



TELECOM INFRA PROJECT

# MUST Optical SDN Controller SBI Technical Requirement for Open Terminals

*TIP OOPT PG - Version: 1.0*



Copyright © 2021 Telecom Infra Project, Inc. All rights reserved.

The Telecom Infra Project logo is a trademark of Telecom Infra Project, Inc. (the “Project”) in the United States or other countries and is registered in one or more countries. Removal of any of the notices or disclaimers contained in this document is strictly prohibited.

The publication of this Specification is for informational purposes only. THIS SPECIFICATION IS PROVIDED “AS IS,” AND WITHOUT ANY WARRANTY OF ANY KIND, INCLUDING WITHOUT LIMITATION, ANY EXPRESS OR IMPLIED WARRANTY OF NONINFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. UNDER NO CIRCUMSTANCES WILL THE PROJECT BE LIABLE TO ANY PARTY UNDER ANY CONTRACT, STRICT LIABILITY, NEGLIGENCE OR OTHER LEGAL OR EQUITABLE THEORY, FOR ANY INCIDENTAL INDIRECT, SPECIAL, EXEMPLARY, PUNITIVE, OR CONSEQUENTIAL DAMAGES OR FOR ANY COMMERCIAL OR ECONOMIC LOSSES, WITHOUT LIMITATION, INCLUDING AS A RESULT OF PRODUCT LIABILITY CLAIMS, LOST PROFITS, SAVINGS OR REVENUES OF ANY KIND IN CONNECTION WITH THE USE OR IMPLEMENTATION OF THIS SPECIFICATION.

TIP does not own or control patents. Contributors to this Specification, as defined in the TIP IPR Policy, have undertaken patent licensing obligations as set forth in TIP’s IPR Policy which can be found at [here](#).

#### TIP CONFIDENTIAL

This document contains TIP Confidential Information as defined in Section 1.3 of the TIP Bylaws. Subject to Section 16.2 of the TIP Bylaws, use and disclosure of the document and its contents are strictly prohibited.



## Authors:

- **Alfredo Gonzalez**
  - o Telefonica CTIO
  - o [alfredo.gonzalezmuniz.ext@telefonica.com](mailto:alfredo.gonzalezmuniz.ext@telefonica.com)
- **Arturo Mayoral**
  - o Technology Expert, Telefonica CTIO
  - o [Arturo.mayoral@telefonica.com](mailto:Arturo.mayoral@telefonica.com)
- **Victor López**
  - o Technological Expert; IP & Transport Networks; Telefonica CTIO
  - o [victor.lopezalvarez@telefonica.com](mailto:victor.lopezalvarez@telefonica.com)
- **Esther Le Rouzic**
  - o Research engineer, Orange
  - o [esther.lerouzic@orange.com](mailto:esther.lerouzic@orange.com)
- **Jean-Francois Bouquier**
  - o Optical and SDN Network Architect, Vodafone
  - o [jeff.bouquier@vodafone.com](mailto:jeff.bouquier@vodafone.com)
- **Dirk Breuer**
  - o Technology Architecture & Innovation, Deutsche Telekom AG.
  - o [d.breuer@telekom.de](mailto:d.breuer@telekom.de)
- **Anders Lindgren**
  - o Network Architect, Telia Company
  - o [anders.x.lindgren@teliacompany.com](mailto:anders.x.lindgren@teliacompany.com)
- **Stefan Melin**
  - o Network Architect, Telia Company
  - o [Stefan.Melin@teliacompany.com](mailto:Stefan.Melin@teliacompany.com)
- **Luc-Fabrice Ndifor**
  - o Fixed Networks & Technology Management, MTN
  - o [luc-fabrice.ndifor@mtn.com](mailto:luc-fabrice.ndifor@mtn.com)
- **Steven Hill**
  - o Transport Manager : IP Access, Optical and Fiber, MTN
  - o [steven.hill@mtn.com](mailto:steven.hill@mtn.com)



Change Tracking

Date	Revision	Author(s)	Comment
25/03/2021	V1.0	All	Consolidated version 1.0



Table of Contents

1 Introduction..... 8

1.1 Goal ..... 11

1.2 Scope of the document ..... 12

1.3 Document Structure..... 12

1.4 Release notes ..... 12

2 Proccotol considerations..... 13

2.1 Netconf..... 13

2.1.1 Data encoding ..... 13

2.2 gRPC/gNMI..... 14

2.2.1 Data encoding ..... 14

3 OpenConfig considerations..... 14

3.1 OpenConfig SDK version and documentation ..... 14

3.2 OpenConfig Information model ..... 14

4 Initial state of device (commissioning stage) ..... 16

5 OpenConfig atomic operations. .... 17

5.1 Discovery operations..... 17

5.1.1 [Discovery-1] Discovery of all components ..... 19

5.1.2 [Discovery-2] Discovery of type of component..... 21

5.1.3 [Discovery-3] Discovery of line and client ports and subcomponents ..... 23

5.1.4 [Discovery-4] Discovery of operational modes available and capabilities using operational-mode-augment (Annex A) ..... 27

5.1.5 [Discovery-5] Discovery of admin status of optical port ..... 32

5.1.6 [Discovery-5b] Discovery of admin status of interface ..... 33

5.1.7 [Discovery-6] Discovery of client port capabilities (protocol, bit rate and mapping). ..... 33

5.1.8 [Discovery-7] Discovery of line port capabilities (protocol, bit rate, available granularity and allocation)..... 37

5.1.9 [Discovery-8] Discovery of frequency, target-out-power and operational mode configured in line port 38

5.1.10 [Discovery-9] Discovery of type of logical channel assignment (preset or flexible)..... 39

5.1.11 [Discovery-10] Discovery of logical channel-assignment ..... 40

5.1.12 [Discovery-11] Discovery of location of a component and mapping to INVENTORY\_ID (TAPI) 42

5.1.13 [Discovery-12] Retrieval of global structure of HW inventory ..... 45

5.1.14 [Discovery-13] Retrieval of HW inventory information of each component ..... 50

5.1.15 [Discovery-14] Slot-id information ..... 53

5.2 Provisioning in devices with preset logical channels ..... 54

5.2.1 [Provisioning-1] Provisioning of admin-state enable of client port..... 54

5.2.2 [Provisioning-2] Provisioning of admin-state enable of line port ..... 56

5.2.3 [Provisioning-3] Provisioning of frequency in line port ..... 57

5.2.4 [Provisioning-4] Provisioning of optical output power in line port. .... 57

5.2.5 [Provisioning-5] Provisioning of operational mode in port line ..... 59

5.3 Provisioning in devices with flexible/configurable logical channels (no preset) ..... 60

5.3.1 [Provisionig-6] Provisioning in devices with flexible/configurable logical channels (no preset) 61

5.3.2 [Provisionig-7] Assignment of client port to a flexible logical-channel (OTN matrix) ..... 61

5.3.3 [Provisioning-8] Assignment of logical channel in another logical channel (cross connection) 63

5.3.4 [Provisioning-9] Assignment of a logical channel into many logical channels and vice versa

(optional) 64

5.3.5

[Provisioning-10] Assignment of a logical channel into optical channel (line-port).....

65

5.3.6

[Provisioning-11] Configuration of rate (rate-class), protocol (trib-protocol and logical channel type) on client port.....

66

5.3.7

[Provisioning-12] Configuration of various optical channels into a superchannel (multiples optical channel into the same optical channel) .....

69

5.4

Service deletion operations .....

71

5.4.1

[Delete-1] Disable the client port .....

71

5.4.2

[Delete-2] Disable line port.....

72

5.4.3

[Delete-3] Disable logical channel and change assignment.....

72

5.5

Notification .....

73

5.5.1

[NETCONF notification-1] Subscribe for all path with Netconf.....

73

5.5.2

[NETCONF notification-2] Create subscription to specific path .....

75

5.5.3

[gRPC-1] Subscribe for all paths with gNMI/gRPC .....

77

5.5.4

[gRPC-2] Create subscription for specific path with gNMI .....

78

5.5.5

[gRPC-3] Create subscription of a specific path with a defined SAMPLE (sample\_interval)

79

5.5.6

[gRPC-4] Discovery of capabilities and encoding supported .....

79

5.6

PM operations .....

80

5.6.1

Client ports performance monitoring .....

80

5.6.2

Line ports performance monitoring .....

83

5.6.3

Example of PM RPC call.....

87

5.6.4

Example of creation a PM notification.....

87

5.6.5

Example of PM subscription with gnmi .....

88

5.7

Alarms .....

88

6

[Annex A] - Operational Mode augment .....

91

6.1

Yang model. Tree output .....

91

6.2

Yang model example .....

92

7

[Annex B] - Operations for netconf validation .....

95

8

[Annex C] - Preconfiguration of the telemetry behaviour of the device using OpenConfig telemetry-model.....

100

9

References .....

105

Glossary.....

105



Table of Figures

Figure 1: Optical Architectures: a) Aggregated, b) Partially Disaggregated and c) Fully Disaggregated .... 9

Figure 2: Monolithic aggregated network and scope of MUST Optical deliverable 1 ..... 10

Figure 3: Partial disaggregated network and scope of MUST Optical deliverable 2..... 11

Figure 4: Commissioning stage with preset mode ..... 16

Figure 5: Commissioning stage with no-preset mode..... 17

Figure 6: Components within the device, hierarchy and relations ..... 19

Figure 7: Modules retrieved in the discovery of line and client ports and subcomponents operation..... 24

Figure 8: Modules retrieved in the discovery of logical channel-assignment..... 41

Figure 9: Diagram of HW components relationship within OpenConfig model ..... 43

Figure 10: Relationship between locations parameters from OpenConfig to TAPI..... 43

Figure 11: Parts of a network elements and their relationship..... 46

Figure 12: Example of the component representation. .... 50

Figure 13: Logical channel created automatically with the operational mode..... 59

Figure 14: Logical channels and assignments created when transceivers is inserted ..... 62

Figure 15: Assignment of logical channel into another (cross-connection)..... 63

Figure 16: Device with multiplexing of logical channels..... 65

Figure 17: Assignment of logical channel into optical channel ..... 66

List of Tables

Table 1: Openconfig YANG models for optical SDN operations summary. .... 16



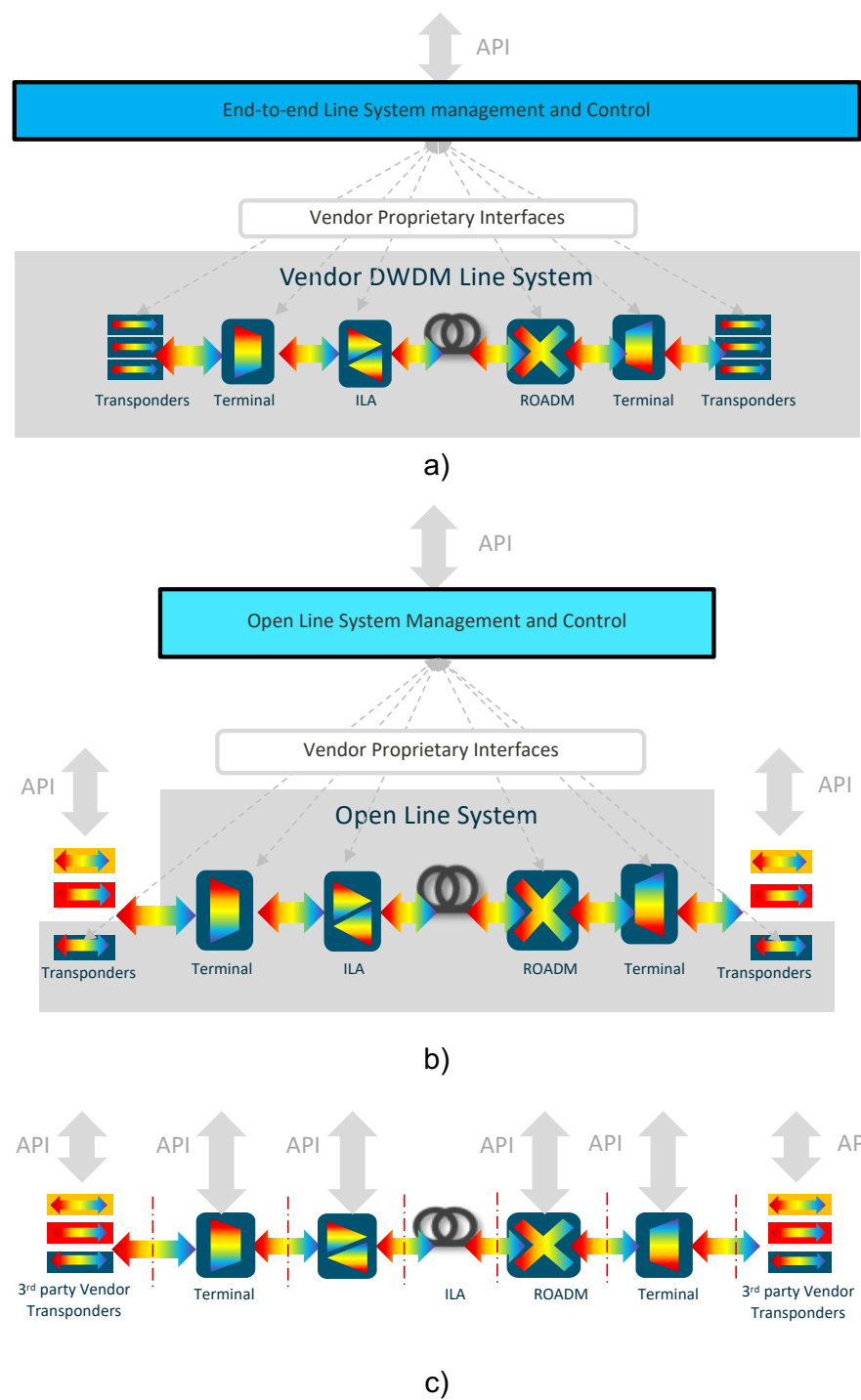
# 1 Introduction

The momentum toward open optical networking continues to build. It is mainly driven by two factors:

- Firstly, the rapid rate of innovation in coherent dense wavelength-division multiplexing (DWDM) transponders (in average every 2 years) leads to a natural desire to decouple the transponders from the other optical systems (with an average lifecycle of 10 years) in order to take advantage of the best transponders at any given time.
- Secondly, software-defined networking (SDN) principles and technologies, including open interfaces, abstraction and programmability, appear ready to help solve the complex problem of end-to-end management and control in multi-vendor optical networks. Open Line System (OLS) is widely seen as a concrete, realistic first step on the way toward higher optical network disaggregation.

Figure 1 shows different optical disaggregated architecture. The first approach is current close aggregated solution, second a partial disaggregated solution, where the Open Terminals (OTs) are disaggregated from the rest of the network (Open Optical Line System – O-OLS). The final step would be the fully disaggregated architecture, where each element in the line system is configurable.





**Figure 1: Optical Architectures: a) Aggregated, b) Partially Disaggregated and c) Fully Disaggregated**

Software Defined Networking (SDN) as a key technology enabler to bring dynamicity to the optical network. The idea is that enabling network programmability, it would be possible to carry out network creation, resources discovery and monitoring and service creation for L0-L1-L2 layers. The interfaces towards each of the abstraction elements should be able to configure and adjust the required optical parameters depending on the abstraction level.

In the first MUST optical deliverable [Reference] the focus of the specification was on Optical SDN domain controller North Bound interface for monolithic (aggregated networks) considering a set of prioritised use

cases. Option 1 based on T-APIv2.1.3 was selected in this first version due to higher maturity and wider Vendor support today while Option 2 based on IETF ACTN API is considered as another alternative option and we encourage that ONF T-API and IETF ACTN start to converge in the future at least regarding the set of use cases supported. Figure 2 below shows this monolithic (aggregated network) scenario with SDN:

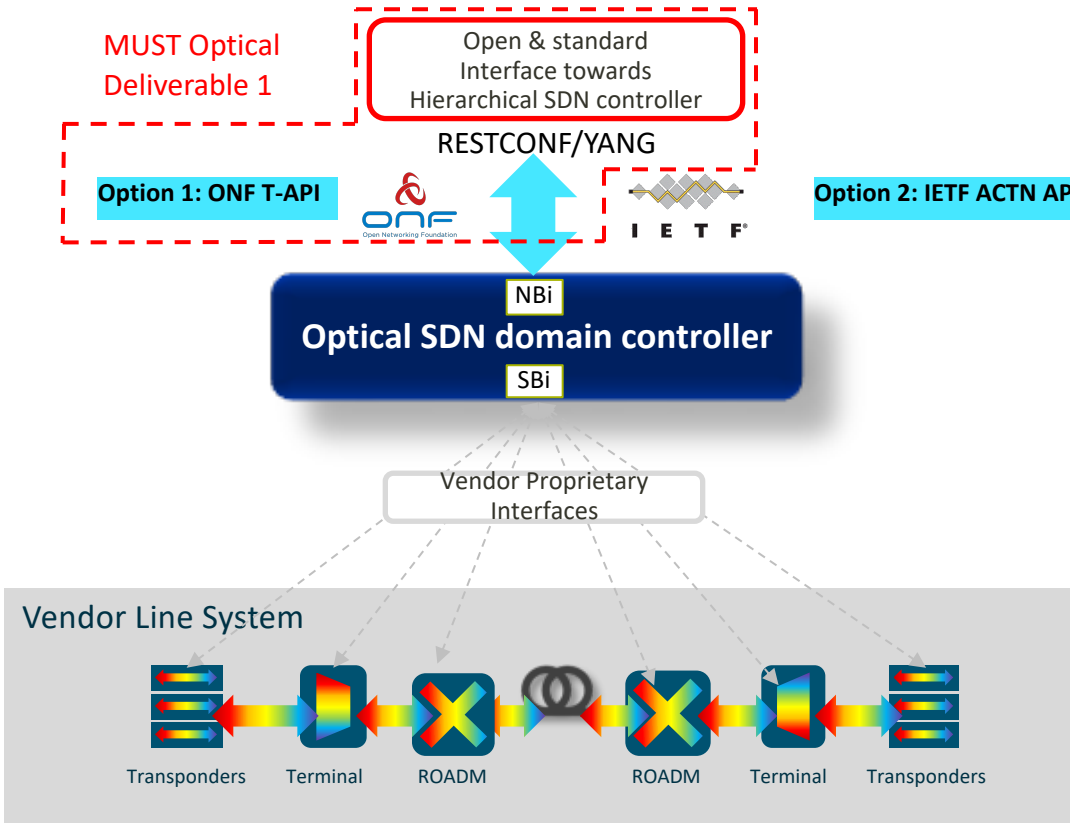


Figure 2: Monolithic aggregated network and scope of MUST Optical deliverable 1

As shown on the above figure, even if SDN brings the expected programmability and abstraction towards upper layers, this scenario is not enabling a complete multi-vendor approach in the optical network. The use of 3<sup>rd</sup> party Open Terminals (e.g. transponders/muxponders) is possible but at the expense of using different NMS/controllers from these 3<sup>rd</sup> party Vendors and without providing the end-to-end service visualisation, management and control for these alien wavelengths which represents a strong drawback for operators.

