamp-reflectors
twamp-reflector[name]	/parameter/twamp-reflectors/twamp-reflector
name	/parameter/twamp-reflectors/twamp-reflector/name
:(y1731-meps)	/parameter
y1731-meps	/parameter/y1731-meps
y1731-mep[name]	/parameter/y1731-meps/y1731-mep
name	/parameter/y1731-meps/y1731-mep/name
:(sip-accounts)	/parameter
sip-accounts	/parameter/sip-accounts
sip-account["2" on page 58]	/parameter/sip-accounts/sip-account

(Continued)

Item	YANG model path: /accounts/account/monitors/monitor/parameters ...
account	/parameter/sip-accounts/sip-account/account
test-agent	/parameter/sip-accounts/sip-account/test-agent
interface	/parameter/sip-accounts/sip-account/interface
sip-address	/parameter/sip-accounts/sip-account/sip-address
:(iptv-channels)	/parameter
iptv-channels	/parameter/iptv-channels
iptv-channel[name]	/parameter/iptv-channels/iptv-channel
name	/parameter/iptv-channels/iptv-channel/name

- 1. account test-agent interface
- 2. account test-agent interface sip-address

Item	YANG model path: /accounts/account/monitors ...
status	/monitor/status
last-15-minutes	/monitor/status/last-15-minutes
status	/monitor/status/last-15-minutes/status
status-value	/monitor/status/last-15-minutes/status-value
last-hour	/monitor/status/last-hour
status	/monitor/status/last-hour/status
status-value	/monitor/status/last-hour/status-value
last-24-hours	/monitor/status/last-24-hours

(Continued)

Item	YANG model path: /accounts/account/monitors ...
status	/monitor/status/last-24-hours/status
status-value	/monitor/status/last-24-hours/status-value
templates	/templates
template[name]	/templates/template
name	/templates/template/name
description	/templates/template/description
parameters	/templates/template/parameters
parameter[key]	/templates/template/parameters/parameter
key	/templates/template/parameters/parameter/key
type	/templates/template/parameters/parameter/type

Prerequisites for Monitor Orchestration

Before you can start a monitor through NETCONF using ncclient, you need to build a monitor template in the Control Center GUI as explained in the in-app help under "Tests and monitors" > "Creating templates". All fields specified as "Template input" in that template will be required as parameters in the XML when orchestrating the initiation of the template.

Getting Input Parameters from Monitor Templates

Below, two templates are shown. The first is for UDP monitoring between two Test Agent interfaces, and the second is for HTTP using a single Test Agent interface.

To find out the input parameters of a template, click the box representing the template. For the HTTP template, the parameters may look like this:

We need to define these parameters in the next step when starting a monitor.

Starting a Monitor

Using the Test Agents that we defined and deployed in the section ["Creating and Deploying a New Test Agent" on page 17](#), we can start a monitor from the template "HTTP" as shown below.

For each parameter, the `<key>` attribute needs to be supplied. The key is identical to the parameter's **Variable name** in Control Center. You can inspect variable names as follows:

- Click **Monitoring** on the side bar and select **New Monitor**.
- Click **My Templates**.
- Click the **Edit** link below the template of interest.
- Click the **Edit input** button in the top right corner.

In our example, and by default, the variable names are simply lowercase versions of the display names seen in Control Center ("url" vs. "URL", etc.). However, in the Control Center GUI, you can rename the variables to whatever you like.

Besides the key, each parameter needs to have its *type* specified: for example, `<string>` for the URL. Please note that full information on the parameter type is found in the YANG model. For Test Agent interfaces the type has a more complex structure, as evidenced in the code below.

In the example that follows, no *alarm* is associated with the monitor. For examples involving alarms, go to the section ["Starting a Monitor with an Alarm" on page 62](#).

```
with manager.connect(host=args.host, port=args.port, username=args.username,
                    password=args.password, hostkey_verify=False) as m:

# Apply configuration for a monitor, creating a monitor in Control Center
xml = """<config xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <accounts xmlns="http://ncc.netrounds.com">
    <account>
      <name>{account}</name>
      <monitors>
        <monitor>
          <name>{name}</name>
          <template>{template}</template>
          <parameters>
            <parameter>
              <key>url</key>
              <string>{url}</string>
            </parameter>
            <parameter>
              <key>clients</key>
              <test-agent-interfaces>
                <test-agent-interface>
                  <account>{account}</account>
                  <test-agent>{testagent}</test-agent>
                  <interface>{interface}</interface>
                  <ip-version>ipv4</ip-version>
                </test-agent-interface>
              </test-agent-interfaces>
            </parameter>
          </parameters>
        </monitor>
      </monitors>
    </account>
  </accounts>
</config>""".format(account=args.netrounds_account,
                    name='Web sites monitor',
                    template='HTTP',
                    testagent='vtal',
                    interface='eth0',
```



```

        url='http://example.com',server='vTA1',
    )

    monitor = m.edit_config(target='running', config=xml)

```

Run this script using ncclient. Provided that all is correct, the monitor will be initiated and started:

Web sites monitor				
[Click here to add a description]				
Using template "HTTP monitor" 				
<div> <div>● HTTP</div> <div></div> <div></div> </div>				
Client	ES history	Response time average (ms)	First byte received (ms)	Rate (Mbit/s)
VTA1:eth0 (IPv4)		198	198	0.051

Starting a Monitor with an Alarm

To associate an alarm with a monitor, you can either point to an alarm template that has been defined, or you can supply the entire alarm configuration when creating the monitor. We will give one example of each approach below.

Setting Up a Monitor Alarm by Pointing to an Alarm Template

In order to make use of an alarm template, you must know its ID. To this end, first retrieve all your alarm templates as described in the section ["Retrieving All Alarm Templates" on page 39](#) and note the name of the relevant template. You can then refer to that template as follows:

```
# Note: This example requires an existing SNMP manager, alarm email list and monitor template in order to work
```

```
with manager.connect(host=args.host, port=args.port, username=args.username,
password=args.password, hostkey_verify=False) as m:
```

```

# Apply configuration for a monitor, creating a monitor in Control Center
xml = """<config xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <accounts xmlns="http://ncc.netrounds.com">
        <account>
            <name>{account}</name>
            <monitors>

```



```

        <monitor>
            <name>{name}</name>
            <template>{template}</template>
            <alarm-configs>
                <alarm-config>
                    <identifier>{id}</identifier>
                    <template>{alarm_template}</template>
                    <thr-es-critical>{thr_es_critical}</thr-es-critical>
                </alarm-config>
            </alarm-configs>
        </monitor>
    </monitors>
</account>
</accounts>
</config>""".format(account=args.netrounds_account,
                      name='HTTP monitor',
                      template='name_of_monitor_template',
                      id='name_of_alarm_config',    # Enter a name for the alarm
configuration here
                      alarm_template='name_of_alarm_template',
                      thr_es_critical=20)          # The template value will be
overridden by this one

    print m.edit_config(target='running', config=xml)