As a consequence the short/medium target is partially disaggregated networks and therefore to define the specifications for open and standard interfaces in OT (Open Terminals) North Bound Interface and as consequence in Optical SDN domain controller South Bound Interface as depicted in the next picture:

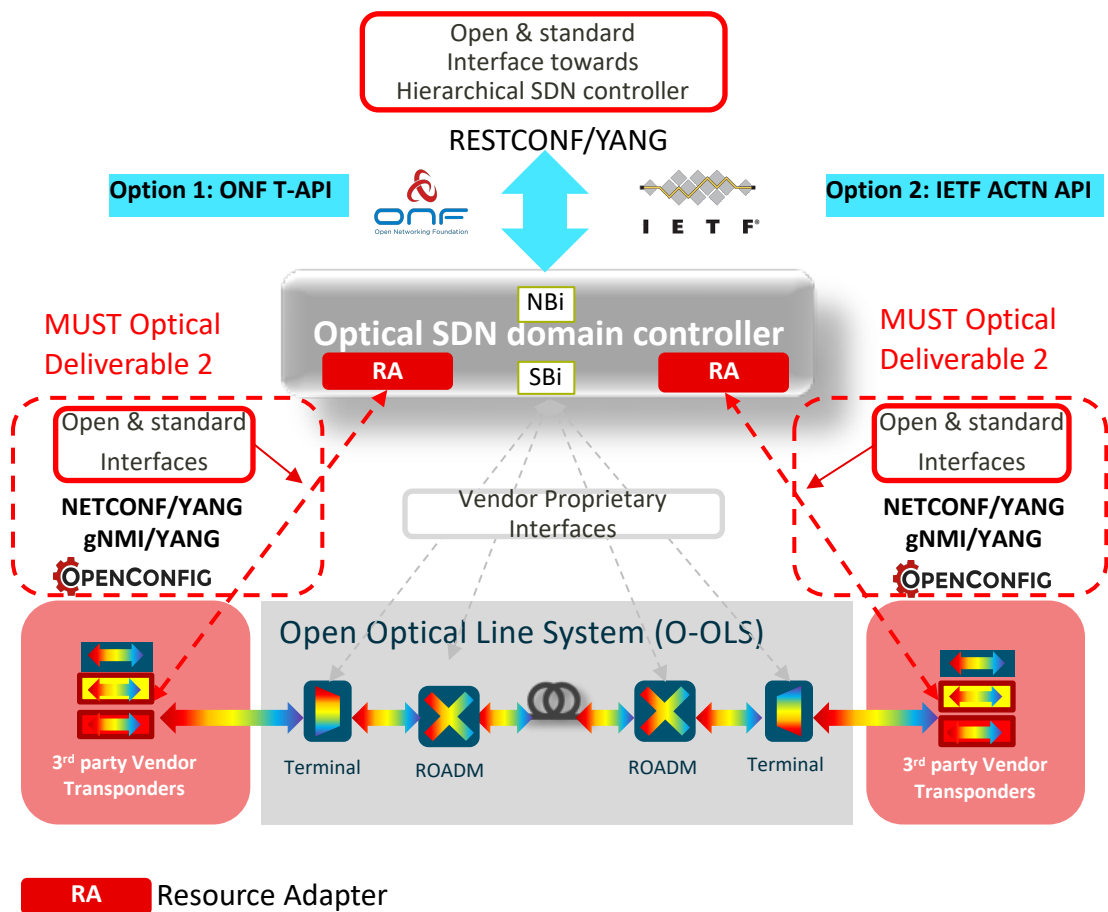


Figure 3: Partial disaggregated network and scope of MUST Optical deliverable 2

The advantages for operators of such partially disaggregated network are clear:

- True multi-vendor eco-system as originally sought by SDN introduction based on open and standard APIs (Netconf and gNMI) and data models (OpenConfig [Open]) between OTs and Optical SDN domain controller
- Possibility to introduce 3<sup>rd</sup> party transponders/muxponders and therefore alien wavelengths over the existing optical line system with a unified end-to-end network/service discovery, service/trails provisioning, performance and fault monitoring from the Optical SDN domain controller
- All this leveraging of course on the existing open and standard North Bound interface of the optical SDN domain controllers towards upper layers (e.g. Hierarchical SDN controller/OSS/Orchestration layer)

At architecture level they may be different approaches for this partial disaggregation scenario when considering single domain vs multi-domain or when considering monolithic aggregated networks that won't evolve towards SDN and possible role of the Hierarchical SDN controller etc. These architecture considerations are not in scope of this deliverable since a specific whitepaper is under preparation in TIP about these architectural considerations for Partial disaggregated networks and Open Optical Line Systems (O-OLS).

1.1 Goal

The purpose of this document is to describe the Optical SDN Controller SBI Technical Requirements for Open Terminals defined within the Open Optical Packet Transport (OOPT) working group.



## 1.2 Scope of the document

This document describes operations that can be done in the Open Terminals using OpenConfig. By Open Terminal we consider in this version of the specification the case of transponders and muxponders which is the priority currently for operators. More complex Open Terminals including for example a centralised OTN cross-connect are out-of-scope of this specification.

## 1.3 Document Structure

This document is structured as follows:

- Chapter 2: Protocol considerations Netconf and gRPC/gNMI protocols
- Chapter 3: Openconfig data model considerations
- Chapter 4: Initial state of device
- Chapter 5: Openconfig atomic operations
- Annexes

## 1.4 Release notes

**[DRAFT-Version]:** This release is in a draft version. All new operations and major changes in this version are not enforced for implementation, and the main objective is to disseminate this technical proposal and pull for comments and feedback to the industry to consolidate the specification into the next releases.

**Telecom infra project requests Netconf and/or gnmi for discovery, provisioning and deletion operations both** in OTs and O-OLS domain controller/optical SDN domain controller.

**[ANNEX A. OPERATIONAL MODE AUGMENT]:** Please note that this is a draft TIP MUST proposal agreed by several operators not standardized at OpenConfig and thus it is not a normative specification yet. The intention is to disseminate the proposal with the industry and gather feedback in order to propose it for standardization in the OpenConfig group, it is not TIP MUST intention to define a proprietary extension ad hoc for the TIP MUST group.

## 2 Protocol considerations

Telecom Infra Project proposes NETCONF and gNMI as the transport protocols for all the defined management operations in the NorthBound Device Interface Specification of Open Terminals.

### 2.1 Netconf

NETCONF is a network management protocol developed and standardized by the IETF published in June 2011 [RFC-6241].

#### 2.1.1 Data encoding

TIP state that any SBI implementation **MUST** support XML encoding format. All NETCONF messages **MUST** be well-formed XML, encoded in UTF-8 [RFC3629]. If a peer receives an <rpc> message that is not well-formed XML or not encoded in UTF-8, it **SHOULD** reply with a "malformed-message" error. If a reply cannot be sent for any reason, the server **MUST** terminate the session.



## 2.2 gRPC/gNMI

Telecom infra project also propose gRPC as the transport protocols for all the defined management operations in the NorthBound Device Interface Specification of Open Terminals. It is a communication protocol and framework built to standardize data transport over HTTP/2. It uses a client-server model to build a connected system and works across a variety of programming languages and platforms. A client application directly calls methods on a remote server application. The implementation of gNMI should be according to:

<https://github.com/openconfig/reference/blob/master/rpc/gNMI/gNMI-specification.md>

Also is required to support the gNMI path convention

<https://github.com/openconfig/reference/blob/master/rpc/gNMI/gNMI-path-conventions.md>

gNMI/gRPC is specially recommended for streaming telemetry. Streaming telemetry is a method of network monitoring that allows you to continuously stream data from network devices with efficient, incremental updates. Telemetry is sent based on subscriptions.

Streaming telemetry notifications are generated according to the type of subscription and at the frequency requested by the client.

The following modes MUST be supported

- **ONCE** — returns one-off data
- **STREAM** — a long-lived subscription that streams data according to triggers specified within the subscription, for example, a trigger could be when the state of an object changes, or based on a sampling interval. It include:
  - ON\_CHANGE. Only if data model change
  - SAMPLE. Repeated
  - TARGET-DEFINED

### 2.2.1 Data encoding

Any SBI implementation based on gRPC MUST support JSON, JSON\_IETF or Protobuf encoding formats.

## 3 OpenConfig considerations

This chapter describe the data model proposed by MUST for the NBI of the openterminals

### 3.1 OpenConfig SDK version and documentation

The OpenConfig project is a collaborative effort by network operators to develop programmatic interfaces and tools for managing networks in a dynamic, vendor-neutral way. Thus, its models are periodically updated. All OpenConfig models can be found at:

<https://github.com/openconfig/public>

### 3.2 OpenConfig Information model

The OpenConfig information model is composed by a set of abstract modules. Each one is composed by a set of YANG models and represents a specific capability and features of a network device, such as HW components hierarchy, interfaces, OSPF configuration, QoS, among others.

The main modules used for Optical SDN operations are the following:

- Platform:
  - o Platform - [openconfig-platform.yang](#) – It constitutes the main model to define the hardware components of a network device.
  - o CPU - [openconfig-platform-cpu.yang](#) – It augments the platform model to add specific parameters of a CPU component.
  - o FAN - [openconfig-platform-fan.yang](#) – It augments the platform model to add specific parameters of a FAN component.
  - o LINECARD - [openconfig-platform-linecard.yang](#) – It augments the platform model to add specific parameters of a Linecard component.
  - o PORT - [openconfig-platform-port.yang](#) – It augments the platform model to add specific parameters of a port component.
  - o PSU - [openconfig-platform-psu.yang](#) – It augments the platform model to add specific parameters of a PSU (Power Supply Unit) component.
  - o TRANSCEIVER - [openconfig-platform-transceiver.yang](#) – It augments the platform model to add specific parameters of a Transceiver component.
  - o Platform Types - [openconfig-platform-types.yang](#) – It defines the types used to define the parameters in the platform module parameters
- Optical-Transport:
  - o Terminal Device - [openconfig-terminal-device.yang](#) – It defines the main model to define a terminal optics device.
  - o Optical Transport Types - [openconfig-transport-types.yang](#) – It defines the types used to define the parameters in the optical transport module parameters.

The entire OpenConfig list of YANG models used in Optical SDN can be found in the following table:

Model	Req version <sup>1</sup>
<b>openconfig-platform-transceiver.yang</b>	0.7.1
<b>openconfig-platform.yang</b>	0.13
<b>openconfig-platform-linecard.yang</b>	0.1.2
<b>openconfig-platform-port.yang</b>	0.3.3
<b>openconfig-platform-types.yang</b>	1.1.0
<b>openconfig-platform-cpu.yang</b>	0.1.1
<b>openconfig-platform-fan.yang</b>	0.1.1
<b>openconfig-platform-psu.yang</b>	0.2.1
<b>openconfig-terminal-device.yang</b>	1.7.3
<b>openconfig-transport-types.yang</b>	0.12
<b>openconfig-transport-line-common.yang</b>	0.6.0
<b>openconfig-types.yang</b>	0.6.0
<b>openconfig-interfaces.yang</b>	2.4.3
<b>openconfig-yang-types.yang</b>	0.2.2
<b>openconfig-alarm-types.yang</b>	0.2.1
<b>openconfig-alarms.yang</b>	0.3.2
<b>openconfig-system.yang</b>	0.9.1
<b>openconfig-lldp.yang</b>	0.2.1

<sup>1</sup> These are the minimum required versions to be supported. Implementations based on newer versions of the models are also valid if it does not break backward compatibility.

openconfig-ldp-types.yang	0.1.1
openconfig-telemetry.yang	0.5.1
openconfig-platform-ext.yang	0.1.1
openconfig-telemetry-types.yang	0.4.2

Table 1: Openconfig YANG models for optical SDN operations summary.

4 Initial state of device (commissioning stage)

We require a minimum configuration existing in the device in the initial moment prior to starting any configuration or discovery operation. This configuration must be set in a commissioning stage by the supplier and include the predefined IP and the credentials for the device management as well as a basic structure of the logical channels and the assignments, as described later. Management licences are out of the scope of this document, but we encourage the solutions to support an easy procedure to manage them. The following picture shows the configuration required during the commissioning.

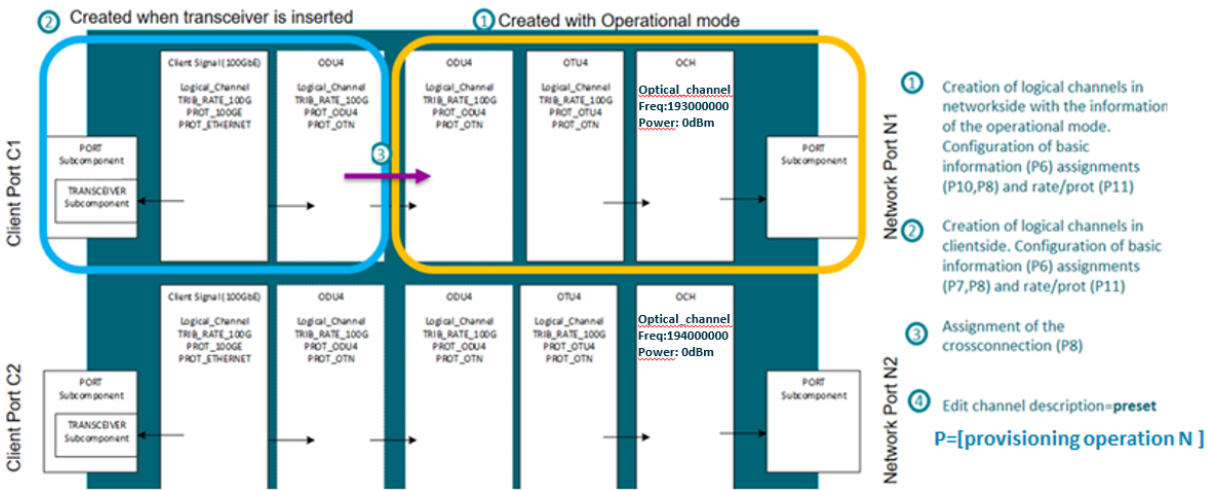


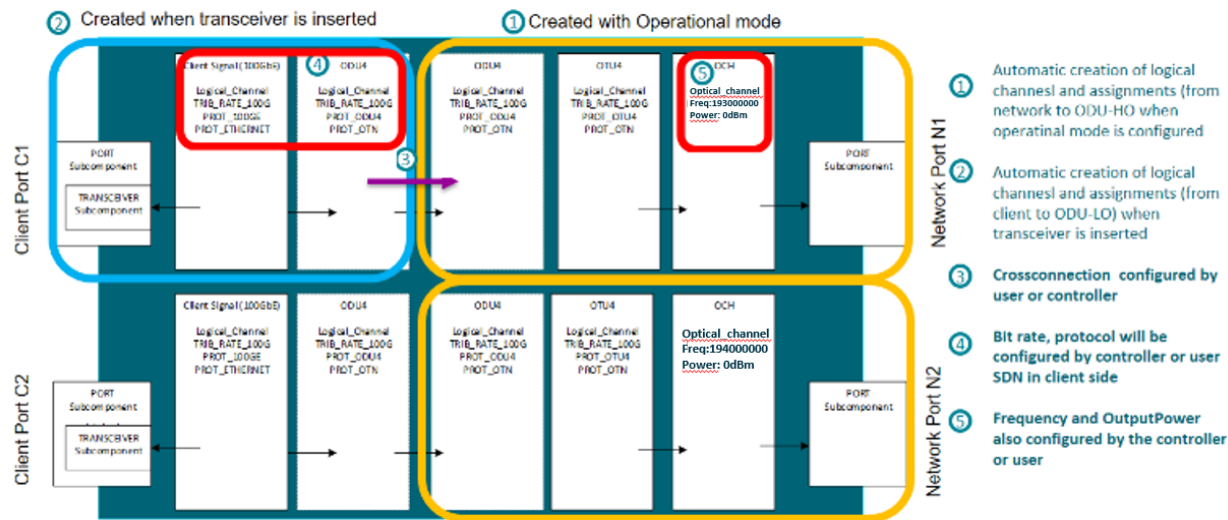
Figure 4: Commissioning stage with preset mode

1. Creation of logical channels, assignments and basic configuration necessary (everything except frequency and output power) to the operational mode selected, from the logical channel associated with the network/line port through the optical channel to the ODU-HO logical channel
2. Creation of logical channels, assignments and the basic configuration (everything except bit rate and protocol) from logical channel associated with ingress transceiver to ODU-LO channel.
3. Depending on type of device (preset or flexible) see [Discovery-9], assignment between ODU-HO and ODU-LO must be configured in the commissioning stage or in a later stage by the SDN-C/ Devices with fixed cross connections (transponders) must preset the assignments. Devices with flexibles assignments (OTN matrix) will be configured by the SDN-C/SDTN or the user during configuration stage.

**Note:** This version focusses on transponders and muxponders without a centralized OTN matrix. Open terminals with centralized matrix are kept for a future version.

Once finished the commissioning stage, the device is ready for operation. The following picture shows an example of preset and not preset (flexible) device configuration once the commissioning stage is finished. All the operations necessities to accomplish the provisioning are describes in the Atomic operations [\(chapter 5\)](#)





## 5 OpenConfig atomic operations.

This chapter describes the atomic OpenConfig operations required to accomplish the use cases of discovery and provisioning of E2E services

The operations are divided in four packs:

- **Discovery operations.** Such operations refer to required operations to obtain the device ports, available capabilities and the actual type of the device (preset or configurable).
- **Provisioning operations** for device with “preset” logical channels. As stated, this refers to provisioning operations required in devices without programmable cross connections (such as transponders or muxponders).
- **Provisioning operations** for device with configurable cross connections (OTN matrix configurable)
- **Delete operations** refer to operations needed to remove configuration, e.g.release a cross-connection

**Important note:** Discovery-1, Discovery-2, Discovery-3 and Provisioning-1 operation show the calls using both protocols and the gNMI\_cli client. The rest of operations only show the Netconf example but the gNMI call can be deduced from the example of Discovery-1, 2, 3 and the Provisionig-1 operations.

### 5.1 Discovery operations

These operations are common for all types of devices

- [Discovery-1] Discovery of all components
- [Discovery-2] Discovery of type of component
- [Discovery-3] Discovery of line and client ports and subcomponents
- [Discovery-4] Discovery of operational modes available in device.
- [Discovery-5] Discovery of admin and operational status of port



- [Discovery-5b] Discovery of admin status of interface
- [Discovery-6] Discovery of client port capabilities (protocol type, rate and mapping)
- [Discovery-7] Discovery of line port capabilities (protocol type, rate granularity and allocation)
- [Discovery-8] Discovery of frequency and operational mode configured in a line port
- [Discovery-9] Discovery of type of logical channel assignments (Preset or flexible)
- [Discovery-10] Discovery of logical channel assignment
- [Discovery-11] Discovery of location of a component and mapping to INVENTORY\_ID (related to its relationship to the TAPI model and how the device can be integrated.)
- [Discovery-12] Retrieval of global structure of HW inventory.
- [Discovery-13] Retrieval of inventory information of each component.
- [Discovery-14] Retrieval of Slot identifiers (Slot-id) information-

5.1.1 [Discovery-1] Discovery of all components

This operation will retrieve the list of components within a device. The OpenConfig Platform models specify that a given device is composed of “components”. Each component is characterized by its type and its attributes depend on it. It is worth noting that device “ports” are a type of component, but there are other component types. In short, the discovery operation will retrieve each component (uniquely identified by its name). We consider the following picture, which represents the structure of the components within the device and the relation between them.

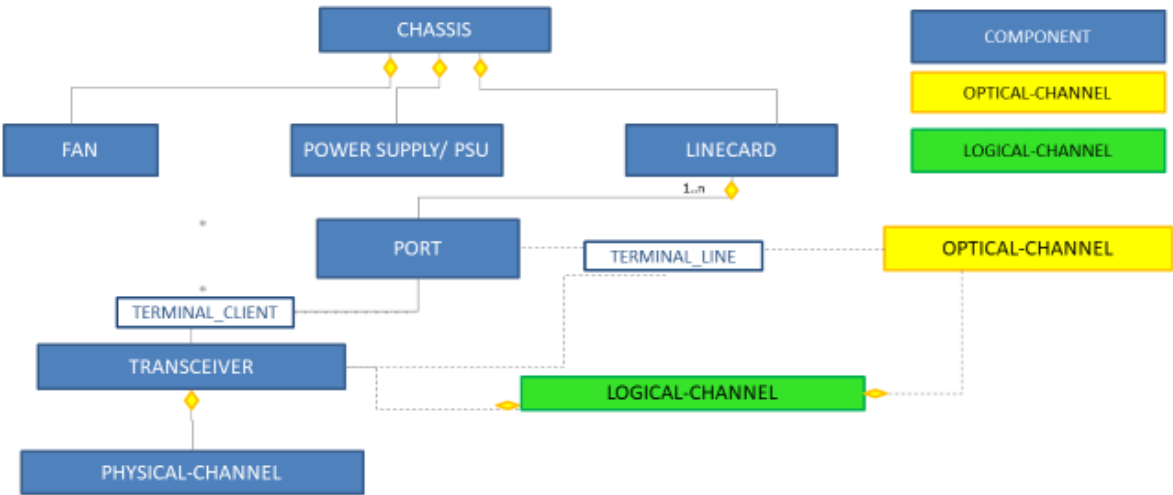


Figure 6: Components within the device, hierarchy and relations

RPC call using Netconf

```
<get>
  <filter type="subtree">
    <components xmlns="http://openconfig.net/yang/platform">
      <component>
        <name/>
      </component>
    </components>
  </filter>
</get>
```

Path structure with gNMI

```
path {
  elem {
    name: "components"
  }
  elem {
```

```
name: "component"
key {
  key: "name"
  value: "*"
}
}
elem {
  name: "state"
}
elem {
  name: "name"
}
}
```

GNMI\_cli

```
./ gNMI_cli -address {{server_ip_address:port}} \
  -get \
  -proto "path: <elem: <name: 'components'> elem:<name:'component'>
elem: <name: 'state'> elem: <name:'name'>>" \
  -timeout 5s -alsologtostderr \
  -client_cert certs/client1.crt \
  -client_key certs/client1.key \
  -ca_cert certs/onfca.crt
```

/components/component				
Attribute	Datatype/Members/Description	Mod	Sup	Notes
name	Leafref// Device name for the component – this must not be a configurable parameter	RO	M	• Provided by server
state/type	openconfig-platform-types: OPENCONFIG_HARDWARE_COMPONENT openconfig-platform-types:PORT, CHASSIS,LINECARD, FRU, FAN, POWER_SUPPLY, TRANSCEIVER, CPU, CONTROLLER_CARD, STORAGE,	RO	M	• Provided by server

Reply example

```
<data>
  <components xmlns="http://openconfig.net/yang/platform">
    <component>
      <name>SYSTEM-SOFTWARE</name>
    </component>
    <component>
      <name>CHASSIS-1</name>
    </component>
    <component>
```

```
<name>CARD-1-2</name>
</component>
<component>
  <name>PORT-1-2-1</name>
</component>
<component>
  <name>OCH-1-2-1</name>
</component>
<component>
  <name>PORT-1-2-2</name>
</component>
<component>
  <name>TRANSCEIVER-1-2-2</name>
</component>
<component>
  <name>PORT-1-2-3</name>
</component>
<component>
  <name>TRANSCEIVER-1-2-3</name>
</component>
<component>
  <name>CARD-1-4</name>
</component>
<component>
  <name>PORT-1-4-1</name>
</component>
<component>
  <name>OCH-1-4-1</name>
</component>
<component>
  <name>PORT-1-4-2</name>
</component>
<component>
  <name>TRANSCEIVER-1-4-2</name>
</component>
<component>
  <name>PORT-1-4-3</name>
</component>
<component>
  <name>TRANSCEIVER-1-4-3</name>
</component>
</components>
</data>
```

5.1.2 [Discovery-2] Discovery of type of component

This operation obtains the type of the component given as a parameter. The name must be a valid name, e.g. as retrieved from the [Discovery-1].

Netconf RPC call example

```
<get>
  <filter type="subtree">
    <components xmlns="http://openconfig.net/yang/platform">
      <component>
        <name>{{name_of_component}}</name>
        <state>
          <type/>
```

```
        </state>
      </component>
    </components>
  </filter>
</get>
```

Path structure with gNMI

```
path {
  elem {
    name: "components"
  }
  elem {
    name: "component"
    key {
      key: "name"
      value: "{name_of_component}"
    }
  }
  elem {
    name: "state"
  }
  elem {
    name: "type"
  }
}
```

gNMI cli call example

```
./ gNMI_cli -address {{server_ip_address:port}} \
    -get \
    -proto "path: <elem: <name: 'components'> elem:<name:'component'
key<key:'name' value:'{{name_of component}}' >> elem: <name: 'state'> elem:
<name:'type'>>" \
    -timeout 5s -alsologtostderr \
    -client_cert certs/client1.crt \
    -client_key certs/client1.key \
    -ca_cert certs/onfca.crt
```

/components/component				
Attribute	Data type/Members/description	Mod	Sup	Notes
name	Leafref// Device name for the component – this must not be a configurable parameter	RO	M	• Provided by server
State/type	openconfig-platform-types: OPENCONFIG_HARDWARE_COMPONENT openconfig-platform-types:PORT, CHASIS,LINECARD, FRU, FAN,	RO	M	• Provided by server

	POWER_SUPPLY, TRANSCEIVER, CPU, CONTROLLER_CARD, STORAGE,			
--	--	--	--	--

Netconf RPC reply example

```
<data xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <components xmlns="http://openconfig.net/yang/platform">
    <component>
      <name>PORT-1-4-L1</name>
      <state>
        <name>PORT-1-4-L1</name>
        <type xmlns:openconfig-platform-
types="http://openconfig.net/yang/platform-types">openconfig-platform-
types:PORT</type>
      </state>
    </component>
  </components>
</data>
```

Addendum: to retrieve all components of type “PORT”

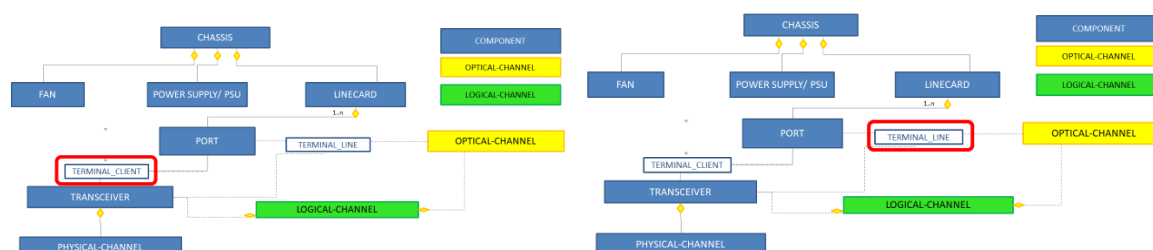
RPC call

```
<get>
  <filter type="subtree">
    <components xmlns="http://openconfig.net/yang/platform">
      <component>
        <name/>
        <state>
          <type xmlns:oc-platform-
types="http://openconfig.net/yang/platform-types">oc-platform-
types:PORT</type>
        </state>
      </component>
    </components>
  </filter>
```

5.1.3 [Discovery-3] Discovery of line and client ports and subcomponents

This operation obtains the information about a component of PORT type including the optical port type and its subcomponents. This allows the controller to differentiate client and line ports using optical-port-type attribute. The operation retrieves the optical-port-type state leaf of the port augment for optical devices.

Figure 7 shows the modules retrieved in this operation.



**Figure 7: Modules retrieved in the discovery of line and client ports and subcomponents operation**

## Netconf RPC call

```
<get>
  <filter type="subtree">
    <components xmlns="http://openconfig.net/yang/platform">
      <component>
        <name>{{name_of_component_type_port}}</name>
        <subcomponents/>
        <port>
          <optical-port>
            <state>
              <optical-port-type/>
            </state>
          </optical-port>
        </port>
      </component>
    </components>
  </filter>
</get>
```

### RPC CLIENT PORT reply example

```
<data xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <components xmlns="http://openconfig.net/yang/platform">
    <component>
      <name>PORT-1-4-C1</name>
      <subcomponents>
        <subcomponent>
          <name>TRANSCEIVER-1-4-C1</name>
          <config>
            <name>TRANSCEIVER-1-4-C1</name>
          </config>
          <state>
            <name>TRANSCEIVER-1-4-C1</name>
          </state>
        </subcomponent>
      </subcomponents>
    <port>
      <optical-port xmlns="http://openconfig.net/yang/transport-line-
common">
        <state>
          <optical-port-type xmlns:openconfig-transport-line-
common="http://openconfig.net/yang/transport-line-
common">TERMINAL CLIENT</optical-port-type>
```



```
        </state>
      </optical-port>
    </port>
  </component>
</components>
</data> </components>
</data>
```

RPC LINE PORT reply example

```
<data xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <components xmlns="http://openconfig.net/yang/platform">
    <component>
      <name>PORT-1-4-L1</name>
      <subcomponents>
        <subcomponent>
          <name>TRANSCEIVER-1-4-L1</name>
          <config>
            <name>TRANSCEIVER-1-4-L1</name>
          </config>
          <state>
            <name>TRANSCEIVER-1-4-L1</name>
          </state>
        </subcomponent>
        <subcomponent>
          <name>OCH-1-4-L1</name>
          <config>
            <name>OCH-1-4-L1</name>
          </config>
          <state>
            <name>OCH-1-4-L1</name>
          </state>
        </subcomponent>
      </subcomponents>
      <optical-port xmlns="http://openconfig.net/yang/transport-line-common">
        <state>
          <optical-port-type xmlns:openconfig-transport-line-
common="http://openconfig.net/yang/transport-line-common">oc-opt-
types:TERMINAL_LINE</optical-port-type>
        </state>
      </optical-port>
    </component>
  </components>
</data>
```

All the client interfaces will have the optical-port-type attribute equal to TERMINAL\_CLIENT and the lineports will use TERMINAL\_LINE.

gNMI get call

```
./ gNMI_cli -address {{server_ip_address:port}} \
  -get \
  -proto "path: <elem: <name: 'components'> elem:<name:'component'
key<key:'name' value:'{{name_of component}}' >> elem: <name: 'port'> elem:
<name:'optical-port'> elem: <name: 'state'> elem: <name: 'optical-port-type'>
">" \
  -timeout 5s -alsologtostderr \
  -client_crt certs/client1.crt \
```

```
-client_key certs/client1.key \  
-ca_cert certs/onfca.crt
```

gNMI Reply

```
notification: <  
  timestamp: 1601454667571390000  
  update: <  
    path: <  
      elem: <  
        name: "components"  
      >  
      elem: <  
        name: "component"  
        key: <  
          key: "name"  
          value: "PORT-A"  
        >  
      >  
      elem: <  
        name: "port"  
      >  
      elem: <  
        name: "optical-port"  
      >  
      elem: <  
        name: "state"  
      >  
      elem: <  
        name: "optical-port-type"  
      >  
    >  
    val: <  
      json_ietf_val: "\"openconfig-transport-types:TERMINAL_LINE\""  
    >  
  >  
>
```

/components/component/port/oc-line-com:optical-port/oc-line-com:state				
Attribute	Allowed Values/Format	Mod	Sup	Notes
admin-state	["ENABLED", "DISABLED", "MAINT"]	RW	M	• Provided by server or user
optical-port-type	oc-opt-types:["TERMINAL_LINE", "TERMINAL_CLIENT", "MONITOR", "DROP", "ADD", "EGRESS", "INGRESS"]	RO	M	• Provided by server

/components/component	
-----------------------	--

Attribute	Allowed Values/Format	Mod	Sup	Notes
name	String "Name of the port"	RO	M	• Provided by server
state/type	openconfig-platform-types:PORT	RO	M	• Provided by server
oper-status	["ACTIVE", "INACTIVE", "DISABLE"]	RO	M	• Component is enabled and active • Component is enabled BUT INACTIVE • Component is administratively disabled
subcomponents/ subcomponent/name	String.	RO	M	• Provided by server • It is assumed that subcomponent is TRANSCEIVER type and OCH (only for line ports)
parent	• Leafref	RO	M	Reference to the name of the parent component. Note that this reference must be kept synchronized with the corresponding subcomponent reference from the parent component.

5.1.4 [Discovery-4] Discovery of operational modes available and capabilities using operational-mode-augment (Annex A)

**Disclaimer:** Please note that this use case is a draft proposal and not a normative specification yet. The intention is to disseminate the proposal with the industry and gather feedback in order to propose it for standardization in the appropriate fora.

This operation will obtain the operational modes available in device. Operational modes define the modulation and the main capabilities available within the line ports. We propose to use the augment from [annex A](#) to be able to expose the capabilities of the device.

RPC call to retrieve the details of an operational mode

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="1">
  <get>
    <filter>
      <terminal-device
xmlns='http://openconfig.net/yang/terminal-device'>
        <operational-modes>
          <mode>
            <mode-id>100</mode-id>
            <state/>
          </mode>
        </operational-modes>
      </terminal-device>
    </filter>
  </get>
</rpc>
```

RPC reply example

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="1">
  <data>
    <terminal-device xmlns="http://openconfig.net/yang/terminal-device">
      <operational-modes>
        <mode>
          <mode-id>100</mode-id>
          <config/>
          <state>
            <mode-id>100</mode-id>
            <description>MODE20</description>
            <vendor-id>TIPMUST</vendor-id>
            <oc-oper-mode:standard-mode
              xmlns:oc-oper-
mode="http://example.net/yang/operational-mode-draft-tipmust">B-DScW-ytz (v)
              </oc-oper-mode:standard-mode>

            </state>
          </property>
        </properties>
      </mode>
      <mode>
        <mode-id>101</mode-id>
        <config/>
        <state>
          ...
        </state>
      </mode>
      <mode>
        ...
      </mode>
    </operational-modes>
  </terminal-device>
</data>
</rpc-reply>
```

RPC call to retrieve operational models at 100G

```
<?xml version="1.0" encoding="UTF-8"?>
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="1">
  <get>
    <filter>
      <terminal-device xmlns='http://openconfig.net/yang/terminal-device'>
        <operational-modes>
          <mode>
            <state>
              <operational-mode-capabilities
xmlns="http://example.net/yang/operational-mode-draft-tipmust">
              <bit-rate xmlns:oc-opt-
types="http://openconfig.net/yang/transport-types">oc-opt-types:TRIB-RATE-
100G</bit-rate>
              </operational-mode-capabilities>
            </state>
          </mode>
        </operational-modes>
      </terminal-device>
    </filter>
  </get>
</rpc>
```

RPC reply with 100G mode



```
<mode-id>54</mode-id>
<config/>
<state>
  <mode-id>54</mode-id>
  <description>MODE54</description>
  <vendor-id>VENDOR_A</vendor-id>
  <operational-mode-capabilities
xmlns="http://example.net/yang/operational-mode-draft-tipmust">
    <description>Additional details</description>
    <modulation-format>8QAM</modulation-format>
    <baud-rate>37.5</baud-rate>
    <bit-rate xmlns:oc-opt-
types="http://openconfig.net/yang/transport-types">oc-opt-types:TRIB-RATE-
100G</bit-rate>
      <min-osnr>13.5</min-osnr>
      <grid-type>FLEX</grid-type>
    <adjustment-granularity>G_25GHZ</adjustment-granularity>
    <otsi-media-channel>40</otsi-media-channel>
    <effective-media-channel>50</effective-media-channel>
      <central-frequency-min>191000000</central-frequency-min>
      <central-frequency-max>196000000</central-frequency-max>
      <fec-coding xmlns:oc-oper-
mode="http://example.net/yang/operational-mode-draft-tipmust">oc-oper-
mode:FEC-G</fec-coding>
        <min-output-power>0.0</min-output-power>
        <max-output-power>2.0</max-output-power>
        <input-power-sensitivity>-10.0</input-power-sensitivity>
        <minimum-q-value>0.0</minimum-q-value>
        <chromatic-dispersion-tolerance>0.0</chromatic-dispersion-
tolerance>
        <differential-group-delay-tolerance>0.0</differential-group-
delay-tolerance>
        <filter>
          <shape>ROLL_Shape</shape>
          <order>3</order>
          <roll-off>0.3</roll-off>
        </filter>
        <sop>FAST_SOP</sop>
      </operational-mode-capabilities>
    </state>
  <properties>
    <property>
      <name>FactorX</name>
      <config>
        <name>FactorX</name>
        <value>10</value>
      </config>
      <state>
        <name>FactorX</name>
        <value>10</value>
        <configurable>true</configurable>
      </state>
    </property>
  </properties>
  <mode-id>101</mode-id>
  <config/>
  <state>
    ...
  </state>
</mode>
<mode>
  ...
```

```

    </mode>
  </operational-modes>
</terminal-device>
</data>
</rpc-reply>
```

RPC call to retrieve min and max frequencies of a given mode

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="1">
  <get>
    <filter>
      <terminal-device
xmlns='http://openconfig.net/yang/terminal-device'>
        <operational-modes>
          <mode>
            <mode-id>101</mode-id>
            <state>
              <operational-mode-capabilities
xmlns="http://example.net/yang/operational-mode-draft-tipmust">
                <central-frequency-min/>
                <central-frequency-max/>
              </operational-mode-capabilities>
            </state>
          </mode>
        </operational-modes>
      </terminal-device>
    </filter>
  </get>
</rpc>
```

RPC reply

```
<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="1">
  <data>
    <terminal-device xmlns="http://openconfig.net/yang/terminal-device">
      <operational-modes>
        <mode>
          <mode-id>101</mode-id>
          <state>
            <operational-mode-capabilities
xmlns="http://example.net/yang/operational-mode-draft-tipmust">
              <central-frequency-min>191000000</central-frequency-min>
              <central-frequency-max>196000000</central-frequency-max>
            </operational-mode-capabilities>
          </state>
        </mode>
      </operational-modes>
    </terminal-device>
  </data>
</rpc-reply>
```

RPC call to retrieve operational modes with a given min freq

```
<?xml version="1.0" encoding="UTF-8"?>
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="1">
  <get>
    <filter>
```

```
<terminal-device xmlns='http://openconfig.net/yang/terminal-
device'>
  <operational-modes>
    <mode>
      <mode-id/>
      <state>
        <operational-mode-capabilities
xmlns="http://example.net/yang/operational-mode-draft-tipmust">
          <central-frequency-min>191000000</freq-min>
        </operational-mode-capabilities>
      </state>
    </mode>
  </operational-modes>
</terminal-device>
</filter>
</get>
</rpc>
```

**RPC call using x-path to select the mode for 64 QAM, obtain only the mode-id**

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="1">
  <get>
    <filter type="xpath"
xmlns:td='http://openconfig.net/yang/terminal-device'
xmlns:om='http://example.net/yang/operational-mode-draft-tipmust'
select="/td:terminal-device/td:operational-
modes/td:mode[td:state/om:operational-mode-capabilities/om:modulation-
format='64QAM']/td:mode-id"/>
  </get>
</rpc>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="1">
  <data>
    <terminal-device xmlns="http://openconfig.net/yang/terminal-device">
      <operational-modes>
        <mode>
          <mode-id>102</mode-id>
        </mode>
      </operational-modes>
    </terminal-device>
  </data>
</rpc-reply>
```

5.1.5 [Discovery-5] Discovery of admin status of optical port

This operation retrieves the main information required for the optical port. In case optical port container is not supported, operation Discovery-5b is required to enable the admin-status of the port

RPC call

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="101">
  <get>
    <filter type="subtree">
      <components xmlns="http://openconfig.net/yang/platform">
        <component>
          <name>PORT-1-4-L1</name>
          <port>
            <optical-port>
              <state>
                <admin-state/>
              </state>
            </optical-port>
          </port>
        </component>
      </components>
    </filter>
  </get>
</rpc>
```

RPC reply example

```
<data xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <components xmlns="http://openconfig.net/yang/platform">
    <component>
      <name>PORT-1-4-L1</name>
      <port>
        <optical-port xmlns="http://openconfig.net/yang/transport-line-
common">
          <state>
            <admin-state xmlns="http://openconfig.net/yang/transport-line-
common">ENABLED</admin-state>
          </state>
        </optical-port>
      </port>
    </component>
  </components>
</data>
```

/components/component/				
Attribute	Allowed Values/Format	Mod	Sup	Notes
name	String			• Provided by server • Name of the port (Ex 1-4-1)
port/optical- port/state/admin-state	• [ENABLED,DISABLED, MAINT]	RO	M	• Provided by <i>server</i>



5.1.6 [Discovery-5b] Discovery of admin status of interface

This operation retrieves the information regarding the admin-status of the interface. This operation is required in case optical-port container is not supported.

**Note: optical-port admin status and the interface admin status must be synchronized in case that both were present**

Netconf RPC call example

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="101">
  <get>
    <filter>
      <interfaces xmlns="http://openconfig.net/yang/interfaces">
        <interface>{{interface_name}}</interface>
        <config>
          <enabled/>
        </config>
      </interface>
    </interfaces>
  </filter>
</get> </rpc>
```

RPC reply example

```
<data xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <interfaces xmlns="http://openconfig.net/yang/interfaces">
    <interface>{{interface_name}}</interface>
    <config>
      <enabled>TRUE</enabled>
    </config>
  </interface>
</interfaces>
</data>
```

/components/component/				
Attribute	Allowed Values/Format	Mod	Sup	Notes
name	String			• Provided by server • Name of the port (Ex 1-4-1)
port/optical-port/state/admin-state	• [ENABLED,DISABLED, MAINT]	RO	M	• Provided by server

5.1.7 [Discovery-6] Discovery of client port capabilities (protocol, bit rate and mapping).

This operation gets the client port capabilities (protocol, bit rate, mapping and granularity available).

**NOTE: When the transceiver is inserted in the chassis, the logical channel assigned will be automatically generated with the lower bit rate available.**

RPC call to obtain all channels

```
<get>
  <filter type="subtree">
    <terminal-device xmlns="http://openconfig.net/yang/terminal-device">
      <logical-channels>
        <channel/>
      </logical-channels>
    </terminal-device>
  </filter>
</get>
```

To obtain the channels associated with the client (TRANSCEIVER), is recomendable to use the filter type xpath.