```

Setting Up a Monitor Alarm by Configuring It Directly

Alternatively, you can set up an alarm for a monitor by supplying its entire configuration when creating the monitor, without referring to an alarm template. This is done as shown in the following example.

```

# Note: This example requires an existing SNMP manager, alarm email list and monitor template in
order to work

with manager.connect(host=args.host, port=args.port, username=args.username,
password=args.password, hostkey_verify=False) as m:

    # Apply configuration for a monitor, creating a monitor in Control Center
    xml = """<config xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
        <accounts xmlns="http://ncc.netrounds.com">
            <account>
                <name>{account}</name>

```

```

<monitors>
  <monitor>
    <name>{name}</name>
    <template>{template}</template>
    <parameters>
      <parameter>
        <key>url</key>
        <string>{url}</string>
      </parameter>
      <parameter>
        <key>clients</key>
        <test-agent-interfaces>
          <test-agent-interface>
            <account>{account}</account>
            <test-agent>{testagent}</test-agent>
            <interface>{interface}</interface>
            <ip-version>ipv4</ip-version>
          </test-agent-interface>
        </test-agent-interfaces>
      </parameter>
    </parameters>
    <alarm-configs>
      <alarm-config>
        <identifier>{id}</identifier>
        <email>{email}</email>
        <snmp>{snmp}</snmp>
        <action>{action}</action>
        <window-size>{window_size}</window-size>
        <interval>{interval}</interval>
        <send-only-once>{send_only_once}</send-only-once>
        <snmp-trap-per-stream>{snmp_trap_per_stream}</snmp-trap-
per-stream>
        <thr-es-critical>{thr_es_critical}</thr-es-critical>
        <thr-es-critical-clear>{thr_es_critical_clear}</thr-es-
critical-clear>
        <thr-es-major>{thr_es_major}</thr-es-major>
        <thr-es-major-clear>{thr_es_major_clear}</thr-es-major-
clear>
        <thr-es-minor>{thr_es_minor}</thr-es-minor>
        <thr-es-minor-clear>{thr_es_minor_clear}</thr-es-minor-
clear>
        <thr-es-warning>{thr_es_warning}</thr-es-warning>
        <thr-es-warning-clear>{thr_es_warning_clear}</thr-es-

```

```

warning-clear>
                                </alarm-config>
                            </alarm-configs>
                        </monitor>
                    </monitors>
                </account>
            </accounts>
        </config>""".format(account=args.netrounds_account,
                                name='HTTP monitor',
                                template='name_of_monitor_template',
                                testagent='vtal',
                                interface='eth0',
                                url='http://example.com',
                                id='alarm_config',
                                email='email_list',
                                snmp='snmp_manager',
                                action='some_action_string',
                                window_size=60,
                                interval=300,
                                send_only_once='false',
                                snmp_trap_per_stream='false',
                                thr_es_critical=9,
                                thr_es_critical_clear=8,
                                thr_es_major=7,
                                thr_es_major_clear=6,
                                thr_es_minor=5,
                                thr_es_minor_clear=4,
                                thr_es_warning=3,
                                thr_es_warning_clear=2)

print m.edit_config(target='running', config=xml)

```

Retrieving Running Monitors

To retrieve all monitors that are currently executing, run this script:

```

with manager.connect(host=args.host, port=args.port, username=args.username,
                    password=args.password, hostkey_verify=False) as m:

```

```

# Get configuration for running monitors in account
xml = """<accounts xmlns="http://ncc.netrounds.com">
    <account>
        <name>{account}</name>
        <monitors>
            <monitor>
            </monitor>
        </monitors>
    </account>
</accounts>""".format(account=args.netrounds_account)

monitor = m.get_config(source='running', filter=('subtree', xml)).data_xml

# Convert to ElementTree object
ele = to_ele(monitor)

# Convert back to XML string and print

print to_xml(ele, encoding='UTF-8', pretty_print=True)

```

The output is a list of all running monitors as shown below:

```

<?xml version="1.0" ?>
  <data xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
  xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
    <accounts xmlns="http://ncc.netrounds.com">
      <account>
        <name xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">demo</name>
        <monitors>
          <monitor>
            <name>Network Quality</name>
            <template>UDP</template>
            <parameters>
              <parameter>
                <key>clients</key>
                <test-agent-interfaces>
                  <test-agent-interface>
                    <account>demo</account>
                    <test-agent>vTA1</test-agent>
                    <interface>eth1</interface>
                    <ip-version>ipv4</ip-version>
                  </test-agent-interface>

```

```

</test-agent-interfaces>
</parameter>
<parameter>
  <key>server</key>
  <test-agent-interfaces>
    <test-agent-interface>
      <account>demo</account>
      <test-agent>vTA2</test-agent>
      <interface>eth1</interface>
      <ip-version>ipv4</ip-version>
    </test-agent-interface>
  </test-agent-interfaces>
</parameter>
</parameters>
</monitor>
</monitors>
</account>
</accounts>
</data>

```

Retrieving SLA Status for a Monitor

Here is how to retrieve the SLA status for a monitor. In this example, we are retrieving the SLA status for the monitor "Network Quality" for three intervals of time: the last 15 minutes, the last hour, and the last 24 hours.

```
with manager.connect(host=args.host, port=args.port, username=username,
                    password=args.password, hostkey_verify=False) as m:

    # Get status of monitor named "Network Quality" in account
    xml = ""<accounts xmlns="http://ncc.netrounds.com">
        <account>
            <name>{account}</name>
            <monitors>
                <monitor>
                    <name>Network Quality</name>
                    <status></status>
                </monitor>
            </monitors>
```

```

        </account>
    </accounts>""".format(account=args.netrounds_account)

monitor = m.get(filter=('subtree', xml)).data_xml

# Convert to ElementTree object
ele = to_ele(monitor)

# Convert back to XML string and print
print to_xml(ele, encoding='UTF-8', pretty_print=True)

```

The output will look something like this:

```

<?xml version="1.0" ?>
<data xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <accounts xmlns="http://ncc.netrounds.com">
    <account>
      <name xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">demo</name>
      <monitors>
        <monitor>
          <name xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">Network Quality</
name>
          <status>
            <id>27</id>
            <last-15-minutes>
              <status>good</status>
              <status-value>100.0</status-value>
            </last-15-minutes>
            <last-hour>
              <status>good</status>
              <status-value>100.0</status-value>
            </last-hour>
            <last-24-hours>
              <status>good</status>
              <status-value>100.0</status-value>
            </last-24-hours>
          </status>
        </monitor>
      </monitors>
    </account>

```

```
</accounts>
</data>
```

NETCONF Notifications

NETCONF notifications for monitors are triggered by SLA violations. These occur when the SLA for the monitor drops below an SLA threshold ("Good" or "Acceptable") within a given time window, by default the last 15 minutes. It should be noted that SLA violation notifications are quick to appear after a service is impacted by an issue, while the SLA status will revert to "Good" only after 15 minutes, and only if no further violations occur.

The time window can be changed by editing the setting `SLA_STATUS_WINDOW` (value in seconds) in `/etc/netrounds/netrounds.conf`.

Exporting and Importing Monitor Templates

This is done in exactly the same way as for test templates; compare the section ["Exporting and Importing Test Templates" on page 52](#). The code snippets below illustrate how to export and import templates for monitors.