For achive this operations is highly recommended to support capability xpath:1.0. (filter type=xpath)

```
<capability>urn:ietf:params:netconf:capability:xpath:1.0</capability>
```

```
<get>
  <filter type="xpath">
    <terminal-device xmlns:t="http://openconfig.net/yang/terminal-device">
      select="/t:terminal-devices/t:logical-
channel/t:channel[t:ingress={{name of the transceiver in ingress}}]"/>
    </filter>
</get>
```

If capability xpath:1.0 is not available the user can find the same information once we have obtained the index of the channel.

**RPC example to obtain index of ingress channel associated whit transceiver in client port**

```
<get>
  <filter type="subtree">
    <terminal-device xmlns="http://OpenConfig.net/yang/terminal-device">
      <logical-channels>
        <channel>
          <ingress>
            <state>
              <transceiver>{{name of transceiver}}<transceiver>
            </state>
          </ingress>
          <index/>
        </channel>
      </logical-channels>
    </terminal-device>
  </filter>
</get>
```

**RPC call example to obtain capabilities**

The following operation is used to retrieve the configuration of a logical channel given its index.

```
<get>
  <filter type="subtree">
    <terminal-device xmlns="http://openconfig.net/yang/terminal-device">
      <logical-channels>
        <channel>
          <index>{{Index channel associated ingress-port}}</index>
          <config/>
          <otn/>
          <logical-channel-assignments/>
        </channel>
      </logical-channels>
    </terminal-device>
  </filter>
</get>
```

```
        </logical-channels>
      </terminal-device>
    </filter>
  </get>
```

Then using the right index we can obtain the capabilities.

RPC reply for capabilities

```
<data xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <terminal-device xmlns="http://OpenConfig.net/yang/terminal-device">
    <logical-channels>
      <channel>
        <index>20975681</index>
        <config>
          <index>20975681</index>
          <rate-class xmlns:oc-opt-
types="http://openconfig.net/yang/transport-types">oc-opt-
types:TRIB_RATE_40G</rate-class>
          <trib-protocol xmlns:oc-opt-
types="http://openconfig.net/yang/transport-types">oc-opt-
types:PROT_40GE</trib-protocol>
          <logical-channel-type xmlns:oc-opt-
types="http://openconfig.net/yang/transport-types">oc-opt-
types:PROT_ETHERNET</logical-channel-type>
        </config>
        <logical-channel-assignments>
          <assignments>
            <config>
              <mapping>GMP</mapping>
            </config>
          </assignments>
        </logical-channel-assignments>
      </channel>
    </logical-channels>
  </terminal-device>
```

The following parameters are mandatory:

/terminal-device/logical-channels/channel				
Attribute	Allowed Values/Format	Mod	Sup	Notes
index	uint32	RW	M	Reference to the index of the logical channel
description	String (preset or flexible)	RO	M	Indicate the type of assignment(preset or flexible)(see [Discovery-9] ) Provided by server
logical-channel-assignment/assignment /config/assignment-type	Type enumeration: LOGICAL_CHANNEL OPTICAL_CHANNEL	RW	M	Provided by OT Subsequent channel is a logical channel"; "Subsequent channel is an optical channel / carrier

logical-channel-assignment/assignment/config/logical-channel	Reference to the index of the logical-channel	RW	M	Provided by <i>OT or user</i>
logical-channel-assignment/assignment/config/assignment-type/optical-channel	Reference to the name of the optical-channel	RW	M	Provided by <i>OT or user</i>
trib-protocol	<ul style="list-style-type: none"><li>PROT_1GE, OC48, STM16, 10GE_LAN, 10GE_WAN, OC192, STM64, OTU2, OTU2E, OTU1E, ODU2, ODU2E, 40GE, OC768, STM256, OTU3, ODU3, 100GE, 100GE_MLG, OTU4, OTUCN, ODU4</li></ul>	RW	M	Base identity for protocol framing used by tributary signals. <ul style="list-style-type: none"><li>rate class: 1G protocols: 1GE</li><li>rate class: 2.5G protocols: OC48, STM16</li><li>rate class: 10G protocols: 10GE LAN, 10GE WAN, OC192, STM64, OTU2, OTU2e, OTU1e, ODU2, ODU2e, ODU1e</li><li>rate class: 40G protocols: 40GE, OC768, STM256, OTU3, ODU3</li><li>rate class: 100G protocols: 100GE, 100G MLG, OTU4, OTUCn, ODU4";</li></ul>
rate-class	<ul style="list-style-type: none"><li>Tributary_rate_class_type:1G, 2.5G, 10G, 40G, 100G,150G, 200G, 250G, 300G, 400G, 500G, 600G, 700G, 800G, 900G, 1000G, 1100G</li></ul>	RW	M	Rounded bit rate of the tributary signal. Exact bit rate will be refined by protocol selection.
mapping	<ul style="list-style-type: none"><li>AMP, GMP, BMP, CBR, GFP_T, GFP_F</li></ul>	RO	M	Provided by OT
allocation	<ul style="list-style-type: none"><li>decimal64</li></ul>	RW	M	Allocation of the logical client channel to the tributary or sub-channel, expressed in Gbps. Please note that if the assignment is to an OTN logical channel, the allocation must be an integer multiplication to tributary-slot-granularity of the OTN logical channel.
otn/tributary_slot_granularity	TRIB_SLOT_1.25G, TRIB_SLOT_2.5G, TRIB_SLOT_5G			Base identity for tributary slot granularity for OTN logical channels.
logical-channel-type	PROT_ETHERNET, PROT_OTN			Type of protocol framing used on the logical channel or tributary

5.1.8 [Discovery-7] Discovery of line port capabilities (protocol, bit rate, available granularity and allocation)

This operation gets the capabilities of line ports (protocol, bit rate, mapping and granularity available).

Netconf RPC call to obtain all the channels in the device

```
<get>
  <filter type="subtree">
    <terminal-device xmlns="http://openconfig.net/yang/terminal-device">
      <logical-channels>
        <channel/>
      </logical-channels>
    </terminal-device>
  </filter>
</get>
```

Check the index of channels and use the index of the channel assigned to the OCH/lineport. The controller must use the 0 operation to obtain the OCH associated with the line port.

Obtain the

```
<get>
  <filter type="subtree">
    <terminal-device xmlns="http://OpenConfig.net/yang/terminal-device">
      <logical-channels>
        <channel>
          <index>{{Index channel associated with OCH}}</index>
          <config/>
          <otn/>
          <logical-channel-assignments>
            <assignments>
              <config>
                <allocation/>
              </config>
            </assignments>
          </logical-channel-assignments>
        </channel>
      </logical-channels>
    </terminal-device>
  </filter>
</get>
```

**Note:** For any other channel within the device the same request can be used to know the bit rate, protocol, granularity, mapping and allocation

RPC reply

```
<data xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <terminal-device xmlns="http://OpenConfig.net/yang/terminal-device">
    <logical-channels>
      <channel>
        <index>20975681</index>
        <config>
          <index>20975681</index>
```



```

    <rate-class xmlns:oc-opt-
types="http://openconfig.net/yang/transport-types">oc-opt-
types:TRIB_RATE_200G</rate-class>
    <trib-protocol xmlns:oc-opt-
types="http://openconfig.net/yang/transport-types">oc-opt-
types:PROT_OTUCN</trib-protocol>
    <logical-channel-type xmlns:oc-opt-
types="http://openconfig.net/yang/transport-types">oc-opt-
types:PROT_OTN</logical-channel-type>
    </config>
    <otn>
    <config>
    <tributary-slot-granularity>TRIB_SLOT_5G</tributary-slot-
granularity>
    </config>
    </otn>
    <logical-channel-assignments>
    <assignments>
    <config>
    <allocation>200</allocation>
    </config>
    </assignments>
    </logical-channel-assignments>
    </channel>
    </logical-channels>
</terminal-device>

```

5.1.9 [Discovery-8] Discovery of frequency, target-out-power and operational mode configured in line port

This operation retrieves the frequency, output power and operational mode configured in line port. To do so, the controller must use the 0 operation to obtain the OCH associated with the line port. This operation uses that value to obtain the frequency and the mode-id configured in that port line.

RPC call example

```

<get>
  <filter type="subtree">
    <components xmlns="http://openconfig.net/yang/platform">
      <component>
        <name>OCH-1-4-L1</name>
        <oc-opt-term:optical-channel>
          <config>
            <frequency/>
            <operational-mode/>
            <target-output-power/>
          </config>
        </optical-channel>
      </component>
    </components>
  </filter>
</get>

```

RPC response example

```
<data xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <components xmlns="http://openconfig.net/yang/platform">
    <component>
      <name>OCH-1-4-L1</name>
      <optical-channel xmlns="http://openconfig.net/yang/terminal-device">
        <config>
          <frequency>194000000</frequency>
          <operational-mode>30</operational-mode>
          <target-output-power>-2.5</target-output-power>
        </config>
      </optical-channel>
    </component>
  </components>
</data>
```

/components/component/oc-opt-term:optical-channel				
Attribute	Allowed Values/Format	Mod	Sup	Notes
name	String	RO	M	• Provided by server
frequency	uint64	RW	M	• Frequency of the optical channel, expressed in MHz
operational-mode		RO	M	• Provided by server

5.1.10 [Discovery-9] Discovery of type of logical channel assignment (preset or flexible)

Once discovered the ports and capabilities of the ports in operation and Discovery of line port capabilities (protocol type, rate granularity and allocation) [Discovery-6] [Discovery-7]. This operation will discover the type of cross-connections within the device. The OTs can have the logical channel assignments (cross-connections) configured in either preset or flexible modes.

Since there is no field to indicate if cross connections are fixed (hardcoded) or not, we require to use the container properties to include this information. Using the name “CROSSCONNECTION” for the property and values preset or flexible to indicate the type of crossconnection

If future products will include OTN matrices, it will be key to find a way to obtain the logical channel assignment type and other information related to the type of crossconnection in a standard fashion.

**In the case of fixed assignment (preset) , the equipment will have all the logical channel assignments pre-configured (commissioning stage) and this information will be retrieved by the SDN-C.**

RPC call example to get the propertie

```
<get>
  <filter type="subtree">
    <components xmlns="http://openconfig.net/yang/platform">
      <component>
        <name>{{name_of_component_tye_optical_channel}}</name>
        <properties>
```

```

    <property>
      <state>
        <name>CROSS_CONNECTION</name>
      </state>
    </property>
  </prproperties>
</component>
</components>
</filter>
</get>
```

RPC reply example

```

<data xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <components xmlns="http://openconfig.net/yang/platform">
    <component>
      <name>{{name_of_component_tye_optical_channel}}</name>
      <properties>
        <property>
          <state>
            <name>CROSS_CONNECTION</name>
            <value>PRESET</value>
          </state>
        </property>
      </prproperties>
    </component>
  </components>
</data>
```

/component/name{}/properties/property				
Attribute	Allowed Values/Format	Mod	Sup	Notes
name	<ul style="list-style-type: none"><li>string</li></ul>	RO	M	Name for the property
value	<ul style="list-style-type: none"><li>String (preset or flexible)</li></ul>	RO	M	Indicate the type of assignment (preset or flexible) Provided by server

5.1.11 [Discovery-10] Discovery of logical channel-assignment

Frequently, the logical channel assignment is fixed/preset. In this case, the SDN-C can discover the preset logical channel assignments from the client port to the line port.

Logical channel elements may be assigned directly to optical channels for line-side transmission or it can be further groomed into additional stages of logical channel elements. The grooming can multiplex (i.e., split the current element into multiple elements in the subsequent stage) or de-multiplex (i.e., combine the current element with other elements into the same element in the subsequent stage) logical elements in each stage. Note that to support the ability to groom the logical elements, the list of logical channel elements should be populated with an entry for the logical elements at each stage, starting with the initial assignment from the respective client physical port. Each logical element assignment consists of a pointer to an element in the next stage, or to an optical channel, along with a bandwidth allocation for the corresponding assignment (e.g., to split or combine signal).

RPC call to obtain all logical channels



```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="101">
  <get>
    <filter type="subtree">
      <terminal-device xmlns="http://openconfig.net/yang/terminal-device">
        <logical-channels>
          <channel>
            <index/>
          </channel>
        </logical-channels>
      </terminal-device>
    </filter>
  </get>
</rpc>
```

RPC reply should be all the index of the logical-channels

RPC call example to obtain the assignment of a logical channel

```
<get>
  <filter type="subtree">
    <terminal-device xmlns="http://openconfig.net/yang/terminal-device">
      <logical-channels>
        <channel>
          <index>"channel_index"</index>
          <logical-channel-assignments>
            <assignment>
              <config>
                <logical-channel/>
              </config>
            </assignment>
          </logical-channel-assignments>
        </channel>
      </logical-channels>
    </terminal-device>
  </filter>
</get>
```

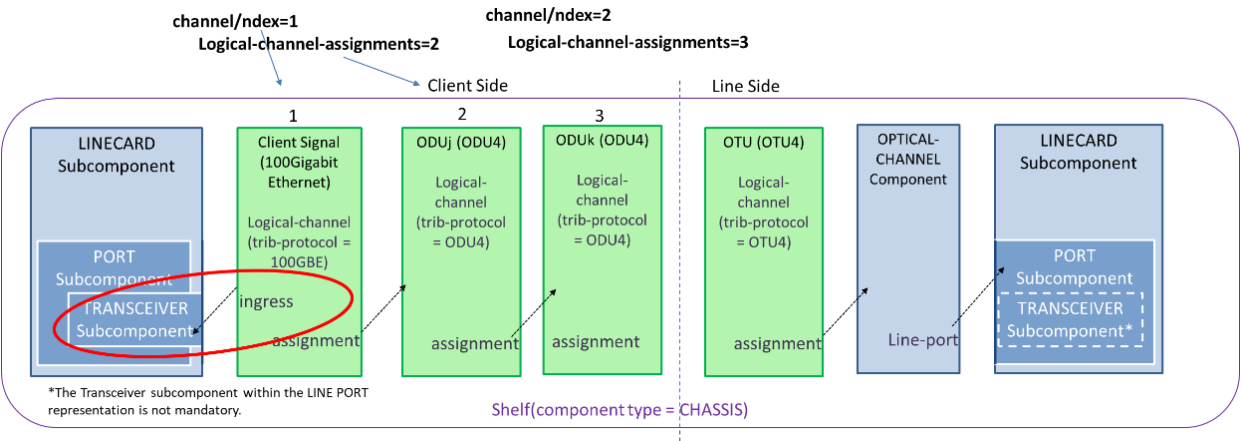


Figure 8: Modules retrieved in the discovery of logical channel-assignment

RPC reply

```
<data xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <terminal-device xmlns="http://OpenConfig.net/yang/terminal-device">
    <logical-channels>
      <channel>
        <index>20975681</index>
        <logical-channel-assignments>
          <assignments>
            <config>
              <logical-channel>25655655</logical-channel>
            </config>
          </assignments>
        </logical-channel-assignments>
      </channel>
    </logical-channels>
  </terminal-device>
```

The procedure should be:

1. Start from the transceiver component that is included in the ingress port, and check its logical-channel assignment (index of the next logical channel)
2. Note that the logical channels can be assigned to another logical channel or to an optical-channel. In this latter case, the assignment-type is OPTICAL\_CHANNEL which is typically a sub-component of a LINE port.

5.1.12 [Discovery-11] Discovery of location of a component and mapping to INVENTORY\_ID (TAPI)

Figure 9 shows the relative position of each “equipment” (chassis, slot, subplot, port) in a graphical representation in OpenConfig model.

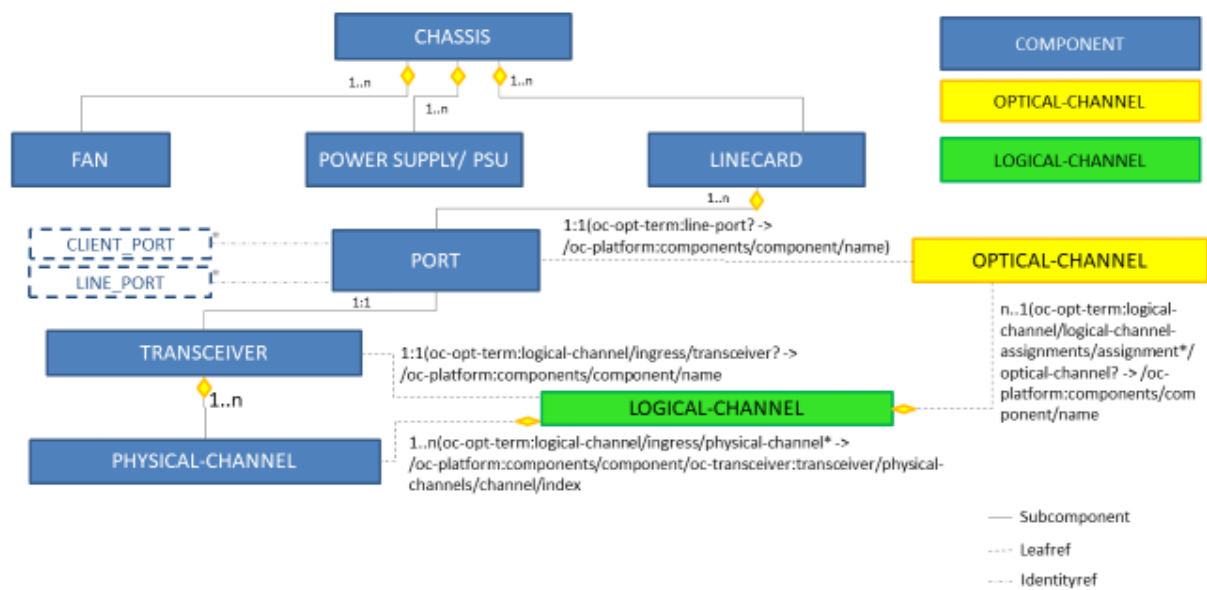


Figure 9: Diagram of HW components relationship within OpenConfig model

The names structure of an equipment and the naming of the different components are very similar in both models (OpenConfig and TAPI).

The mapping between TAPI and OpenConfig models for the different type of component is the following:

- Chassis=SUBRACK=CHASSIS
- Card in slot= CIRCUIT\_PACK=LINECARD or FAN or PSU
- Card in subslot= CIRCUIT\_PACK= LINECARD or FAN or PSU.
- Port= SMALL\_FORMFACTOR\_PLUGGABLE=PORT and TRANSCEIVER

Once we have the equivalence of the component in both models we need to specify the relative location of the component. In TAPI is used “holder location”.