Exporting Monitor Templates

```
with manager.connect(host=args.host, port=args.port, username=args.username,
                    password=args.password, hostkey_verify=False) as m:

    # Export selected templates from Control Center
    # In order to export all templates, remove the <templates></templates> node
    export_xml = ""
    <export-monitor-templates xmlns="http://ncc.netrounds.com">
        <account>{account}</account>
        <templates>
            <template>
                <name>template_a</name>
            </template>
        </templates>
```

```

        <name>template_b</name>
    </template>
</templates>
</export-monitor-templates>""".format(account=args.netrounds_account)

elem = to_ele(export_xml)
response = m.dispatch(elem)

# Get json config from response
root = ET.fromstring(response._raw)
json_config = root[0].text

print json_config

```

Importing Monitor Templates

```

with manager.connect(host=args.host, port=args.port, username=args.username,
                    password=args.password, hostkey_verify=False) as m:

    json_string = """
    {
        "templates":[
            {
                "input":[
                    {
                        "require_bridge":false,
                        # ...
                        # Insert rest of JSON string here
                        # ...
                    }
                ],
                "version":"2.29",
                "export_date":"2019-05-29T12:34:47.834430Z"
            }
        ]
    }
    """

    # Import templates back to Control Center
    import_xml = """
    <import-monitor-templates xmlns="http://ncc.netrounds.com">

```



```

        <account>{account}</account>
        <overwrite>false</overwrite>
        <data>{data}</data>
    </import-monitor-templates>""".format(account=args.netrounds_account,
                                           data=json_string)

    elem = to_ele(import_xml)
    response = m.dispatch(elem)

    print response

```

Using Tags

IN THIS SECTION

- [Creating Tags | 72](#)
- [Assigning a Tag | 72](#)
- [Updating a Tag | 75](#)
- [Unassigning a Tag | 76](#)
- [Deleting a Tag | 77](#)

Tags defined in Paragon Active Assurance can be applied to:

- monitors
- monitor templates
- Test Agents
- TWAMP reflectors
- Ping hosts.

For example, you can tag a monitor with the same tag as a subset of Test Agents that are going to run the monitor. This feature is particularly helpful if you have a large number of monitors and templates defined.

If you have set up an alarm with SNMP traps for a monitor, then the SNMP traps will be assigned the same tags as the monitor, if any.

Creating Tags

Below we show how to create a tag with name and color as defined by the XML <tag> substructure.

```
with manager.connect(host=args.host, port=args.port, username=args.username,
                    password=args.password, hostkey_verify=False) as m:

    xml = """
    <config xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
      <accounts xmlns="http://ncc.netrounds.com">
        <account>
          <name>{account}</name>
          <tags>
            <tag>
              <name>{tag}</name>
              <color>#00ff00</color>
            </tag>
          </tags>
        </account>
      </accounts>
    </config>""".format(account=args.netrounds_account,
                        tag=args.tag_name)

    monitor = m.edit_config(target='running', config=xml)

    print "ok" if monitor.ok else monitor.errors
```

Assigning a Tag

To assign a tag to a resource, you add it as a new <tag> element under the <tags> element for that resource.

Here is how to assign a tag to a Test Agent:

```
with manager.connect(host=args.host, port=args.port, username=args.username,
                    password=args.password, hostkey_verify=False) as m:

    xml = """
    <config xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
      <accounts xmlns="http://ncc.netrounds.com">
        <account>
          <name>{account}</name>
          <test-agents>
            <test-agent>
              <name>vta1</name>
              <tags>
                <tag>
                  <name>{tag}</name>
                </tag>
              </tags>
            </test-agent>
          </test-agents>
        </account>
      </accounts>
    </config>""".format(account=args.netrounds_account,
                        tag=args.tag_name)

    monitor = m.edit_config(target='running', config=xml, default_operation='merge')

    print "ok" if monitor.ok else monitor.errors
```

To assign a tag to a TWAMP reflector, do the following:

```
with manager.connect(host=args.host, port=args.port, username=args.username,
                    password=args.password, hostkey_verify=False) as m:

    xml = """
    <config xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
      <accounts xmlns="http://ncc.netrounds.com">
        <account>
          <name>{account}</name>
          <twamp-reflectors>
            <twamp-reflector>
```

```

        <name>{name}</name>
    </tags>
</twamp-reflector>
</twamp-reflectors>
</account>
</accounts>
</config>""".format(account=args.netrounds_account,
                    name='Name',
                    tag='Tag1')

monitor = m.edit_config(target='running', config=xml, default_operation='merge')

print "ok" if monitor.ok else monitor.errors

```

Assigning a tag to a monitor is handled similarly:

```

with manager.connect(host=args.host, port=args.port, username=args.username,
                    password=args.password, hostkey_verify=False) as m:

    xml = """
    <config xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
        <accounts xmlns="http://ncc.netrounds.com">
            <account>
                <name>{account}</name>
                <monitors>
                    <monitor>
                        <name>{name}</name>
                        <tags>
                            <tag>
                                <name>{tag}</name>
                            </tag>
                        </tags>
                    </monitor>
                </monitors>
            </account>
        </accounts>
    </config>""".format(account=args.netrounds_account,
                        name='HTTP monitor',

```

```

tag='Tag 1')

monitor = m.edit_config(target='running', config=xml, default_operation='merge')

print "ok" if monitor.ok else monitor.errors

```

Alternatively, you can assign an existing tag to any of these resource types when creating the resource, by including the <tags> element containing the tag in question.

Updating a Tag

Updating an existing tag with new attributes is analogous to creating a tag:

```

with manager.connect(host=args.host, port=args.port, username=args.username,
                    password=args.password, hostkey_verify=False) as m:

    xml = """
    <config xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
      <accounts xmlns="http://ncc.netrounds.com">
        <account>
          <name>{account}</name>
          <tags>
            <tag>
              <name>{tag}</name>
              <color>#ff0000</color>
            </tag>
          </tags>
        </account>
      </accounts>
    </config>""".format(account=args.netrounds_account, tag=args.tag_name)

    monitor = m.edit_config(target='running', config=xml, default_operation='merge')

    print "ok" if monitor.ok else monitor.errors

```

Unassigning a Tag

To unassign a tag from a resource, add the attribute `nc:operation="delete"` to the `<tag>` element belonging to the resource. Below, we unassign a tag from a monitor.

```
with manager.connect(host=args.host, port=args.port, username=args.username,
                    password=args.password, hostkey_verify=False) as m:

    xml = """
    <config xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
      <accounts xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0" xmlns="http://
ncc.netrounds.com">
        <account>
          <name>{account}</name>
          <monitors>
            <monitor>
              <name>{name}</name>
              <tags>
                <tag nc:operation="delete">
                  <name>{tag}</name>
                </tag>
              </tags>
            </monitor>
          </monitors>
        </account>
      </accounts>
    </config>""".format(account=args.netrounds_account,
                        name='HTTP monitor',
                        tag='Tag 1')

    monitor = m.edit_config(target='running', config=xml, default_operation='merge')

    print "ok" if monitor.ok else monitor.errors
```

Deleting a Tag

In order to delete a tag altogether from Control Center, the attribute `nc:operation="delete"` is again used, but this time applied to the tag itself, defined under `<account>`.

```
with manager.connect(host=args.host, port=args.port, username=args.username,
                    password=args.password, hostkey_verify=False) as m:

    xml = """
    <config xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
      <accounts xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0" xmlns="http://
ncc.netrounds.com">
        <account>
          <name>{account}</name>
          <tags>
            <tag nc:operation="delete">
              <name>{tag}</name>
            </tag>
          </tags>
        </account>
      </accounts>
    </config>""".format(account=args.netrounds_account, tag=args.tag_name)

    monitor = m.edit_config(target='running', config=xml)

    print "ok" if monitor.ok else monitor.errors
```

Troubleshooting

IN THIS SECTION

- [Problem: Orchestrator and Paragon Active Assurance Out of Sync | 78](#)
- [Problem: Initial Sync \(sync-from-ncc\) Failed Due to Unsupported Resources | 78](#)
- [Problem: NETCONF commands fail with ncclient.operations.rpc.RPCError: application communication failure | 79](#)

Problem: Orchestrator and Paragon Active Assurance Out of Sync

The orchestrator and Paragon Active Assurance can end up out of sync for example if configuration changes have been made in the Control Center GUI, or if applying a configuration was not successful and rolling back to the previous state failed.

In case of a failed rollback, the NETCONF server will no longer accept configuration changes; it will reply with an error message stating that configuration is locked until back in sync. To get back in sync and unlock configuration changes, you need to run the command

```
rpc sync-from-ncc
```

which synchronizes all configuration from Control Center to the configuration database.

NOTE: The `confd@netrounds.com` user (or whatever has been configured) must have superuser privileges for everything to be synced successfully. This can be achieved with the command

```
ncc user-update confd@netrounds.com --is-superuser
```

If the user is not a superuser, a warning will appear saying that not everything could be synced, but that all that could be handled has been.

NOTE: If your orchestrator also stores the configuration, you will need to re-synchronize that as well since the requested configuration (the configuration which the orchestrator expects Control Center to have) will not have been applied.

Problem: Initial Sync (sync-from-ncc) Failed Due to Unsupported Resources

If you try to run `rpc sync-from-ncc` on an account that has its configuration created in the Control Center GUI, you might run into problems if the account contains unsupported resources. It is recommended that you start with an empty account and do all configuration of it through NETCONF. Otherwise, if you encounter issues with resource conflicts, you will have to remove the conflicting resources from the account.

Problem: NETCONF commands fail with `ncclient.operations.rpc.RPCError: application communication failure`

The NETCONF server does not restore connectivity to the Control Center server automatically if Control Center is restarted. To restore the connection to Control Center, restart the NETCONF process:

```
sudo systemctl restart netrounds-confd
```

Notes on Test Agent Applications and Test Agent Appliances

IN THIS SECTION

- Test Agent Applications in ConfD | 79
- Empty Interface Configuration for Test Agent Appliance | 80
- YANG Schema Changes Regarding Undefined Interfaces | 80
- Limitations When Registering a Test Agent Created in ConfD | 80

Test Agent Applications in ConfD

Among Test Agents, the (newer) Test Agent *Application* works a bit differently from the (older) Test Agent *Appliance*.

Test Agent Applications do not currently support interface configuration. Therefore, the YANG schema allows specifying an *empty* interface configuration for such Test Agents. See ["this passage" on page 23](#) for an example.

When synchronizing the ConfD database with Control Center using the `sync-from-ncc` command, you want the interface configuration to remain empty and not to be overwritten with what is found in Control Center. Therefore you need to use a special flag `--without_interface_config` with that command when working with Test Agent Applications.

Empty Interface Configuration for Test Agent Appliance

As noted above, Test Agent *Application* does not support interface configuration, and it is therefore possible to omit interfaces in the YANG schema.

But there are also use cases where you might want to omit the interface configuration from a Test Agent *Appliance*. An example of this could be an orchestration scenario where you are spinning up a Test Agent using cloud-init, and you want the interface configuration from there to be used, instead of letting ConfD overwrite it as the Test Agent comes online.

YANG Schema Changes Regarding Undefined Interfaces

Since an empty interface configuration is now allowed (from version 2.34.0 onward), it is possible to specify any interface name as input to a *task* running as part of a test or monitor.

This is required to be able to use a Test Agent *Application*, since for these no interface names are defined in ConfD. Note, however, that this also means you can run into problems if by accident you configure a test or monitor to use a non-existing interface. So please be mindful of this.

Limitations When Registering a Test Agent Created in ConfD

When creating a Test Agent via the REST or NETCONF/YANG API, we cannot know beforehand which type it is: Test Agent Appliance or Test Agent Application. This becomes clear only after the Test Agent has registered.

Once the Test Agent has been registered and has turned into one of these concrete types, you are not allowed to re-register it as a different type of Test Agent. This means you are not allowed to first register it as a Test Agent Appliance, then re-register it as a Test Agent Application, or vice versa. If you need a Test Agent of a different type, you will need to create a new Test Agent.

Appendix: Tree Structure of Full YANG Model

IN THIS SECTION

- [Legend | 81](#)
- [YANG Model Tree Structure | 82](#)

In this appendix, the section "[Legend](#)" on [page 81](#) explains the syntax of the YANG model tree structure generated with the command `pyang -f tree`.

The section "[YANG Model Tree Structure](#)" on [page 82](#) gives the output from that command applied to `netrounds-ncc.yang`. Parts of this output are reproduced elsewhere in the document.

Legend

Each node is printed as:

`<status> <flags> <name> <opts> <type> <if-features>`

`<status>` is one of:

- + for current
- x for deprecated
- o for obsolete

`<flags>` is one of:

- rw for configuration data
- ro for non-configuration data
- x for rpcs and actions
- n for notifications

`<name>` is the name of the node

`(<name>)` means that the node is a choice node

`:(<name>)` means that the node is a case node

If the node is augmented into the tree from another module, its name is printed as `<prefix>:<name>`.