The translation of the location from the OpenConfig model will be done in a first step to the holder-location in TAPI model and secondly to INVENTORY\_ID

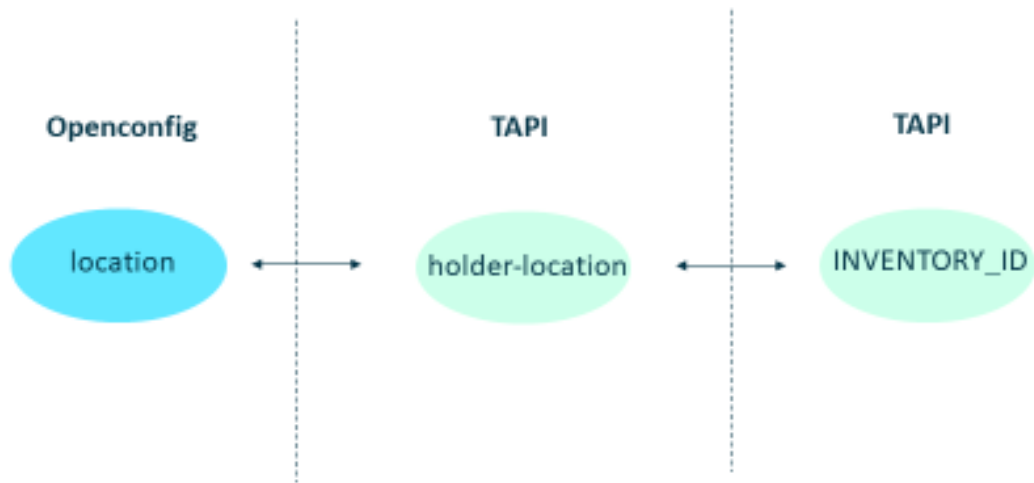


Figure 10: Relationship between locations parameters from OpenConfig to TAPI

RPC call example

```
<get>
  <filter type="subtree">
    <components xmlns="http://openconfig.net/yang/platform">
      <component>
        <name>FAN-1-33</name>
        <state>
          <location/>
          <parent/>
        </state>
      </component>
    </components>
  </filter>
</get>
```

RPC reply example

```
CHASSIS 1
Openconfig-----
<component>
  <name>CHASSIS-1</name>
  <state>
    <name>CHASSIS-1</name>
    <location>1<location>

TAPI-----
/tapi-common:context/tapi-equipment:physical-context/equipment/contained-
holder/actual-holder/common-holder-properties/holder-location:"1"
name": [{"value_name": "INVENTORY_ID", "value": "/ne=MadridNorte/r=1/sh=1" }]

+++++

CARD in SLOT 5
Openconfig-----

</component>
  <component>
    <name> LINECARD-1-5</name>
    <state>
      <name>LINECARD-1-5</name>
      .....
      <parent>CHASSIS-1</parent>
      <location>5<location>

TAPI-----
"holder-location": "1-5-0"
name": [{"value_name": "INVENTORY_ID", "value":
"/ne=MadridNorte/r=1/sh=1/sl=5" }]
```



5.1.13 [Discovery-12] Retrieval of global structure of HW inventory

The mandatory parameter requirements for the retrieving HW inventory information are the following:

- Part number=<part-no>
- Serial number=<serial-no>
- Name=<name>
- Description=<description>
- Version of the component=<software-version>
- Type=<type>
- Relative position of the component into the network element. This information must be discovered through the OpenConfig Platform model =<location>
- Operational state=<oper-status>
- Manufacturer=<mfg-name>
- Removable=<removable>
- Used power=<used-power>

component/state				
Attribute	Allowed Values/Format	Mod	Sup	Notes
part-no	String	RO	M	System-assigned part number for the component. This should be present in particular if the component is also an FRU (field replaceable unit)
Serial-no	String	RO	M	Provided by tapi-server. System-assigned serial number of the component.
description	String	RO	M	Description of the component
software-version	String	RO	M	For software components such as operating system or other software module, this is the version of the currently running software.
type	{ "base": "oc-platform-types:OPENCONFIG_HARDWARE_COMPONENT", "options": [ "oc-platform-types:BACKPLANE", "oc-platform-types:CHASSIS", "oc-platform-types:CONTROLLER_CARD", "oc-platform-types:CPU", "oc-platform-types:FABRIC", "oc-platform-types:FAN", "oc-platform-types:FRU", "oc-platform-	RO	M	Type of component as identified by the system

	types:INTEGRATED_CIRCUIT","oc-platform-types:LINECARD","oc-opt-types:OPTICAL_CHANNEL","oc-platform-types:PORT","oc-platform-types:POWER_SUPPLY","oc-platform-types:SENSOR","oc-platform-types:STORAGE","oc-platform-types:TRANSCEIVER"],"datatype":"identityref"}			
location	String	RO	M	System-supplied description of the location of the component within the system
oper-status	oc-platform-types:COMPONENT_OPERATOR_STATUS	RO	M	If applicable, this reports the current operational status of the component.
mfg-name	String	RO	M	System-supplied identifier for the manufacturer of the component
removable	boolean	RO	M	Pluggable or removable
used-power	Unit32	RO	M	Actual power used by the component

The basic structure of a network elements is represented in the following picture:

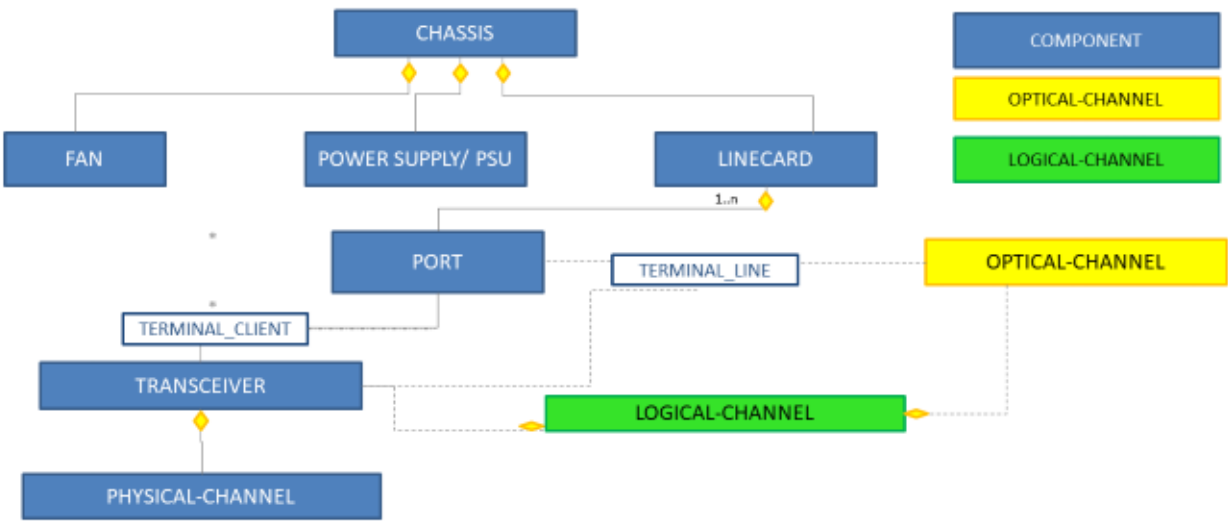


Figure 11: Parts of a network elements and their relationship

The relationship between a component parent and a subcomponent is represented in two ways:

- Using a subcomponent leaf within the component. Only one level of subcomponent requested



- Using parent parameter to indicate the relation with the parent component.

**RPC procedure**

These are the steps to obtain the device structure of components:

1. Discovery of all components by using 0
2. Discovery of type components by using 5.1.2
3. Retrieval of the chassis structure to obtain the linecards/psu/fan/power supply, etc. from chassis.

**Note: Repeat this step if there is more than one chassis within the device**

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="101">
  <get>
    <filter type="subtree">
      <components xmlns="http://openconfig.net/yang/platform">
        <component>
          <name>{{name of component_type chassis}}</name>
          <subcomponents/>
        </component>
      </components>
    </filter>
  </get>
</rpc>
```

4. Retrieval of the linecard structure to obtain ports

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="101">
  <get>
    <filter type="subtree">
      <components xmlns="http://openconfig.net/yang/platform">
        <component>
          <name>{{name_of_component_type_linecard}}</name>
          <subcomponents/>
        </component>
      </components>
    </filter>
  </get>
</rpc>
```

5. Retrieval of ports structure to obtain the transceiver

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="101">
  <get>
    <filter type="subtree">
      <components xmlns="http://openconfig.net/yang/platform">
        <component>
          <name>{{port_name}}</name>
          <subcomponents/>
        </component>
      </components>
    </filter>
  </get>
</rpc>
```

**RPC reply example of CHASSIS**

```
<data xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
```



```
<components xmlns="http://openconfig.net/yang/platform">
  <component>
    <name>CHASSIS-1</name>
    <subcomponents>
      <subcomponent>
        <name>LINECARD-1-1</name>
        <config>
          <name>LINECARD-1-1</name>
        </config>
        <state>
          <name>LINECARD-1-1</name>
        </state>
      </subcomponent>
      <subcomponent>
        <name>LINECARD-1-3</name>
        <config>
          <name>LINECARD-1-3</name>
        </config>
        <state>
          <name>LINECARD-1-3</name>
        </state>
      </subcomponent>
      <subcomponent>
        <name>LINECARD-1-2</name>
        <config>
          <name>LINECARD-1-2</name>
        </config>
        <state>
          <name>LINECARD-1-2</name>
        </state>
      </subcomponent>
      <subcomponent>
        <name>LINECARD-1-4</name>
        <config>
          <name>LINECARD-1-4</name>
        </config>
        <state>
          <name>LINECARD-1-4</name>
        </state>
      </subcomponent>
      <subcomponent>
        <name>PSU-1-21</name>
        <config>
          <name>PSU-1-21</name>
        </config>
        <state>
          <name>PSU-1-21</name>
        </state>
      </subcomponent>
      <subcomponent>
        <name>PSU-1-22</name>
        <config>
          <name>PSU-1-22</name>
        </config>
        <state>
          <name>PSU-1-22</name>
        </state>
      </subcomponent>
      <subcomponent>
        <name>FAN-1-33</name>
        <config>
          <name>FAN-1-33</name>
        </config>
      </subcomponent>
    </subcomponents>
  </component>
</components>
```





```

    </config>
    <state>
      <name>FAN-1-33</name>
    </state>
  </subcomponent>
  <subcomponent>
    <name>FAN-1-32</name>
    <config>
      <name>FAN-1-32</name>
    </config>
    <state>
      <name>FAN-1-32</name>
    </state>
  </subcomponent>
  <subcomponent>
    <name>FAN-1-31</name>
    <config>
      <name>FAN-1-31</name>
    </config>
    <state>
      <name>FAN-1-31</name>
    </state>
  </subcomponent>
  <subcomponent>
    <name>CU-1-41</name>
    <config>
      <name>CU-1-41</name>
    </config>
    <state>
      <name>CU-1-41</name>
    </state>
  </subcomponent>
</subcomponents>
</component>
</components>
</data>
```

After these three calls the SDN-C must be able to get the full relation of the components. An example of the representation is represented bellow.

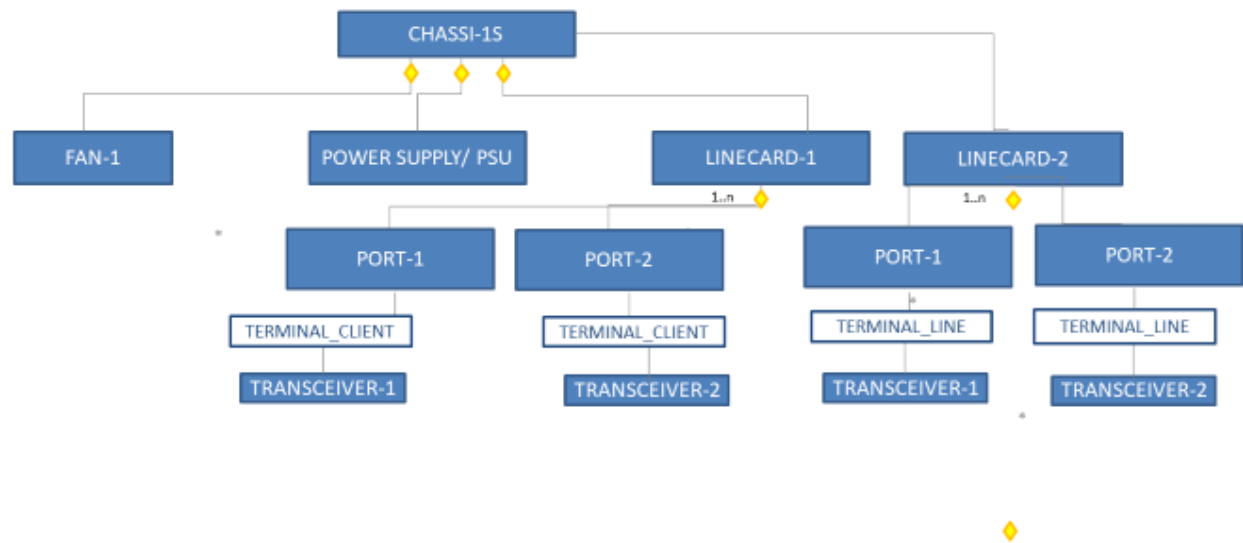


Figure 12: Example of the component representation.

5.1.14 [Discovery-13] Retrieval of HW inventory information of each component

Once the SDN-C has discovered the full structure of components, the following call will be used to obtain the specific inventory information of each component:

Netconf RPC call example

```
<get>
  <filter type="subtree">
    <components xmlns="http://openconfig.net/yang/platform">
      <component>
        <name>{{name_of_component}}</name>
        <state/>
      </component>
    </components>
  </filter>
</get>
```

Component example of CHASSIS

```
<component>
  <name>CHASSIS-1</name>
  <config>
    <name>CHASSIS-1</name>
  </config>
  <state>
    <type>CHASSIS</type>
    <id>0x10100000000000</id>
    <name>CHASSIS-1</name>
    <description>Cloud OTN Subrack 2U</description>
```

```

    <mfg-name>XXXX</mfg-name>
    <software-version>11111<hardware-version>
    <serial-no>JN1226D32AHA</serial-no>
    <part-no>11111</part-no>
    <location>1<location>
    <oper-status>ACTIVE</oper-status>
    <mfg-name>ACME<mfg-name>
  </state>
</component>

```

Card inventory structure (FAN or PSU)

Any subcomponent or component must use same structure for the mandatories inventory parameters. **Location must be used to indicate the relative position, the parent where the component is included and the subcomponent included in a component,**

Note: Currently the vendors usually use the name to indicate the location. Eg Linecard 1-3 represent the relative position within the NE (chasis-1, slot 3). FAN-1-31 indicate in this case chassis 1, slot 31. This implementation is not allowed. Location, parent and mainly subcomponent must be used to indicate this information.

**Is mandatory use the location parameter to indicate the position of the component within the system and the subcomponents and parent to indicate the relation between components.**

```

<component>
  <name>FAN-1-31</name>
  <config>
    <name>FAN-1-31</name>
  </config>
  <state>
    <type>FAN</type>
    <id>0x5010300000000000</id>
    <description>Fan Control</description>
    <mfg-name>XXXX</mfg-name>
    <software-version>Vxx @NCPU</software-version>
    <serial-no>716188800022</serial-no>
    <part-no>180000351225</part-no>
    <location>31<location>
    <parent>CHASSIS-1<parent>
    <temperature>
      <avg>38.7</avg>
      <instant>39.0</instant>
    </temperature>
  </state>
</component>

```

component/state				
Attribute	Allowed Values/Format	Mod	Sup	Notes
part-no	String	RO	M	System-assigned part number for the component. This should be present in particular if the component is also an FRU (field replaceable unit)
serial-no	String	RO	M	Provided by server. System-assigned serial number of the

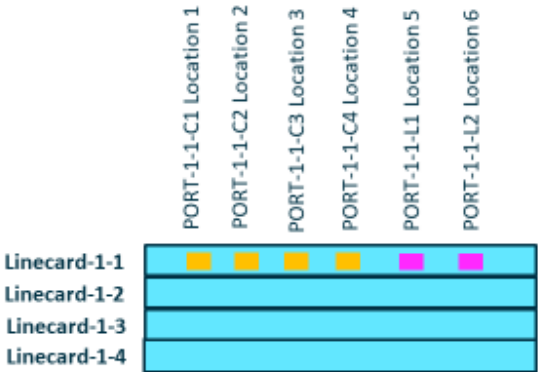
				<i>component.</i>
description	String	RO	M	
software-version	String	RO	M	<i>For software components such as operating system or other software module, this is the version of the currently running software.</i>
type	{ "base": "oc-platform-types:OPENCONFIG_HARDWARE_COMPONENT", "options": [ "oc-platform-types:BACKPLANE", "oc-platform-types:CHASSIS", "oc-platform-types:CONTROLLER_CARD", "oc-platform-types:CPU", "oc-platform-types:FABRIC", "oc-platform-types:FAN", "oc-platform-types:FRU", "oc-platform-types:INTEGRATED_CIRCUIT", "oc-platform-types:LINECARD", "oc-opt-types:OPTICAL_CHANNEL", "oc-platform-types:PORT", "oc-platform-types:POWER_SUPPLY", "oc-platform-types:SENSOR", "oc-platform-types:STORAGE", "oc-platform-types:TRANSCEIVER"], "datatype": "identityref" }	RO	M	Type of component as identified by the system
location	String	RO	M	System-supplied description of the location of the component within the system
oper-status	oc-platform-types:COMPONENT_OPERATOR_STATUS ["ACTIVE", "INACTIVE", "DISABLED"]	RO	M	If applicable, this reports the current operational status of the component.
mfg-name	String	RO	M	System-supplied identifier for the manufacturer of the component
removable	boolean	RO	M	
used-power	Unit32	RO	M	

5.1.15 [Discovery-14] Slot-id information

The LINE-CARD component can also report the relative location using the slot-id. In that case the slot-id must be equal to the location parameter.

Note: Currently the vendors usually use the name to indicate the location. Eg Linecard 1-3 represent the relative position within the NE (chasis-1, slot 3). FAN-1-31 indicate in this case chassis 1, slot 31. As mentioned before, this implementation is not allowed.

The following figure shows the equipment used in these examples:



/components/component/oc-linecard:linecard/state				
Attribute	Allowed Values/Format	Mod	Sup	Notes
power-admin-state	oc-platform-types:component-power-type  POWER_ENABLED or POWER_DISABLED	RO	M	Provided by server Request state of power within the linecard
slot-id	String	RO	M	Provided by server Identifier for the slot or chassis position in which the linecard is installed

The linecard component uses an extra parameter (slot-id). Slot-id must be the same as the location parameter.

Card inventory structure (Linecard)

```
<component>
  <name>LINECARD-1-4</name>
  ...
  <linecard xmlns="http://openconfig.net/yang/platform/linecard">
    <config>
      <power-admin-state
xmlns="http://openconfig.net/yang/platform/linecard">POWER_ENABLED</power-admin-state>
    </config>
    <state>
```

```

    <power-admin-state
xmlns="http://openconfig.net/yang/platform/linecard">POWER_ENABLED</power-
admin-state>
    <slot-id
xmlns="http://openconfig.net/yang/platform/linecard">4</slot-id>
    </state>
</component>

```

## 5.2 Provisioning in devices with preset logical channels

Following list contains the provisioning in devices with preset logical channels:

- [Provisioning-1]** Provisioning of admin-state enable on client port
- [Provisioning-2]** Provisioning of admin-state enable on line-port
- [Provisioning-3]** Provisioning of a frequency in line port
- [Provisioning-4]** Provisioning of power line in line port
- [Provisioning-5]** Provisioning of operational mode-id (line capabilities)

### 5.2.1 [Provisioning-1] Provisioning of admin-state enable of client port

This operation consists in the configuration needed to enable the client port. It is done after the operation [Discovery-2], where the components are discovered.

#### RPC call example

```

<edit-config>
  <target>
    <{{target}}/>
  </target>
  <config>
    <components xmlns="http://openconfig.net/yang/platform">
      <component>
        <name>{{client-port-name}}</name>
        <port>
          <optical-port>
            <config>
              <admin-state>ENABLED/admin-state>
            </config>
          </optical-port>
        </port>
      </component>
    </components>
  </config>
</edit-config>

```

Result: The optical port will change the enabled state.

GNMI call example

```
gNMI_cli -address {server_ip_address:port} \
  -set \
  -proto "update:<path: <elem: <name: 'components'> elem: <name:
'component' key<key:'name' value:'{{client-port-name}}'>> elem: <name:
'port'> elem: <name: 'optical-port'> <elem: <name: 'config'> <elem: <name:
'admin-state'> > val: <string_val: 'enabled'>>" \
  -timeout 5s \
  -alsologtostderr \
  -client_cert certs/client1.crt \
  -client_key certs/client1.key \
  -ca_cert certs/onfca.crt
```

**Important note:** This action is the only one required to enable the client and all the possible subcomponents of the port (for example, the transceiver within the port). In case of the transceivers belonging to the port, they must change the admin state in coordination with the port and report the same status than the line port

/components/component/port/oc-line-com:optical-port/oc-line-com:config				
Attribute	Allowed Values/Format	Mod	Sup	Notes
admin-state	["ENABLED", "DISABLED", "MAINT"]	RW	M	Provided by <i>server or user</i>

5.2.2 [Provisioning-2] Provisioning of admin-state enable of line port

This operation consists in the configuration needed to enable the line-port It is done after the operation [Discovery-2], where the components are discovered.

RPC call

```
<edit-config>
  <target>
    <{{target}}/>
  </target>
  <config>
    <components xmlns="http://openconfig.net/yang/platform">
      <component>
        <name>{{line-port-name}}</name>
        <port>
          <optical-port>
            <config>
              <admin-state>ENABLED</admin-state>
            </config>
          </optical-port>
        </port>
      </component>
    </components>
  </config>
</edit-config>
```

/components/component/port/oc-line-com:optical-port/oc-line-com:state				
Attribute	Allowed Values/Format	Mod	Sup	Notes
admin-state	["ENABLED", "DISABLED", "MAINT"]	RW	M	Provided by server or user

**Important note:** This action is the only one required to enable the line port and all the possible subcomponents of the port (for example the transceiver within the port). In case of the transceivers subcomponent of the port, it must change the admin state in coordination with the port and report the same status than the line port



5.2.3 [Provisioning-3] Provisioning of frequency in line port

This operation consists in the configuration needed to provision the frequency in the line side.

To obtain the OCH\_component name used previously [Discovery-3] to obtain the name of the subcomponent of the line ports.

RPC call