<opts> is one of:

- ? for an optional leaf, choice, anydata or anyxml
- ! for a presence container
- * for a leaf-list or list

YANG Model Tree Structure

```

module: netrounds-ncc
  +--rw statistics
    | +--ro users?                               int32
    | +--ro users-created-24h?                   int32
    | +--ro users-active-1h?                     int32
    | +--ro test-agent-appliance?                 int32
    | +--ro test-agent-appliance-online?         int32
    | +--ro test-agent-application?              int32
    | +--ro test-agent-application-online?       int32
    | +--ro monitors-started?                    int32
    | +--ro monitor-tasks-started?               int32
    | +--ro monitor-tasks-running-http?          int32
    | +--ro monitor-tasks-running-ping?          int32
    | +--ro monitor-tasks-running-dns?           int32
    | +--ro monitor-tasks-running-iptv?          int32
    | +--ro streams-active?                      int32
    | +--ro accounts?                            int32
    | +--ro accounts-created-24h?                int32
    | +--ro scheduled-calls-running?              int32
    | +--ro scheduled-call-latency?              decimal64
  +--rw accounts
    +--rw account* [name]
      +--rw name                                string
      +--rw statistics
        | +--ro users?                          int32
        | +--ro users-created-24h?              int32
        | +--ro users-active-1h?                int32
        | +--ro test-agent-appliance?            int32
        | +--ro test-agent-appliance-online?     int32
        | +--ro test-agent-application?          int32
        | +--ro test-agent-application-online?   int32
        | +--ro monitors-started?                int32

```

```

| +--ro monitor-tasks-started?      int32
| +--ro monitor-tasks-running-http? int32
| +--ro monitor-tasks-running-ping? int32
| +--ro monitor-tasks-running-dns?  int32
| +--ro monitor-tasks-running-iptv?  int32
| +--ro streams-active?              int32
+--rw test-agents
| +--rw test-agent* [name]
|   +--rw name          string
|   +--rw description?  string
|   +--rw tags
|   | +--rw tag* [name]
|   |   +--rw name      -> ../../../../tags/tag/name
|   +--ro state
|   | +--ro last-update?      yang:date-and-time
|   | +--ro status?          test-agent-status-t
|   | +--ro version?         string
|   | +--ro uptime?          int64
|   | +--ro cpu-usage?       percent-t
|   | +--ro memory-usage?    percent-t
|   | +--ro interface-config-error? string
|   | +--ro update-error?    string
|   | +--ro ntp
|   | | +--ro client
|   | | | +--ro time-offset? decimal64
|   | | | +--ro sources* [address]
|   | |   +--ro address      string
|   | |   +--ro status?      string
|   | |   +--ro stratum?     int64
|   | |   +--ro poll-interval? int64
|   | |   +--ro reach?       string
|   | |   +--ro last-rx?     int64
|   | +--ro load-average
|   |   +--ro last-1-minute?  decimal64
|   |   +--ro last-5-minutes? decimal64
|   |   +--ro last-15-minutes? decimal64
|   +--rw ntp
|   | +--rw interface?      string
|   | +--rw servers* [address]
|   | | +--rw address      union
|   | +--rw enable-ipv6?    boolean
|   +--rw management
|   | +--rw interface?      string

```

```

|   | +--rw use-public-address?  boolean
|   +--rw interfaces
|   | +--rw interface* [name]
|   |   +--rw name                string
|   |   +--rw description?        string
|   |   +--rw interface-type?     interface-type-t
|   |   +--rw mac?                yang:mac-address
|   |   +--rw mtu?                uint16
|   |   +--rw speed?              speed-t
|   |   +--rw physical
|   |   | +--rw auth
|   |   |   +--rw (auth-type)?
|   |   |   +---:(eap-tls)
|   |   |   | +--rw eap-tls!
|   |   |   |   +--rw ca-cert?      string
|   |   |   |   +--rw client-cert?  string
|   |   |   |   +--rw identity      string
|   |   |   |   +--rw private-key?  string
|   |   |   +---:(eap-ttls)
|   |   |   | +--rw eap-ttls!
|   |   |   |   +--rw ca-cert?      string
|   |   |   |   +--rw identity      string
|   |   |   |   +--rw anonymous-identity?  string
|   |   |   |   +--rw password      string
|   |   |   +---:(peap)
|   |   |   | +--rw peap!
|   |   |   |   +--rw ca-cert?      string
|   |   |   |   +--rw identity      string
|   |   |   |   +--rw anonymous-identity?  string
|   |   |   |   +--rw password      string
|   |   +--rw bridge
|   |   | +--rw members
|   |   |   +--rw member* [interface]
|   |   |       +--rw interface    -> ../../../../interface/name
|   |   +--rw vlan
|   |   | +--rw id                vlan-id-t
|   |   | +--rw parent            -> ../../../../interface/name
|   |   +--rw wifi
|   |   | +--rw ssid?             string
|   |   | +--rw bssid?            yang:mac-address
|   |   | +--rw country?          string
|   |   | +--rw ht                boolean
|   |   | +--rw ht40              boolean

```

```

| | | +--rw vht          boolean
| | | +--rw sgi          boolean
| | | +--rw ldpc         boolean
| | | +--rw ht-mcs?      string
| | | +--rw vht-max-mcs  uint8
| | | +--rw vht-max-mimo uint8
| | | +--rw freq-2-4     boolean
| | | +--rw freq-5       boolean
| | | +--rw auth
| | |   +--rw (auth-type)?
| | |     +--:(personal)
| | |       | +--rw personal!
| | |       |   +--rw cipher      cipher-auth-t
| | |       |   +--rw password    string
| | |       +--:(eap-tls)
| | |         | +--rw eap-tls!
| | |         |   +--rw cipher      cipher-auth-t
| | |         |   +--rw ca-cert?    string
| | |         |   +--rw client-cert? string
| | |         |   +--rw identity    string
| | |         |   +--rw private-key? string
| | |         +--:(eap-ttls)
| | |           | +--rw eap-ttls!
| | |           |   +--rw cipher      cipher-auth-t
| | |           |   +--rw ca-cert?    string
| | |           |   +--rw identity    string
| | |           |   +--rw anonymous-identity? string
| | |           |   +--rw password    string
| | |           +--:(peap)
| | |             +--rw peap!
| | |             |   +--rw cipher      cipher-auth-t
| | |             |   +--rw ca-cert?    string
| | |             |   +--rw identity    string
| | |             |   +--rw anonymous-identity? string
| | |             |   +--rw password    string
| | +--rw mobile
| |   | +--rw apn?      string
| |   | +--rw pdp_type  string
| |   | +--rw rat_band  mobile-rat-band-t
| |   | +--rw rat_mode  mobile-rat-mode-t
| |   +--rw interface-config
| |     +--rw ipv4-address
| |     | +--rw (address-type)?

```

```

| | | +--:(static)
| | | | +--rw static!
| | | | +--rw address tailf:ipv4-address-and-prefix-length
| | | | +--rw routes* [network]
| | | | | +--rw network tailf:ipv4-address-and-prefix-length
| | | | | +--rw gateway inet:ipv4-address
| | | | +--rw dns-servers* inet:ipv4-address
| | | +--:(dhcp)
| | | | +--rw dhcp!
| | | | +--rw vendor-id? string
| | | +--rw ipv6-address
| | | | +--rw (address-type)?
| | | | +--:(static)
| | | | | +--rw static!
| | | | | +--rw address tailf:ipv6-address-and-prefix-length
| | | | | +--rw routes* [network]
| | | | | | +--rw network tailf:ipv6-address-and-prefix-length
| | | | | | +--rw gateway inet:ipv6-address
| | | | | +--rw dns-servers* inet:ipv6-address
| | | +--:(dhcp)
| | | | +--rw dhcp!
| | | | +--rw vendor-id? string
| | | +--:(slaac)
| | | | +--rw slaac!
| | | | +--rw dns-servers* inet:ipv6-address
| +--rw ssh-keys
| | +--rw ssh-key* [name]
| | | +--rw name string
| | | +--rw ssh-key-value string
| | | +--ro test-agent? -> /accounts/account/test-agents/test-agent/name
+--ro registered-test-agents
| +--ro test-agent* [name]
| | +--ro name string
| | +--ro description? string
| | +--ro tags
| | | +--ro tag* [name]
| | | | +--ro name -> ../../../../tags/tag/name
| | +--ro state
| | | +--ro last-update? yang:date-and-time
| | | +--ro status? test-agent-status-t
| | | +--ro version? string
| | | +--ro uptime? int64
| | | +--ro cpu-usage? percent-t

```



```

|   | +--ro memory-usage?          percent-t
|   | +--ro interface-config-error? string
|   | +--ro update-error?          string
|   | +--ro ntp
|   | | +--ro client
|   | | | +--ro time-offset?    decimal64
|   | | | +--ro sources* [address]
|   | |   +--ro address          string
|   | |   +--ro status?          string
|   | |   +--ro stratum?         int64
|   | |   +--ro poll-interval?  int64
|   | |   +--ro reach?           string
|   | |   +--ro last-rx?         int64
|   | +--ro load-average
|   |   +--ro last-1-minute?     decimal64
|   |   +--ro last-5-minutes?    decimal64
|   |   +--ro last-15-minutes?   decimal64
| +--ro ntp
|   | +--ro interface?          string
|   | +--ro servers* [address]
|   | | +--ro address          union
|   | +--ro enable-ipv6?        boolean
| +--ro management
|   | +--ro interface?          string
|   | +--ro use-public-address?  boolean
| +--ro interfaces
|   | +--ro interface* [name]
|   |   +--ro name              string
|   |   +--ro description?      string
|   |   +--ro interface-type?   interface-type-t
|   |   +--ro mac?              yang:mac-address
|   |   +--ro mtu?              uint16
|   |   +--ro speed?            speed-t
|   |   +--ro physical
|   |   | +--ro auth
|   |   |   +--ro (auth-type)?
|   |   |     +--:(eap-tls)
|   |   |       | +--ro eap-tls!
|   |   |       |   +--ro ca-cert?    string
|   |   |       |   +--ro client-cert? string
|   |   |       |   +--ro identity    string
|   |   |       |   +--ro private-key? string
|   |   |       +--:(eap-ttls)

```

```

|   |   |   |   +---ro eap-ttls!
|   |   |   |   +---ro ca-cert?           string
|   |   |   |   +---ro identity            string
|   |   |   |   +---ro anonymous-identity? string
|   |   |   |   +---ro password            string
|   |   |   +---:(peap)
|   |   |       +---ro peap!
|   |   |       +---ro ca-cert?           string
|   |   |       +---ro identity            string
|   |   |       +---ro anonymous-identity? string
|   |   |       +---ro password            string
|   |   +---ro bridge
|   |       | +---ro members
|   |       |   +---ro member* [interface]
|   |       |       +---ro interface  -> ../../../../interface/name
|   |   +---ro vlan
|   |       | +---ro id          vlan-id-t
|   |       | +---ro parent      -> ../../../../interface/name
|   |   +---ro wifi
|   |       | +---ro ssid?        string
|   |       | +---ro bssid?       yang:mac-address
|   |       | +---ro country?     string
|   |       | +---ro ht           boolean
|   |       | +---ro ht40         boolean
|   |       | +---ro vht          boolean
|   |       | +---ro sgi          boolean
|   |       | +---ro ldpc         boolean
|   |       | +---ro ht-mcs?      string
|   |       | +---ro vht-max-mcs  uint8
|   |       | +---ro vht-max-mimo uint8
|   |       | +---ro freq-2-4     boolean
|   |       | +---ro freq-5       boolean
|   |       | +---ro auth
|   |       |   +---ro (auth-type)?
|   |       |       +---:(personal)
|   |       |       | +---ro personal!
|   |       |       |   +---ro cipher      cipher-auth-t
|   |       |       |   +---ro password    string
|   |       |       +---:(eap-tls)
|   |       |       | +---ro eap-tls!
|   |       |       |   +---ro cipher      cipher-auth-t
|   |       |       |   +---ro ca-cert?    string
|   |       |       |   +---ro client-cert? string

```

```

| | | | +--ro identity      string
| | | | +--ro private-key?  string
| | | | +---:(eap-ttls)
| | | | | +--ro eap-ttls!
| | | | | +--ro cipher          cipher-auth-t
| | | | | +--ro ca-cert?       string
| | | | | +--ro identity       string
| | | | | +--ro anonymous-identity? string
| | | | | +--ro password       string
| | | | +---:(peap)
| | | | | +--ro peap!
| | | | | +--ro cipher          cipher-auth-t
| | | | | +--ro ca-cert?       string
| | | | | +--ro identity       string
| | | | | +--ro anonymous-identity? string
| | | | | +--ro password       string
| | | +--ro mobile
| | | | +--ro apn?            string
| | | | +--ro pdp_type       string
| | | | +--ro rat_band       mobile-rat-band-t
| | | | +--ro rat_mode       mobile-rat-mode-t
| | | +--ro interface-config
| | | | +--ro ipv4-address
| | | | | +--ro (address-type)?
| | | | | +---:(static)
| | | | | | +--ro static!
| | | | | | +--ro address      tailf:ipv4-address-and-prefix-length
| | | | | | +--ro routes* [network]
| | | | | | | +--ro network    tailf:ipv4-address-and-prefix-length
| | | | | | | +--ro gateway    inet:ipv4-address
| | | | | | +--ro dns-servers* inet:ipv4-address
| | | | | +---:(dhcp)
| | | | | | +--ro dhcp!
| | | | | | +--ro vendor-id?  string
| | | +--ro ipv6-address
| | | | +--ro (address-type)?
| | | | +---:(static)
| | | | | +--ro static!
| | | | | +--ro address      tailf:ipv6-address-and-prefix-length
| | | | | +--ro routes* [network]
| | | | | | +--ro network    tailf:ipv6-address-and-prefix-length
| | | | | | +--ro gateway    inet:ipv6-address
| | | | | +--ro dns-servers* inet:ipv6-address

```

```

|   |           +---:(dhcp)
|   |           | +---ro dhcp!
|   |           |   +---ro vendor-id?  string
|   |           +---:(slaac)
|   |           +---ro slaac!
|   |           +---ro dns-servers*  inet:ipv6-address
|   +---ro ssh-keys
|       +---ro ssh-key* [name]
|           +---ro name          string
|           +---ro ssh-key-value  string
|           +---ro test-agent?    -> /accounts/account/test-agents/test-agent/name
+---rw tags
| +---rw tag* [name]
|     +---rw name      string
|     +---rw color?    string
+---ro tests
| +---ro test* [id]
| | +---ro id          uint32
| | +---ro name?       string
| | +---ro status?     test-status-t
| | +---ro start-time? yang:date-and-time
| | +---ro end-time?   yang:date-and-time
| | +---ro report-url?  string
| | +---ro report-url-pdf? string
| | +---ro steps
| |     +---ro step* [id]
| |         +---ro id          uint32
| |         +---ro name?       string
| |         +---ro start-time? yang:date-and-time
| |         +---ro end-time?   yang:date-and-time
| |         +---ro status?     test-status-t
| |         +---ro status-message? string
| +---ro templates
|     +---ro template* [name]
|         +---ro name          string
|         +---ro description?  string
|         +---ro tags?         string
|         +---ro parameters
|             +---ro parameter* [key]
|                 +---ro key      string
|                 +---ro type?    parameter-type-t
+---rw twamp-reflectors
| +---rw twamp-reflector* [name]