```
<edit-config>
  <target>
    <{{target}}/>
  </target>
  <config>
    <components xmlns="http://openconfig.net/yang/platform">
      <component>
        <name>{{och_component_name}}</name>
        <optical-channel xmlns="http://openconfig.net/yang/terminal-device">
          <config>
            <frequency>{{freq_value}}</frequency>
          </config>
        </optical-channel>
      </component>
    </components>
  </config>
</edit-config>
```

Terminal- device/optical- channel/config				
Attribute	Allowed Values/Format	Mod	Sup	Notes
frequency	uint64	RW	M	Frequency of the optical channel, expressed in MHz

**Result:** If the frequency is not available by the OT an error message must be sent by the OT to the server according to RFC 6241.

5.2.4 [Provisioning-4] Provisioning of optical output power in line port.

This operation consists in the call to provisioning the optical output power in the line port

RPC call

```
<edit-config>
  <target>
    <{{target}}/>
  </target>
  <config>
    <components xmlns="http://openconfig.net/yang/platform">
      <component>
        <name>{{och_component_name}}</name>
        <optical-channel xmlns="http://openconfig.net/yang/terminal-device">
```

```
<config>
  <target-output-power>{{output_power}}</target-output-power>
</config>
</optical-channel>
</component>
</components>
</config>
</edit-config>
```

Terminal-device/optical-channel				
Attribute	Allowed Values/Format	Mod	Sup	Notes
Target-output-power	Decimal64	RW	M	"Target output optical power level of the optical channel in dBm, expressed in increments of 0.01 dBm (decibel-milliwatts)

Result: If the optical power is not available by the OT an error message must be sent by the OT to the server.

**Note: To know the range of the output power, use the mode-id call**

5.2.5 [Provisioning-5] Provisioning of operational mode in port line

This operation consists in the call to provisioning the operational mode in the line port. The operational mode defines the modulation and the main characteristics of the line port.

**Note:** the operational mode will automatically configure all the channel and assignments in the network side. From optical channel to ODU HO. See use case 1a of MS-DIS 2.5 and Figure 13

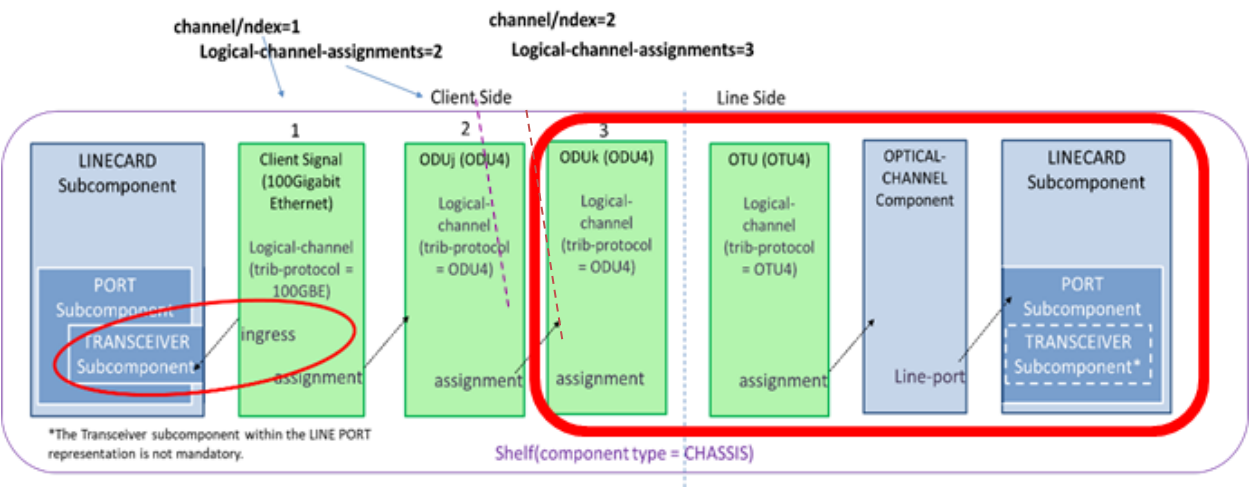


Figure 13: Logical channel created automatically with the operational mode

RPC call example

```
<edit-config>
  <target>
    <{{target}}/>
  </target>
  <config>
    <components xmlns="http://openconfig.net/yang/platform">
      <component>
        <name>{{och_component_name}}</name>
        <optical-channel xmlns="http://openconfig.net/yang/terminal-device">
          <config>
            <operational-mode>{{mode-id}}</operational-mode>
          </config>
        </optical-channel>
      </component>
    </components>
  </config>
</edit-config>
```

Terminal- device/optical - channel/config				
Attribute	Allowed Values/Format	Mod	Sup	Notes

operational-mode	uint16	RW	M	Vendor-specific mode identifier -- sets the operational mode for the channel. The specified operational mode must exist in the list of supported operational modes supplied by the device
------------------	--------	----	---	---

Result: If the operational-mode is not available by the OT, an error message must be sent by the OT to the server.

### 5.3 Provisioning in devices with flexible/configurable logical channels (no preset)

This chapter describes the provisioning operations for device without preset logical channels. This operation only made sense when OTN matrix are present. The operations are the following:

- [Provisioning-6]

Basic configuration of a logical channels
- [Provisioning-7]

Assignment of client port to a flexible logical-channel (OTN matrix)
- [Provisioning-8]

Assignment of logical-channel into another logical-channel (OTN matrix)
- [Provisioning-9]

Assignment of one logical-channel into 2 or more logical-channels and vice versa (OTN matrix)
- [Provisioning-10]

Assignment of logical-channel into an optical channel
- [Provisioning-11]

Configuration of a bit rate (rate-class) and protocol (trib-protol and logical channel type) on client port

5.3.1 [Provisionig-6] Provisioning in devices with flexible/configurable logical channels (no preset)

TIP MUST consortium requests that the suppliers preset all the basic logical channels configuration when the operational mode is configured (see chapter 4 Initial state of device)

In that case the user only needs to create the cross-connections (logical-channel-assignments) between the channels and change the admin state to enable or disable.

The procedure is the following:

- 1. Obtain all logical channels, as done in operation [Discovery-10].
- 2. Change admin state to enable of a logical channel:

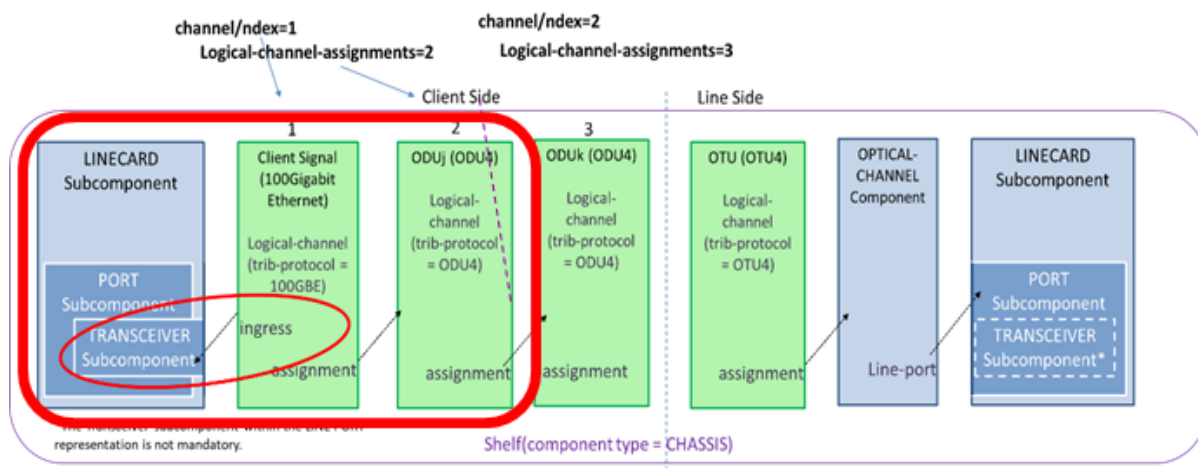
RPC call example

```
<edit-config>
  <target>
    <{{target}}/>
  </target>
  <config>
    <terminal-device xmlns="http://openconfig.net/yang/terminal-device">
      <logical-channels>
        <channel>
          <index>1</index>
          <config>
            <admin-state>ENABLED</admin-state>
          </config>
        </channel>
      </logical-channels>
    </terminal-device>
  </config>
</edit-config>
```

5.3.2 [Provisionig-7] Assignment of client port to a flexible logical-channel (OTN matrix)

This operation consists in the configuration needed to assign a port to a logical channel.

**Note: the granularity is considered as not configurable in this first document and must be discovered by the server**



**Figure 14: Logical channels and assignments created when transceivers are inserted**

**Important Note:** This configuration and the assignment of the logical channel until the ODU-LO must be created automatically when the transceiver (pluggable) is inserted. Otherwise, must be pre-commissioned by the supplier.

This procedure should be done during the commissioning stage (Initial state of device):

1. Create the logical channels
2. Start from the transceiver in the ingress and set the logical channel assignment to the (index of the next logical channel) using the index of the channel as the key
3. Assign the rest of logical channels until ODU-LO channel using index as key

### RPC call example step 1

```
<edit-config>
  <target>
    <{{target}}/>
  </target>
  <config>
    <terminal-device xmlns="http://openconfig.net/yang/terminal-device">
      <logical-channels>
        <channel>
          <index>1</index>
          <ingress>
            <config>
              <transceiver>{{transceiver_client}}</transceiver>
            </config>
          </ingress>
        </channel>
      </logical-channels>
    </terminal-device>
  </config>
</edit-config>
```

Terminal- device/optical-channel				
Attribute	Allowed	Mod	Sup	Notes

	Values/Format			
Ingress	Leafref	RW	M	"Reference to the transceiver carrying the input signal for the logical channel. If specific physical channels are mapped to the logical channel (as opposed to all physical channels carried by the transceiver), they can be specified in the list of physical channel references"

5.3.3 [Provisioning-8] Assignment of logical channel in another logical channel (cross connection)

This operation consists in the configuration needed to assign a logical channel from top to bottom until reach the optical channel (line port).

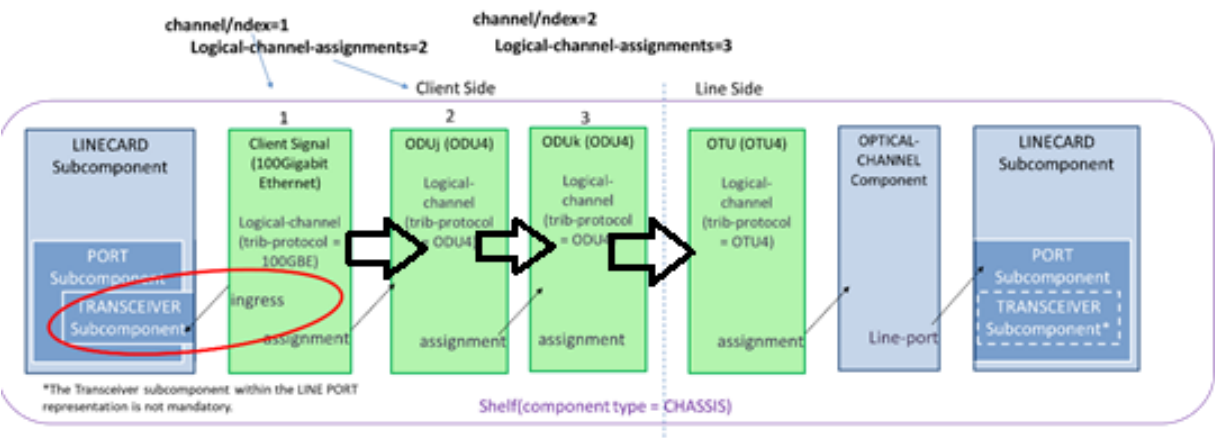
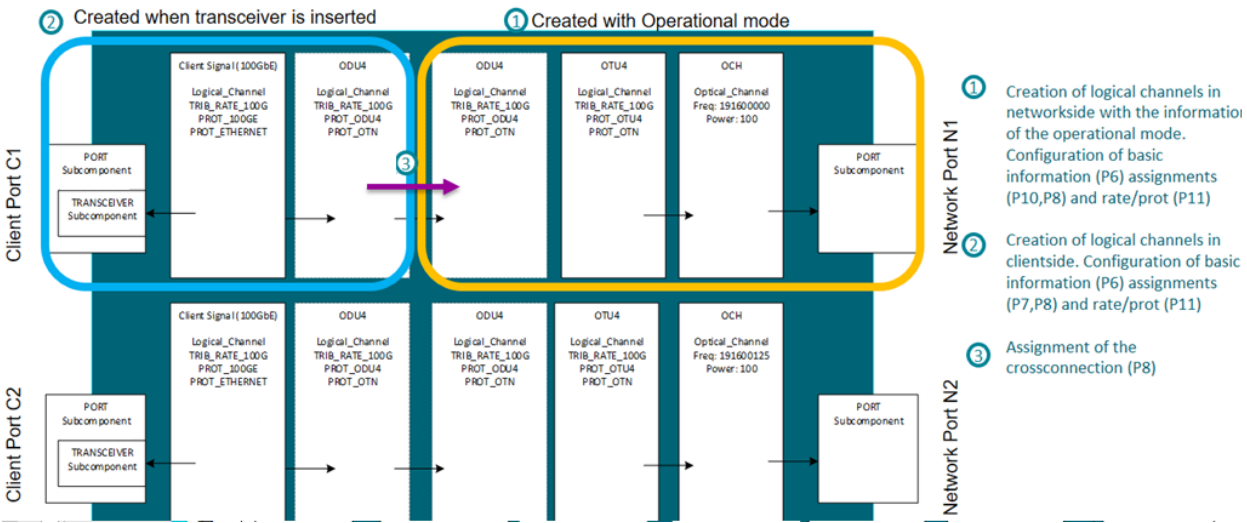


Figure 15: Assignment of logical channel into another (cross-connection)

**Note:** This assignment of the logical channel must be created if the device is not preset. Otherwise, the assignment must be created in the commissioning stage

The procedure should be the following in case of a not preset device (**only step 3**).



RPC call example

```
<edit-config>
  <target>
    <{{target}}/>
  </target>
  <config>
    <terminal-device xmlns="http://openconfig.net/yang/terminal-device">
      <logical-channels>
        <channel>
          <index>1</index>
          <admin-state>ENABLED</admin-state>
          <logical-channel-assignment>
            <assignment>
              <config>
                <logical-channel>2</logical-channel>
                <assignment-type>LOGICAL_CHANNEL</assignment-type>
                <allocation>{{ALLOCATION_IN_GBPS}}</allocation>
                <tributary-slot-index>0</tributary-slot-index>
                <mapping>GMP</mapping>
              </config>
            </assignment>
          </logical-channel-assignment>
        </channel>
      </logical-channels>
    </terminal-device>
  </config>
</edit-config>
```

5.3.4 [Provisioning-9] Assignment of a logical channel into many logical channels and vice versa (optional)

This operation is only needed if the logical structure of the device needs to assign more than one logical channel to construct the final service. Eg Muxponder of 5x40 into 2 lambdas of 100G



The operation is focus on the assignment of 1 logical channel into 2 or more logical channel. The following figure shows an example:

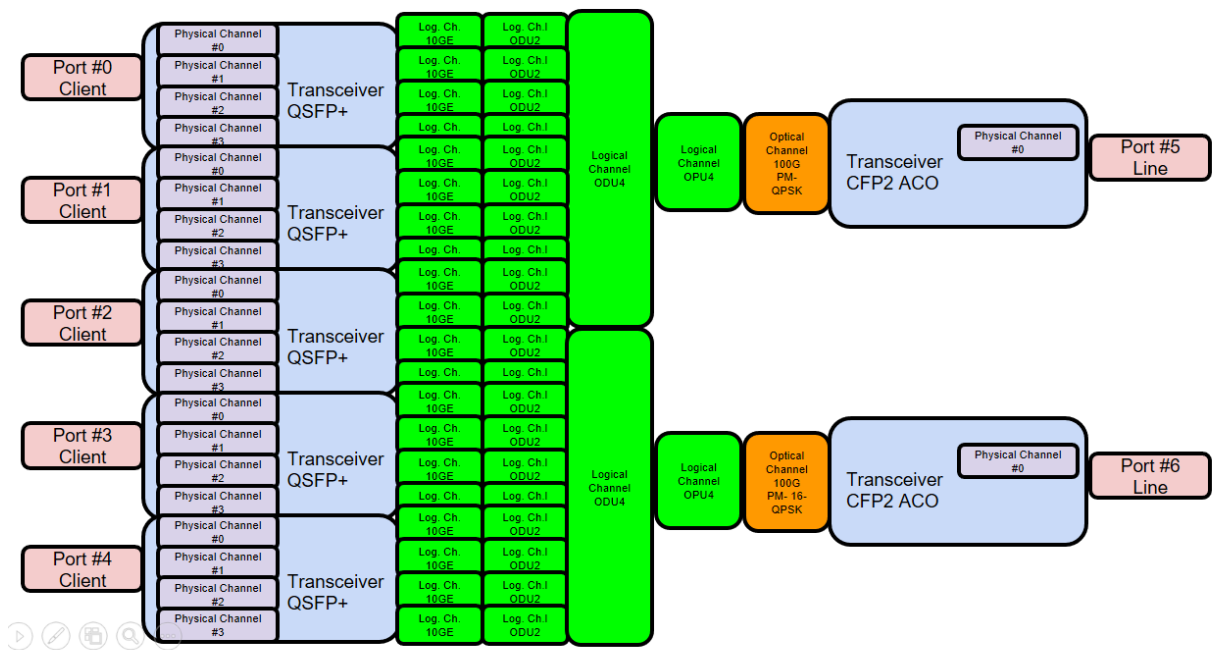


Figure 16: Device with multiplexing of logical channels

There are two types of assignments of one logical channel in more than one optical channel:

Demultiplex operation

This operation required to repeat 5.3.3 as many times as needed reusing the same channel id and assigning it to various logical channels.

Multiplex operation

This operation required to repeat 5.3.3 as many times as needed assigning different channel to a unique logical channel.

5.3.5 [Provisioning-10] Assignment of a logical channel into optical channel (line-port)

This operation finishes the internal cross-connection of the transponder assigning the optical channel associated with the line port (the OCH is a subcomponent of the line port).

**Note:** Setting the operational mode must automatically configure all the channel and assignments in the network side. From optical channel to ODU HO. This operation must be done in the commissioning stage (Initial state of device). The rest of parameters (frequency, power) will be provided by the controller

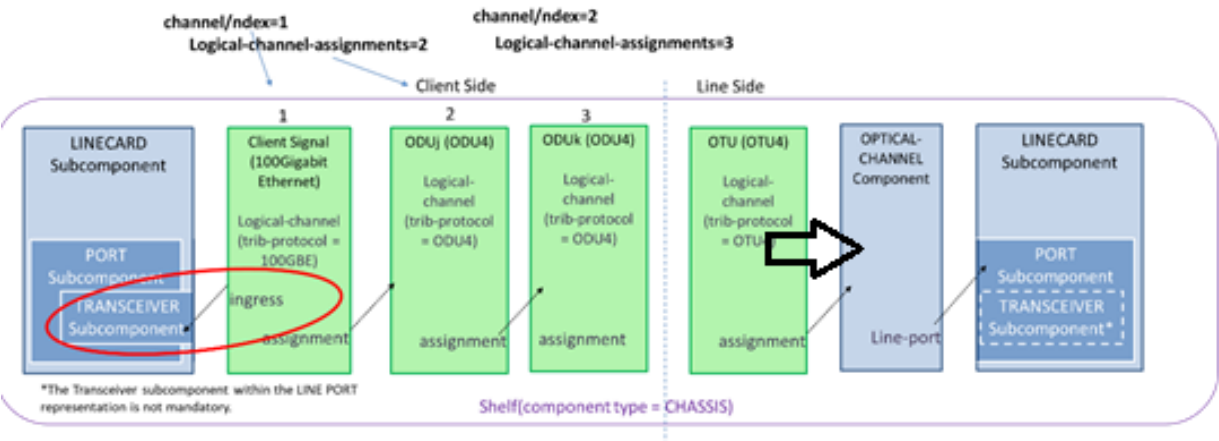


Figure 17: Assignment of logical channel into optical channel

RPC call