```

```

|   +--rw name          string
|   +--rw address       string
|   +--rw ipv6?         boolean
|   +--rw port          uint16
|   +--rw control-port? uint16
|   +--rw gps-lat?      gps-lat-t
|   +--rw gps-long?     gps-long-t
|   +--rw tags
|       +--rw tag* [name]
|           +--rw name    -> ../../../../tags/tag/name
+--rw ping-hosts
|   +--rw ping-host* [name]
|       +--rw name      string
|       +--rw host      string
|       +--rw ipv6?     boolean
|       +--rw gps-lat?  gps-lat-t
|       +--rw gps-long? gps-long-t
|       +--rw tags
|           +--rw tag* [name]
|               +--rw name    -> ../../../../tags/tag/name
+--rw y1731-meps
|   +--rw y1731-mep* [name]
|       +--rw name      string
|       +--rw mac       yang:mac-address
|       +--rw meg-level  uint8
+--rw iptv-channels
|   +--rw iptv-channel* [name]
|       +--rw name      string
|       +--rw ip        string
|       +--rw source?   string
|       +--rw port      uint16
|       +--rw pnun?     uint16
+--rw sip-accounts
|   +--rw sip-account* [sip-address]
|       +--rw sip-address  ascii-string-t
|       +--rw registrar?  ascii-string-t
|       +--rw password    ascii-string-t
|       +--rw proxy?      ascii-string-t
|       +--rw user-auth?  ascii-string-t
|       +--rw uri-rewrite? ascii-string-t
+--rw monitors
|   +--rw monitor* [name]
|       +--rw name      string

```

```

| | +--rw description?    string
| | +--rw started?        boolean
| | +--rw template        string
| | +--rw tags
| | | +--rw tag* [name]
| | |   +--rw name      -> ../../../../tags/tag/name
| | +--rw alarm-configs
| | | +--rw alarm-config* [identifier]
| | |   +--rw identifier      string
| | |   +--rw template?      -> ../../../../alarm-templates/alarm-template/name
| | |   +--rw email?         -> ../../../../alarm-email-lists/alarm-email-list/name
| | |   +--rw snmp?          -> ../../../../snmp-managers/snmp-manager/name
| | |   +--rw thr-es-critical? uint16
| | |   +--rw thr-es-critical-clear? uint16
| | |   +--rw thr-es-major?   uint16
| | |   +--rw thr-es-major-clear? uint16
| | |   +--rw thr-es-minor?   uint16
| | |   +--rw thr-es-minor-clear? uint16
| | |   +--rw thr-es-warning? uint16
| | |   +--rw thr-es-warning-clear? uint16
| | |   +--rw no-data-severity? uint16
| | |   +--rw no-data-timeout? uint16
| | |   +--rw action?        string
| | |   +--rw window-size?   uint16
| | |   +--rw interval?      uint16
| | |   +--rw send-only-once? boolean
| | |   +--rw snmp-trap-per-stream? boolean
| | +--rw parameters
| | | +--rw parameter* [key]
| | |   +--rw key                string
| | |   +--rw (value-type)
| | |     +--:(integer)
| | |       | +--rw integer?      int64
| | |     +--:(float)
| | |       | +--rw float?        decimal64
| | |     +--:(string)
| | |       | +--rw string?       string
| | |     +--:(test-agent-interfaces)
| | |       | +--rw test-agent-interfaces
| | |       |   +--rw test-agent-interface* [account test-agent interface]
| | |       |     +--rw account    -> ../../../../name
| | |       |     +--rw test-agent -> ../../../../test-agents/test-agent/name
| | |       |     +--rw interface  string

```

```

| | | | +--rw ip-version? inet:ip-version
| | | +--:(twamp-reflectors)
| | | | +--rw twamp-reflectors
| | | | +--rw twamp-reflector* [name]
| | | | +--rw name -> ../../../../../../twamp-reflectors/twamp-reflector/name
| | | +--:(ping-hosts)
| | | | +--rw ping-hosts
| | | | +--rw ping-host* [name]
| | | | +--rw name -> ../../../../../../ping-hosts/ping-host/name
| | | +--:(y1731-meps)
| | | | +--rw y1731-meps
| | | | +--rw y1731-mep* [name]
| | | | +--rw name -> ../../../../../../y1731-meps/y1731-mep/name
| | | +--:(sip-accounts)
| | | | +--rw sip-accounts
| | | | +--rw sip-account* [account test-agent interface sip-address]
| | | | +--rw account -> ../../../../../../name
| | | | +--rw test-agent -> ../../../../../../test-agents/test-agent/name
| | | | +--rw interface -> deref(..test-agent)/../interfaces/interface/name
| | | | +--rw sip-address -> ../../../../../../sip-accounts/sip-account/sip-
address
| | | +--:(iptv-channels)
| | | | +--rw iptv-channels
| | | | +--rw iptv-channel* [name]
| | | | +--rw name -> ../../../../../../iptv-channels/iptv-channel/name
| | +--ro status
| | +--ro id? uint32
| | +--ro last-15-minutes
| | | +--ro status? monitor-sla-status-t
| | | +--ro status-value? percent-t
| | +--ro last-hour
| | | +--ro status? monitor-sla-status-t
| | | +--ro status-value? percent-t
| | +--ro last-24-hours
| | +--ro status? monitor-sla-status-t
| | +--ro status-value? percent-t
| +--ro templates
| +--ro template* [name]
| +--ro name string
| +--ro description? string
| +--ro parameters
| | +--ro parameter* [key]
| | +--ro key string

```

```

|      |      +--ro type?   parameter-type-t
|      +--ro tags?         string
+--rw alarm-templates
| +--rw alarm-template* [name]
|   +--rw name              string
|   +--rw email?            -> ../../../../alarm-email-lists/alarm-email-list/name
|   +--rw snmp?             -> ../../../../snmp-managers/snmp-manager/name
|   +--rw thr-es-critical?   uint16
|   +--rw thr-es-critical-clear? uint16
|   +--rw thr-es-major?      uint16
|   +--rw thr-es-major-clear? uint16
|   +--rw thr-es-minor?      uint16
|   +--rw thr-es-minor-clear? uint16
|   +--rw thr-es-warning?    uint16
|   +--rw thr-es-warning-clear? uint16
|   +--rw no-data-severity?   uint16
|   +--rw no-data-timeout?    uint16
|   +--rw action?            string
|   +--rw window-size?       uint16
|   +--rw interval?          uint16
|   +--rw send-only-once?     boolean
|   +--rw snmp-trap-per-stream? boolean
+--rw alarm-email-lists
| +--rw alarm-email-list* [name]
|   +--rw name              string
|   +--rw addresses*        string
+--rw snmp-managers
  +--rw snmp-manager* [name]
    +--rw name              string
    +--rw ip?               string
    +--rw port?              uint16
    +--rw version?           string
    +--rw community?         string
    +--rw engine-id?         string
    +--rw user-name?         string
    +--rw security?          snmp-security-t
    +--rw auth-password?     string
    +--rw priv-password?     string

```

rpcs:

```

+---x start-test
| +---w input
| | +---w name          string

```



```

| | +---w account      -> /accounts/account/name
| | +---w template    string
| | +---w parameters
| |   +---w parameter* [key]
| |     +---w key                string
| |     +---w (value-type)
| |       +---:(integer)
| |         | +---w integer?          int64
| |         +---:(float)
| |           | +---w float?          decimal64
| |           +---:(string)
| |             | +---w string?        string
| |             +---:(test-agent-interfaces)
| |               | +---w test-agent-interfaces
| |                 | +---w test-agent-interface* [account test-agent interface]
| |                   | +---w account      -> ../../../../account
| |                   | +---w test-agent    -> deref ../../../../account)/../test-agents/test-agent/
name
| |   | +---w interface    string
| |   | +---w ip-version?  inet:ip-version
| |   +---:(twamp-reflectors)
| |     | +---w twamp-reflectors
| |     | +---w twamp-reflector* [name]
| |     | +---w name      -> deref ../../../../account)/../twamp-reflectors/twamp-
reflector/name
| |       +---:(ping-hosts)
| |       | +---w ping-hosts
| |       |   +---w ping-host* [name]
| |       |     +---w name      -> deref ../../../../account)/../ping-hosts/ping-host/name
| |       +---:(y1731-meps)
| |       | +---w y1731-meps
| |       |   +---w y1731-mep* [name]
| |       |     +---w name      -> deref ../../../../account)/../y1731-meps/y1731-mep/name
| |       +---:(iptv-channels)
| |       | +---w iptv-channels
| |       |   +---w iptv-channel* [name]
| |       |     +---w name      -> deref ../../../../account)/../iptv-channels/iptv-channel/
name
| |       +---:(sip-accounts)
| |       | +---w sip-accounts
| |       |   +---w sip-account* [account test-agent interface sip-address]
| |       |     +---w account      -> ../../../../account
| |       |     +---w test-agent    -> deref ../../../../account)/../test-agents/test-agent/

```

```

name
| |
| |      +---w interface      -> deref(..test-agent)/../interfaces/interface/name
| |      +---w sip-address    -> deref(..../..../account)/../sip-accounts/sip-
account/sip-address
| +--ro output
|   +---ro result      test-status-t
|   +---ro id?         uint32
|   +---ro error-text?  string
+---x sync-from-ncc
| +---w input
| | +---w without_interface_config?  boolean
| +--ro output
|   +---ro success      boolean
|   +---ro error-text?  string
+---x reboot-test-agents
| +---w input
| | +---w test-agents
| |   +---w (all-or-list)
| |     +---:(test-agent)
| |       | +---w test-agent* [account name]
| |       |   +---w account  -> /accounts/account/name
| |       |   +---w name     -> deref(..account)/../test-agents/test-agent/name
| |       +---:(all-in-account)
| |         +---w all-in-account?  -> /accounts/account/name
| +--ro output
|   +---ro test-agents-failed* [account name]
| | +---ro account  -> /accounts/account/name
| | +---ro name     -> /accounts/account/test-agents/test-agent/name
| | +---ro reason?  string
|   +---ro error-text?      string
+---x update-test-agents
| +---w input
| | +---w test-agents
| |   +---w (all-or-list)
| |     +---:(test-agent)
| |       | +---w test-agent* [account name]
| |       |   +---w account  -> /accounts/account/name
| |       |   +---w name     -> deref(..account)/../test-agents/test-agent/name
| |       +---:(all-in-account)
| |         +---w all-in-account?  -> /accounts/account/name
| +--ro output
|   +---ro test-agents-failed* [account name]
| | +---ro account  -> /accounts/account/name

```

```

|   | +--ro name      -> /accounts/account/test-agents/test-agent/name
|   | +--ro reason?   string
|   +--ro error-text? string
+---x sync-test-agents-stats
| +---w input
| | +---w test-agents
| |   +---w (all-or-list)
| |     +---:(test-agent)
| |       | +---w test-agent* [account name]
| |       |   +---w account  -> /accounts/account/name
| |       |   +---w name     -> deref(..../account)/../test-agents/test-agent/name
| |       +---:(all-in-account)
| |         +---w all-in-account? -> /accounts/account/name
| +--ro output
|   +--ro test-agents-failed* [account name]
|   | +--ro account  -> /accounts/account/name
|   | +--ro name     -> /accounts/account/test-agents/test-agent/name
|   | +--ro reason?  string
|   +--ro error-text? string
+---x export-test-templates
| +---w input
| | +---w account  -> /accounts/account/name
| | +---w templates
| |   +---w template* [name]
| |     +---w name  -> deref(..../account)/../tests/templates/template/name
| +--ro output
|   +--ro data      string
|   +--ro warnings  string
|   +--ro error-text? string
+---x export-monitor-templates
| +---w input
| | +---w account  -> /accounts/account/name
| | +---w templates
| |   +---w template* [name]
| |     +---w name  -> deref(..../account)/../monitors/templates/template/name
| +--ro output
|   +--ro data      string
|   +--ro warnings  string
|   +--ro error-text? string
+---x import-test-templates
| +---w input
| | +---w account  -> /accounts/account/name
| | +---w overwrite? boolean

```

```

| | +---w data      string
| +---ro output
|   +---ro warnings    string
|   +---ro error-text? string
+---x import-monitor-templates
    +---w input
    | +---w account    -> /accounts/account/name
    | +---w overwrite? boolean
    | +---w data      string
    +---ro output
        +---ro warnings    string
        +---ro error-text? string

notifications:
+---n test-agent-status-change
| +---ro account    -> /accounts/account/name
| +---ro test-agent -> /accounts/account/test-agents/test-agent/name
| +---ro status     test-agent-status-t
| +---ro tags?      string
+---n test-status-change
| +---ro account    -> /accounts/account/name
| +---ro id         -> /accounts/account/tests/test/id
| +---ro status     test-status-t
| +---ro finished?  boolean
+---n monitor-sla-status-change
    +---ro account    -> /accounts/account/name
    +---ro monitor     -> /accounts/account/monitors/monitor/name
    +---ro status     monitor-sla-status-t
    +---ro prev-status monitor-sla-status-t
    +---ro url        string
    +---ro tags?      string

```

Juniper Networks, the Juniper Networks logo, Juniper, and Junos are registered trademarks of Juniper Networks, Inc. in the United States and other countries. All other trademarks, service marks, registered marks, or registered service marks are the property of their respective owners. Juniper Networks assumes no responsibility for any inaccuracies in this document. Juniper Networks reserves the right to change, modify, transfer, or otherwise revise this publication without notice. Copyright © 2022 Juniper Networks, Inc. All rights reserved.