```
<edit-config>
  <target>
    <{{target}}/>
  </target>
  <config>
    <terminal-device xmlns="http://openconfig.net/yang/terminal-device">
      <logical-channels>
        <channel>
          <index>1</index>
          <logical-channel-assignment>
            <assignment>
              <config>
                <optical-channel>OCH_name</optical-channel>
                <assignment-type>OPTICAL_CHANNEL</assignment-type>
                <allocation>{{ALLOCATION_IN_GBPS}}</allocation>
              </config>
            </assignment>
          </logical-channel-assignment>
        </channel>
      </logical-channels>
    </terminal-device>
  </config>
</edit-config>
```

5.3.6 [Provisioning-11] Configuration of rate (rate-class), protocol (trib-protocol and logical channel type) on client port

We request that the suppliers preset all the basic logical channels configuration. Only the bit rate and the protocol used for the client port must be configured by the controller.

**Note:** These parameters are automatically configured with the operational mode selection in the line side, thus the configuration is only necessary in the client side

The following call set the bit rate (rate-class), the protocol (trib-protocol) and logical-channel-type in the client side (assuming that channel with index 1 is the channel associated with the client port to be configured)

RPC call

```
<edit-config>
  <target>
    <{{target}}/>
  </target>
  <config>
    <terminal-device xmlns="http://openconfig.net/yang/terminal-device">
      <logical-channels>
        <channel>
          <index>1</index>
          <config>
            <rate-class>TRIB_RATE_100G</rate-class>
            <trib-protocol>PROT_100GE</trib-protocol>
            <logical-channel-type>PROT_ETHERNET</logical-
channel-type>
          </config>
        </channel>
      </logical-channels>
    </terminal-device>
  </config>
</edit-config>
```

/terminal-device/logical-channels/channel				
Attribute	Allowed Values/Format	Mod	Sup	Notes
index	<ul style="list-style-type: none"><li>uint32</li></ul>	RO	M	<ul style="list-style-type: none"><li>Reference to the index of the logical channel</li></ul>
description	<ul style="list-style-type: none"><li>String (preset or flexible)</li></ul>	RO	M	<ul style="list-style-type: none"><li>Indicate the type of assignment (preset or flexible)</li><li>Provided by server</li></ul>
logical-channel-assignment/assignment/config/assignment-type	<ul style="list-style-type: none"><li>Type enumeration: LOGICAL_CHANNEL, OPTICAL_CHANNEL</li></ul>	RO	M	<ul style="list-style-type: none"><li>Provided by server</li><li>Subsequent channel is a logical channel";</li><li>"Subsequent channel is a optical channel / carrier</li></ul>
Logical-channel-assignments/assignment/config/logical-channel	<ul style="list-style-type: none"><li>Reference to the index of the logical-channel</li></ul>			<ul style="list-style-type: none"><li>Provided by tapi-server</li></ul>
Logical-channel-assignments/assignment config/optical-channel	<ul style="list-style-type: none"><li>Reference to the name of the optical-channel</li></ul>			<ul style="list-style-type: none"><li>Provided by tapi-server</li></ul>
Trib-protocol	<ul style="list-style-type: none"><li>PROT_1GE, OC48, STM16, 10GE_LAN, 10GE_WAN, OC192, STM64,</li></ul>			<ul style="list-style-type: none"><li>Base identity for protocol framing used by tributary signals.</li><li>Provided by tapi-server</li><li>rate class: 1G protocols: 1GE</li></ul>



	OTU2, OTU2E, OTU1E, ODU2, ODU2E, 40GE, OC768, STM256, OTU3, ODU3, 100GE, 100GE_MLG, OTU4, OTUCN, ODU4			<ul style="list-style-type: none"><li>rate class: 2.5G protocols: OC48, STM16</li><li>rate class: 10G protocols: 10GE LAN, 10GE WAN, OC192, STM64, OTU2, OTU2e, OTU1e, ODU2, ODU2e, ODU1e</li><li>rate class: 40G protocols: 40GE, OC768, STM256, OTU3, ODU3</li><li>rate class: 100G protocols: 100GE, 100G MLG, OTU4, OTUCn, ODU4";</li></ul>
Rate-class	<ul style="list-style-type: none"><li>Tributary_rate_class_type: 1G, 2.5G, 10G, 40G, 100G, 150G, 200G, 250G, 300G, 400G, 500G, 600G, 700G, 800G, 900G, 1000G, 1100G</li></ul>			<ul style="list-style-type: none"><li>Rounded bit rate of the tributary signal. Exact bit rate will be refined by protocol selection.</li></ul>
MAPPING	<ul style="list-style-type: none"><li>AMP, GMP, BMP, CBR, GFP_T, GFP_F</li></ul>			<ul style="list-style-type: none"><li>Provided_by server</li></ul>
Allocation	<ul style="list-style-type: none"><li>decimal64</li></ul>			<ul style="list-style-type: none"><li>Allocation of the logical client channel to the tributary or sub-channel, expressed in Gbps. Please note that if the assignment is to an OTN logical channel, the allocation must be an integer multiplication to tributary-slot-granularity of the OTN logical channel.</li></ul>
Otn/tributary_slot_granularity	TRIB_SLOT_1.25G, TRIB_SLOT_2.5G, TRIB_SLOT_5G			<ul style="list-style-type: none"><li>Base identity for tributary slot granularity for OTN logical channels.</li></ul>
Logical-channel-type	PROT_ETHERNET, PROT_OTN			<ul style="list-style-type: none"><li>Type of protocol framing used on the logical channel or tributary</li></ul>

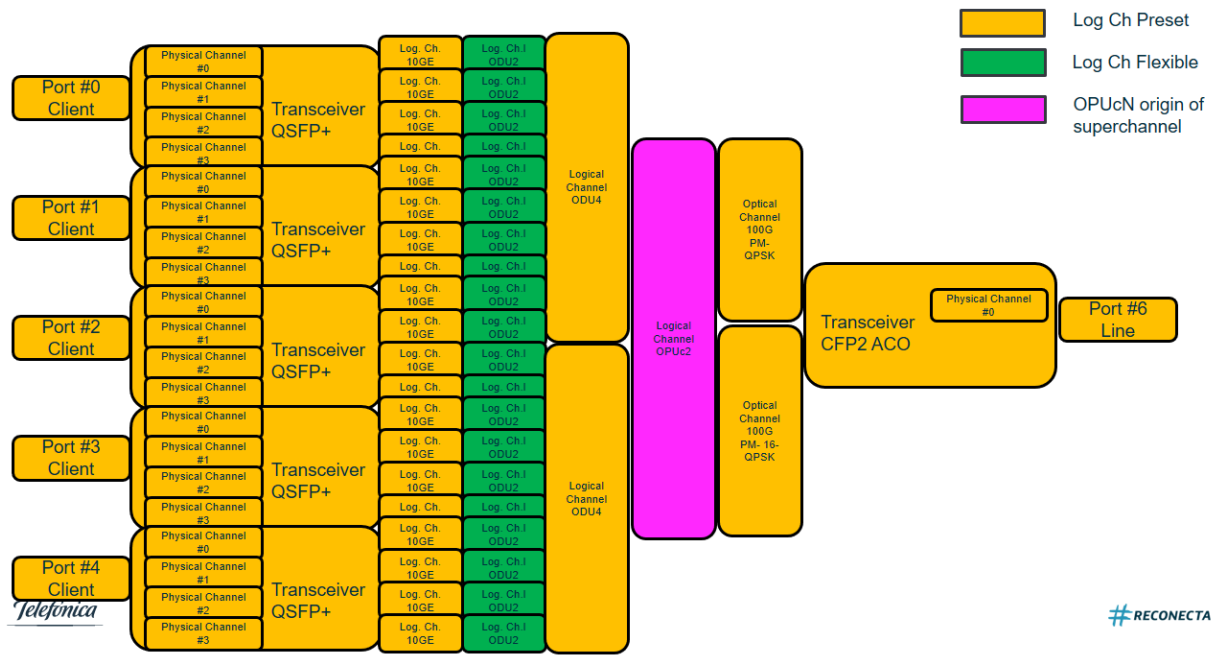
5.3.7 [Provisioning-12] Configuration of various optical channels into a superchannel (multiples optical channel into the same optical channel)

This operation configures a group of optical channels into the same super-channel. This use case triggers the generation of an ODUcN Top Connection (superchannel) which is realized by N number of OTSi Top Connections required to transport the service. The optical channel representing the sub channels can be associated to a unique port or to different ports

The following example shows 2 optical channel belonging to the same superchannel .

The key is the unique OTUcN logical channel, split in various optical channels and then multiplexed in an unique transceiver. The individual capabilities of each OTSi is included in the information of the operational-mode of each optical channel.

**Note: Some case required the use of an insert call to create the assignment and some attributes as the tributary-slot-index**



RPC call

```
<edit-config>
  <target>
    <{{target}}/>
  </target>
  <config>
    <terminal-device xmlns="http://openconfig.net/yang/terminal-device">
      <logical-channels>
        <channel>
          <index>1</index>
          <logical-channel-assignment>
            <assignment>
              <config>
                <optical-channel>{{SCH_NAME}}</optical-channel>
                <assignment-type>OPTICAL_CHANNEL</assignment-type>
                <allocation>{{ALLOCATION_IN_GBPS}}</allocation>
              </config>
            </assignment>
          </logical-channel-assignment>
          <index>2</index>
          <logical-channel-assignment>
            <assignment>
```



```

        <config>
        <optical-channel>{{SCH_NAME}}</optical-channel>
        <assignment-type>OPTICAL_CHANNEL</assignment-type>
        <allocation>{{ALLOCATION_IN_GBPS}}</allocation>
        <config>
        </assignment>
        <logical-channel-assignment>
        </channel>
        </logical-channels>
    </terminal-device>
</config>
</edit-config>
```

**Note:** Once the superchannel is configured. The group id of the optical channels assigned to the super channel must be the same.

## 5.4 Service deletion operations

This chapter describe the operations requires when a service is deleted. The operations are the following:

- [Delete-1] Disable the client port
- [Delete-2] Disable the line port
- [Delete-3] Disable logical-channel and delete assignment (only required if device is not preset)

### 5.4.1 [Delete-1] Disable the client port

This operation consists on the configuration needed to disable the client port.

#### RPC call example

```
<edit-config>
  <target>
    <{{target}}/>
  </target>
  <config>
    <components xmlns="http://openconfig.net/yang/platform">
      <component>
        <name>{{client-port-name}}</name>
        <port>
          <optical-port>
            <config>
              <admin-state>DISABLED</admin-state>
            </config>
          </optical-port>
        </port>
      </component>
    </components>
  </config>
</edit-config>
```

Result: The optical port will change to disable

**Important note:** This action is the only one required to disable the client and all the possible subcomponents of the port (for example the transceiver within the port). In case of the transceivers belonging to the port, they must change the admin state in coordination with the port and report the same status than the line port.

In case optical port container is not available, the admin-status of the interface module will be in charge of this action

/components/component/port/oc-line-com:optical-port/oc-line-com:state				
Attribute	Allowed Values/Format	Mod	Sup	Notes
Admin-state	["ENABLED", "DISABLED", "MAINT"]	RW	M	Provided by server or user

5.4.2 [Delete-2] Disable line port

This operation consists in the configuration needed to disable the line-port.

**In case optical port container is not available, the admin-status of the interface module will be in charge of this action**

RPC call

```
<edit-config>
  <target>
    <{{target}}/>
  </target>
  <config>
    <components xmlns="http://openconfig.net/yang/platform">
      <component>
        <name>{{line-port-name}}</name>
        <port>
          <optical-port>
            <config>
              <admin-state>DISABLED</admin-state>
            </config>
          </optical-port>
        </port>
      </component>
    </components>
  </config>
</edit-config>
```

/components/component/port/oc-line-com:optical-port/oc-line-com:state				
Attribute	Allowed Values/Format	Mod	Sup	Notes
Admin-state	["ENABLED", "DISABLED", "MAINT"]	RW	M	• Provided by <i>server</i> or <i>user</i>

**Important note:** This action is the only one required to disable the line port and all the possible subcomponents of the port (for example the transceiver within the port). In case of the transceivers subcomponent of the port, it must change the admin state in coordination with the port and the report the same status than the line port

5.4.3 [Delete-3] Disable logical channel and change assignment

This operation consists in the configuration needed to disable the logical channel

This operation consists in the configuration needed to disable the logical channel as

RPC call example



```
<edit-config>
  <target>
    <{{target}}/>
  </target>
  <config>
    <terminal-device xmlns="http://openconfig.net/yang/terminal-device">
      <logical-channels>
        <channel>
          <index>1</index>
          <config>
            <admin-state>DISABLED</admin-state>
          </config>
        </channel>
      </logical-channels>
    </terminal-device>
  </config>
</edit-config>
```

5.5 Notification

This chapter describes the operations required to create different types of subscription with Netconf or gRPC/gNMI from the Netconf collector or a gNMI client.

Netconf notification behavior must be previously preconfigured from the device. Event streams are predefined on the managed device using the openconfig-telemetry model, where the user can predefine the paths and the type of notification (on change, the sample interval, encoding)

5.5.1 [NETCONF notification-1] Subscribe for all path with Netconf

This operation subscribes the client for all the notifications available within the server. Notifications include events, alarms and telemetry.

**Note:** Netconf notification behavior must be previously preconfigured from the device. Event streams are predefined on the managed device using the openconfig-telemetry model, where the user can predefine the paths and the type of notification (on change, the sample interval, encoding). For more detail see Annex C

RPC call for subscribe all notifications

```
<create-subscription xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0"/>
```

RPC response

```
<ok>
```

Example of update

```
<notification xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
  <eventTime>2019-01-14T14:53:36Z</eventTime>
  <update>
    <optical-amplifier xmlns="http://openconfig.net/yang/optical-amplifier">
      <amplifiers>
        <amplifier>
          <name>AMPLIFIER-3-2-EDFA</name>
          <state>
            <output-power-c-band>
              <instant>-28.12</instant>
            </output-power-c-band>
          </state>
        </amplifier>
      </amplifiers>
    </optical-amplifier>
  </update>
</notification>
```

If the client performs the subscription for all the paths, the server will send not only the standards notifications, but also the proprietary.

**Example of proprietary delete notification (proprietary)**

```
<notification xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
  <eventTime>2019-01-14T15:08:01.373365Z</eventTime>
  <delete>
    <system xmlns="http://openconfig.net/yang/system">
      <alarms>
        <alarm>
          <id>2354511</id>
          <config/>
          <state>
            <id>2354511</id>
            <resource>TRANSCEIVER-6-4-2</resource>
            <text>Underlying resource unavailable - optical</text>
            <time-created>1547478459000000000</time-created>
            <severity xmlns:oc-alarm-types="http://openconfig.net/yang/alarms/types">oc-alarm-types:MAJOR</severity>
            <type-id>Uru0</type-id>
          </state>
        </alarm>
      </alarms>
    </system>
  </delete>
</notification>
```

### 5.5.2 [NETCONF notification-2] Create subscription to specific path

This operation subscribes the client for a specific path within the server.

**Note:** Netconf notification behavior must be previously preconfigured from the device. Event streams are predefined on the managed device using the openconfig-telemetry model, where the user can predefine the paths and the type of notification (on change, the sample interval, encoding)

#### RPC call

```
<rpc message-id="xx" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<create-subscription xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
  <filter type="subtree">
    <{{filter}}>
  </filter>
</create-subscription>
</rpc>
```

#### RPC call example to subscription of the q-value of a channel

```
<rpc message-id="xx" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<create-subscription xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
  <filter type="subtree">
    <terminal-device
xmlns="http://openconfig.net/yang/terminal-device">
      <logical-channels>
        <channel>
          <index>"Index_of_channel_assigned_to_och_"</index>
          <otn>
            <state>
              <q-value>
                <instant/>
              </q-value>
            </state>
          </otn>
        </channel>
      </logical-channels>
    </terminal-device>
  </filter>
</create-subscription>
</rpc>
```

#### RPC response

```
<ok>
```

A notification may contain one or more leaves of the model tree. The modified leaves and if necessary, list index leaves are reported in the notification.

Example of update



```
<notification xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
  <eventTime>2019-01-14T14:53:36Z</eventTime>
  <update>
    <optical-amplifier xmlns="http://openconfig.net/yang/optical-amplifier">
      <amplifiers>
        <amplifier>
          <name>AMPLIFIER-3-2-EDFA</name>
          <state>
            <output-power-c-band>
              <instant>-28.12</instant>
            </output-power-c-band>
          </state>
        </amplifier>
      </amplifiers>
    </optical-amplifier>
  </update>
</notification>
```

### 5.5.3 [gRPC-1] Subscribe for all paths with gNMI/gRPC

These operations defined the fields needed to create a subscription for all data models.

The gNMI specification includes three different modes: on change, sample and target-defined. All three methods can be used for different proposes, while we recommend the target-defined method as the preferred one, as allows the server to select the best mode (on change vs. sample) for each leaf, depending on the information that it represents.

#### RPC call subscription to all predefined leafs

```
Subscription {
  path: {
    elem: "*"
  }
  Mode: TARGET-DEFINED
  mode: STREAM
  encoding: JSON_IETF
}
```

#### gNMI\_cli client

```
./gNMI_cli -address {{server_ip_address:port}} -insecure -with_user_pass -dt
proto -proto "subscribe: { prefix: { } , subscription: {path: { elem: { name:
\"*\" } } } }"
```

#### Update of leaf

```
update: <
timestamp: 1554115588143693000
update: <
path: <
>
val: <
json_ietf_val:
"{\n\"openconfig-platform:components\":
{\n\"component\":
[\n
{\n \"name\": \"CARD-22-8\",
\n \"openconfig-platform-linecard:linecard\":
{\n \"state\":
{\n \"power-admin-state\": \"POWER_DISABLED\"\n}
\n}
\n}
\n}
\n}
\n}"
```

#### Delete of leaf

```
update: <
timestamp: 1547482621931322000
delete: <
elem: <
name: "system"
>
elem: <
name: "alarms"
```

```
>
elem: <
name: "alarm"
key: <
key: "id"
value: "2357082"
>
>
elem: <
name: "state"
>
elem: <
name: "id"
>
>
delete: <
elem: <
name: "system"
>
elem: <
name: "alarms"
.... truncated ...
```

5.5.4 [gRPC-2] Create subscription for specific path with gNMI

Streaming telemetry is a method of network monitoring that allows you to continuously stream data from network devices with efficient, incremental updates. Telemetry is sent based on subscriptions.

Streaming telemetry notifications are generated according to the type of subscription and at the frequency requested by the client.

The following modes MUST be supported

- ONCE — returns one-off data
- STREAM — a long-lived subscription that streams data according to triggers specified within the subscription, for example, a trigger could be when the state of an object changes, or based on a sampling interval. It includes:
  - ON\_CHANGE. Only if data model change
  - SAMPLE. Repeated
  - TARGET-DEFINED

Wildcard must also be supported to be able filter the subscription and use advanced rules to create subscriptions

For more information see <https://github.com/openconfig/reference/blob/master/rpc/gNMI/gNMI-specification.md#3515-creating-subscriptions>

Example of subscription to components/component/name=PORT-1

```
Subscription {
path: {
  elem: "components"
  elem: "component"
  key: <
                                key: "name"
                                value: "PORT-1"
  }
}
```

```
mode: ONCE
encoding: JSON_IETF
```

Example with gNMI\_cli query

```
gNMI_cli -address {{server_ip_address:port}} -query
"/components/component[name=PORT-1]" -timeout 10s -insecure -with_user_pass -
display_type proto
```

5.5.5 [gRPC-3] Create subscription of a specific path with a defined SAMPLE (sample\_interval)

Example of subscription to pre-fec-ber of channel of index 1001 and sample interval 5 seconds

```
./gNMI_cli -address 10.95.86.86:8980 -insecure -with_user_pass -dt proto -
proto "subscribe: { prefix: { } , subscription: { sample_interval:
5000000000, heartbeat_interval: 0, suppress_redundant: false, path: { elem:
{ name: \"terminal-device\" } elem: { name: \"logical-channel\" } elem: {
name: \"channel\" key { key: \"index\" value: \"1001\" }} elem: { name:
\"otn\" } elem: { name: \"state\" } elem: { name: \"pre-fec-ber\" } elem:
{ name: \"instant\" } } } }
```

All the possible combinations can be found in:

<https://github.com/openconfig/reference/blob/master/rpc/gNMI/>

Path conventions permit the use of wildcards like /\*/ or /\*.../ more information in:

<https://github.com/openconfig/reference/blob/master/rpc/gNMI/gNMI-path-conventions.md>

5.5.6 [gRPC-4] Discovery of capabilities and encoding supported

This operation is used to discovery the models supported and the encoding

```
gNMI_cli -insecure -capabilities -with_user_pass -a
{{server_ip_address:port}}
```

Example of reply

```
name: "urn:ietf:params:xml:ns:yang:iana-if-type"
organization: "IANA"
version: "2017-01-19"
>
supported_models: <
name: "urn:ietf:params:xml:ns:yang:ietf-inet-types"
organization: "IETF NETMOD (NETCONF Data Modeling Language) Working Group"
version: "2013-07-15"
>
```

```
supported_models: <
  name: "urn:ietf:params:xml:ns:yang:ietf-interfaces"
  organization: "IETF NETMOD (NETCONF Data Modeling Language) Working Group"
  version: "2014-05-08"
>
supported_encodings: JSON
supported_encodings: PROTO
supported_encodings: JSON_IETF
gNMI_version: "0.7.0"
```

5.6 PM operations

This chapter describe the operations required for getting the most important performance monitoring information needed for the operations.

This information can be reported by a “get” operation or using a subscription. In case of notification due to a subscription, behavior must be previously preconfigured from the device. Event streams are predefined on the managed device using the openconfig-telemetry model, where the user can predefine the paths (components and counters to be monitored) and the type of notification (on change, the sample interval, encoding) used during the operations.

The following table shows the mandatory paths required for obtaining the PM counters.

5.6.1 Client ports performance monitoring

5.6.1.1.1 Ethernet Client port PM

Attribute	Path	Mod	Sup	Notes
CRC Tx/Rx Errored Packets	Ethernet Rx CRC Errored Packets: /terminal-device/logical-channels/channel[index=i]/ethernet/state/in-crc-errors  Ethernet Tx CRC Errored Packets: /terminal-device/logical-channels/channel[index=i]/ethernet/state/out-crc-errors	RO	M	Number of packets received/transmitted that contained an FCS error and were between 64 bytes and up to the configured maximum frame size.  Rx CRC Errored Packets are not discarded, they are counted and pass through the system transparently.
Ethernet Tx/Rx Block Error Count	Ethernet Rx Block Error Count: /terminal-device/logical-channels/channel[index=i]/ethernet/state/in-block-errors  Ethernet Tx Block Error	RO	M	The number of errored blocks that have been received/ transmitted on a given port.



	Count: /terminal-device/logical-channels/channel[index=i]/ethernet/state/out-block-errors			
Ethernet Tx/Rx PCS Lane BIP Error Count	Ethernet Rx PCS Lane BIP Error Count: /terminal-device/logical-channels/channel[index=i]/ethernet/state/in-pcs-bip-errors  Ethernet Tx PCS Lane BIP Error Count: /terminal-device/logical-channels/channel[index=i]/ethernet/state/out-pcs-bip-errors	RO	M	The measure of the sum of bit interleaved parity errors over all lanes received/transmitted at the given port.
Ethernet PCS Errored Seconds (ES)	/terminal-device/logical-channels/channel[index=i]/ethernet/state/in-pcs-errored-seconds	RO	M	The number of Rx errored seconds at the PCS layer. An errored second is a second in which there was a traffic-impacting defect experienced at the PCS layer.
Ethernet PCS Severely Errored Seconds (SES)	/terminal-device/logical-channels/channel[index=i]/ethernet/state/in-pcs-severely-errored-seconds	RO	M	The number of Rx severely errored seconds at the PCS layer. A severely errored second is a second in which more than the configured number of errors have occurred on the PCS layer.
Ethernet PCS Unavailable Seconds (UAS)	/terminal-device/logical-channels/channel[index=i]/ethernet/state/in-pcs-unavailable-seconds	RO	M	The number of Rx unavailable seconds that have occurred on the MAC layer. When any of the PCS defects (LOBL, HIBER, LOAM, LF, RF) are detected for 10 seconds or more, these seconds are counted as PCS-UAS.
Traffic counters packets (in/out)	/interfaces/interface/state/counters/out-packets  /interfaces/interface/state/counters/in-packets	RO	M	The total number of packets in/out the interface
Traffic counters octets (in/out)	/interfaces/interface/state/counters/out-octets  /interfaces/interface/state/counters/in-octets	RO	M	The total number of octets in/out the interface

5.6.1.1.2 OTN client ports.  
Note: Counters must be supported at near and far OTU/ODU

Attribute	path	Mod	Sup	Notes
OTN Errored Seconds (ES)*	/terminal-device/logical-channel/channel[index=i]/otn/state/errored-seconds	RO	M	The number of Rx errored seconds at OTN layer
OTN Severely Errored Seconds (SES)*	/terminal-device/logical-channel/channel[index=i]/otn/state/severely-errored-seconds	RO	M	The number of Rx severely errored seconds at the OTN layer
OTN Unavailable Seconds (UAS)*	/terminal-device/logical-channel/channel[index=i]/otn/state/unavailable-seconds	RO	M	Unavailable seconds at OTN layer
OTN Background Block Errors (BBE)*	/terminal-device/logical-channel/channel[index=i]/otn/state/background-block-errors	RO	M	The number of background block errors at OTN layer
FEC Uncorrected blocks	/terminal-device/logical-channels/channel/otn/state/fec-uncorrectable-blocks	RO	M	The number of blocks that were uncorrectable by the FEC
FEC Uncorrected Words	/terminal-device/logical-channels/channel/otn/state/fec-uncorrectable-words	RO	M	The number of words that were uncorrectable by the FEC
FEC-corrected-bits	/terminal-device/logical-channels/channel/otn/state/fec-corrected-bits	RO	M	The number of bits that were corrected by the FEC

5.6.1.1.3 PTP (Physical termination point) Client

Attribute	path	Mod	Sup	Notes
Rx Optical Power Avg	/components/component[name=n]/transceiver/physical-channels/channel[index=y]/state/inputpower/avg	RO	M	The average Rx optical power for the client and line PTP.
Rx Optical Power instant	/components/component[name=n]/transceiver/physical-channels/channel[index=y]/state/inputpower/instant	RO	M	The instant Rx optical power value
Tx Optical Power avg	/components/component[name=n]/transceiver/physical-channels/channel[index=y]/state/outputpower/avg	RO	M	The average Tx optical power for the client and line PTP.
Tx Optical Power instant	/components/component[name=n]/transceiver/physical-channels/channel[index=y]/state/outputpower/instant	RO	M	The instant Tx optical power for the client and line PTP.

	state/outputpower/instant			
--	---------------------------	--	--	--

5.6.2 Line ports performance monitoring

Attribute	path	Mod	Sup	Notes
OTN pre-FEC BER avg	//terminal-device/logical-channel/channel[index=i]/otn/state/pre-fec-ber/avg	RO	M	Bit Error Rate before Forward Error Correction
OTN pre-FEC BER instant	/terminal-device/logical-channel/channel[index=i]/otn/state/pre-fec-ber/instant	RO	M	Bit Error Rate before Forward Error Correction
OTN pre-FEC BER interval	/terminal-device/logical-channel/channel[index=i]/otn/state/pre-fec-ber/interval	RO	M	If supported by the system, this reports the time interval over which the min/max/average statistics are computed by the system.
Q-factor (avg)	/terminal-device/logical-channels/channel[index=i]/otn/state/q-value/avg	RO	M	Q-PM representation of FEC Error Count
Q-factor (instant)	/terminal-device/logical-channels/channel[index=i]/otn/state/q-value/instant	RO	M	Q-PM representation of FEC Error Count.
Q-factor (interval)	/terminal-device/logical-channels/channel[index=i]/otn/state/q-value/interval	RO	M	If supported by the system, this reports the time interval over which the min/max/average statistics are computed by the system.
FEC Uncorrected Words	/terminal-device/logical-channel/channel[index=i]/otn/state/fec-uncorrected-words	RO	M	
Chromatic Dispersion (instant)	//components/component[name=n]/optical-channel/state/chromatic-dispersion/instant	RO	M	Chromatic dispersion
Chromatic Dispersion (avg)	//components/component[name=n]/optical-channel/state/chromatic-dispersion/avg	RO	M	Chromatic dispersion
eSNR instant	/terminal-device/logical-channels/channel/otn/state/esnr/instant	RO	M	Electrical signal to noise ratio. Baud rate normalized signal to noise ratio based on error vector magnitude in dB with two decimals precision. Values include the instantaneous, average,

				minimum, and maximum statistics. If avg/min/max statistics are not supported, the target is expected to just supply the instant value
eSNR avg	/terminal-device/logical-channels/channel/otn/state/esnr/avg	RO	M	Electrical signal to noise ratio. Baud rate normalized signal to noise ratio based on error vector magnitude in dB with two decimals precision. Values include the instantaneous, average, minimum, and maximum statistics. If avg/min/max statistics are not supported, the target is expected to just supply the instant value

5.6.2.1.1 OTN line ports.

Note: Counters must be allowed at OTU and ODU layer.

Attribute	path	Mod	Sup	Notes
OTN Errored Seconds (ES)*	/terminal-device/logical-channel/channel[index=i]/otn/state/errored-seconds	RO	M	The number of Rx errored seconds at OTN layer
OTN Severely Errored Seconds (SES)*	/terminal-device/logical-channel/channel[index=i]/otn/state/severely-errored-seconds	RO	M	The number of Rx severely errored seconds at the OTN layer
OTN Unavailable Seconds (UAS)*	/terminal-device/logical-channel/channel[index=i]/otn/state/unavailable-seconds	RO	M	Unavailable seconds at OTN layer
OTN Background Block Errors (BBE)*	/terminal-device/logical-channel/channel[index=i]/otn/state/background-block-errors	RO	M	The number of background block errors at OTN layer
FEC Uncorrected blocks	/terminal-device/logical-channels/channel/otn/state/fec-uncorrectable-blocks	RO	M	The number of blocks that were uncorrectable by the FEC

FEC Uncorrected Words	/terminal-device/logical-channels/channel/otn/state/fec-uncorrectable-words	RO	M	The number of words that were uncorrectable by the FEC
FEC-corrected-bits	/terminal-device/logical-channels/channel/otn/state/fec-corrected-bits	RO	M	The number of bits that were corrected by the FEC

5.6.2.1.2 PTP line ports.

Attribute	path	Mod	Sup	Notes
Rx Optical Power Avg	/components/component[name=n]/transceiver/physical-channels/channel[index=y]/state/inputpower/avg	RO	M	The average Rx optical power for the client and line PTP.
Rx Optical Power instant	/components/component[name=n]/transceiver/physical-channels/channel[index=y]/state/inputpower/instant	RO	M	The instant Rx optical power
Tx Optical Power Avg	/components/component[name=n]/transceiver/physical-channels/channel[index=y]/state/outputpower/avg	RO	M	The average Tx optical power for the client and line PTP.
Tx Optical Power instant	/components/component[name=n]/transceiver/physical-channels/channel[index=y]/state/outputpower/instant	RO	M	The instant Tx optical power for the client and line PTP.
Rx Channel Power Avg	/components/component[name=n]/optical-channel/state/input-power/avg	RO	M	The input optical power of a physical channel in units of 0.01dBm, which may be associated with individual physical channels, or an aggregate of multiple physical channels (i.e., for the overall transceiver). For an aggregate, this may be a measurement from a photodetector or a calculation performed on the device by summing up all of the related individual physical channels.
Rx Channel Power instant	/components/component[name=n]/optical-channel/state/input-power/instant	RO	M	The input optical power of a physical channel in units of 0.01dBm, which may be associated with individual

				physical channels, or an aggregate of multiple physical channels (i.e., for the overall transceiver). For an aggregate, this may be a measurement from a photodetector or a calculation performed on the device by summing up all of the related individual physical channels.
Tx Channel Power Avg	/components/component[name=n]/optical-channel/state/output-power/avg	RO	M	The output optical power of a physical channel in units of 0.01dBm, which may be associated with individual physical channels, or an aggregate of multiple physical channels (i.e., for the overall transceiver). For an aggregate, this may be a measurement from a photodetector or a a calculation performed on the device by summing up all of the related individual physical channels.
Tx Channel Power instant	/components/component[name=n]/optical-channel/state/output-power/instant	RO	M	The output optical power of a physical channel in units of 0.01dBm, which may be associated with individual physical channels, or an aggregate of multiple physical channels (i.e., for the overall transceiver). For an aggregate, this may be a measurement from a photodetector or a a calculation performed on the device by summing up all of the related individual physical channels.

5.6.2.1.3 Counters not included in OC model

To add attributes not included in the openconfig model, we required to use the properties field. As part of this specification, is mandatory to include the unidirectional latency inside the properties to container. Similarly, suppliers must use this container to include any relevant attribute not include in the standard model

Attribute	path	Mod	Su	Notes
-----------	------	-----	----	-------

			p	
latency	/components/component[name=n]/properties/property[name=latency	RO	M	E2E unidirectional laten

5.6.3 Example of PM RPC call

The following call shows an example of RPC used for retrieving the pre-fec-ber.

```
<get>
  <filter>
    <terminal-device xmlns="http://openconfig.net/yang/terminal-device">
      <logical-channels>
        <channel>
          <index>{{INDEX_OF_LOGICAL_CHANNEL_TO_BE_MONITORED}}</index>
          <otn>
            <state>
              <pre-fec-ber>
                <instant/>
              </pre-fec-ber>
            </state>
          </otn>
        </channel>
      </logical-channels>
    </terminal-device>
  </filter>
</get>
```

5.6.4 Example of creation a PM notification

To create a netconf notification we can use the same structure as in operation [NETCONF notification-1]

```
<rpc message-id="xx" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <create-subscription xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
    <filter type="subtree">
      <terminal-device xmlns="http://openconfig.net/yang/terminal-device">
        <logical-channels>
          <channel>
            <index>{{INDEX_OF_LOGICAL_CHANNEL_TO_BE_MONITORED}}</index>
            <otn>
              <state>
                <pre-fec-ber>
                  <instant/>
                </pre-fec-ber>
              </state>
            </otn>
          </channel>
        </logical-channels>
      </terminal-device>
    </filter>
```

```
</create-subscription>
</rpc>
```

5.6.5 Example of PM subscription with gnmi

Example of subscription to pre-fec-ber of channel of index 1001 and sample interval 5 seconds using gnmi client

```
./gNMI_cli -address 10.95.86.86:8980 -insecure -with_user_pass -dt proto -
proto "subscribe: { prefix: { } , subscription: { sample_interval:
5000000000, heartbeat_interval: 0, suppress_redundant: false, path: { elem:
{ name: \"terminal-device\" } elem: { name: \"logical-channel\" } elem: {
name: \"channel\" key { key: \"index\" value: \"1001\" }} elem: { name:
\"otn\" } elem: { name: \"state\" } elem: { name: \"pre-fec-ber\" } elem:
{ name: \"instant\" } } } }
```

5.7 Alarms

The openconfig system model uses the following structure for the fields required to retrieve the information about an alarm.

/system/alarms/ alarm				
Attribute	path	Mod	Sup	Notes
+--ro state		RO	M	
	/system/alarms/alarm/state/id	RO	M	Unique ID for the alarm -- this will not be a configurable parameter on many implementations
resource	/system/alarms/alarm/state/resource	RO	M	The item that is under alarm within the device. The resource may be a reference to an item which is defined elsewhere in the model. For example, it may be a platform/component, interfaces/interface, terminal-device/logical-channels/channel, etc. In this case the system should match the



				name of the referenced item exactly. The referenced item could alternatively be the path of the item within the model.
text	/system/alarms/alarm/state/text	RO	M	The string used to inform operators about the alarm. This MUST contain enough information for an operator to be able to understand the problem. If this string contains structure, this format should be clearly documented for programs to be able to parse that information
time-created	/system/alarms/alarm/state/time-created	RO	M	The time at which the alarm was raised by the system. This value is expressed relative to the Unix Epoch.
severity	/system/alarms/alarm/state/severity	RO	M	The severity level indicating the criticality and impact of the alarm
type-id	/system/alarms/alarm/state/type-id	RO	M	The abbreviated name of the alarm, for example LOS, EQPT, or OTS. Also referred to in different systems as condition type, alarm identifier, or alarm mnemonic. It is recommended to use the OPENCONFIG_ALARM_TYPE_ID identities where possible and only use the string type when the desired identityref is not yet defined

The SNMP entity is a field used to do the mapping of the alarm with the SNMP Entity MIB identifier.

Attribute	path	Mod	Sup	Notes
MIB-ext-entity	/components/component/state/oc-platform-ext:entity-id	RO	M	A unique numeric identifier assigned by the system to the component. This identifier may be used to represent the corresponding SNMP Entity MIB identifier.

This use case include a target "standardized Alarm List" which includes the most commonly found alarms processed in Fault Management systems nowadays. The complete list of alarms to be supported by the current proposed model will be covered by the next deliverable.

We've identified a gap between TAPI and OC, that it will be discussed in the NBi work to reduce the gap between both models.





## 6 [Annex A] - Operational Mode augment

### 6.1 Yang model. Tree output

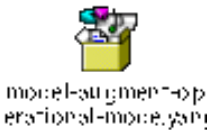
```
module: openconfig-operational-mode-draft-tipmust

augment /oc-opt-term:terminal-device/oc-opt-term:operational-modes/oc-opt-term:mode/oc-opt-term:state:
  +--ro (mode)?
    +--:(G.698.2)
      | +--ro standard-mode?                standard-mode
    +--:(explicit-mode)
      +--ro operational-mode-capabilities
        +--ro description?                  string
        +--ro modulation-format?
modulation-format
      +--ro modulation-format-proprietary
      | +--ro description?    string
      +--ro bit-rate?        bit-rate
      +--ro baud-rate?       decimal64
      +--ro min-osnr?        decimal64
      +--ro grid-type?       grid-type
      +--ro adjustment-granularity?
adjustment-granularity
      +--ro otsi-media-channel?    decimal64
      +--ro effective-media-channel? decimal64
      +--ro central-frequency-min? uint64
      +--ro central-frequency-max? uint64
      +--ro fec
      | +--ro fec-coding?          fec-coding
      | +--ro fec-coding-proprietary
      | | +--ro description?    string
      | +--ro gain?            decimal64
      +--ro min-output-power?    decimal64
      +--ro max-output-power?    decimal64
      +--ro input-power-sensitivity? decimal64
      +--ro min-q-value?         decimal64
      +--ro chromatic-dispersion-tolerance? decimal64
      +--ro differential-group-delay-tolerance? decimal64
      +--ro chromatic-and-polarization-dispersion-penalty* []
      | +--ro chromatic-dispersion?    decimal64
      | +--ro polarization-mode-dispersion? decimal64
      | +--ro penalty?                decimal64
      +--ro max-polarization-dependent-loss? decimal64
      +--ro filter
      | +--ro shape?    string
      | +--ro order?    uint32
      | +--ro roll-off? decimal64
      +--ro sop?        string

  augment /oc-opt-term:terminal-device/oc-opt-term:operational-modes/oc-opt-term:mode:
    +--ro properties
      +--ro property* [name]
      +--ro name      -> ../config/name
      +--ro config
```

```
|  +--ro name?      string
|  +--ro value?     union
+--ro state
  +--ro name?       string
  +--ro value?      union
  +--ro configurable? Boolean
```

The following attached file shows the full Yang model



6.2 Yang model example

```
<?xml version="1.0"?>
<config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <terminal-device xmlns="http://openconfig.net/yang/terminal-device">
    <operational-modes>
      <mode>
        <mode-id>10</mode-id>
        <config/>
        <state>
          <mode-id>10</mode-id>
          <description>MODE10</description>
          <vendor-id>TIPMUST</vendor-id>
          <oc-oper-mode:standard-mode
mode="http://example.net/yang/operational-mode-draft-tipmust">B-DScW-ytz (v)
          </oc-oper-mode:standard-mode>
        </state>
      </mode>
      <mode>
        <mode-id>20</mode-id>
        <config/>
        <state>
          <mode-id>20</mode-id>
          <description>MODE20</description>
          <vendor-id>TIPMUST</vendor-id>
          <oc-oper-mode:standard-mode
mode="http://example.net/yang/operational-mode-draft-tipmust">B-DScW-ytz (v)
          </oc-oper-mode:standard-mode>
        </state>
      </mode>
      <mode>
        <mode-id>100</mode-id>
        <oc-oper-mode:supported-modes xmlns:oc-oper-
mode="http://example.net/yang/operational-mode-draft-tipmust">
          <oc-oper-mode:supported-application-codes>10</oc-oper-
mode:supported-application-codes>
```



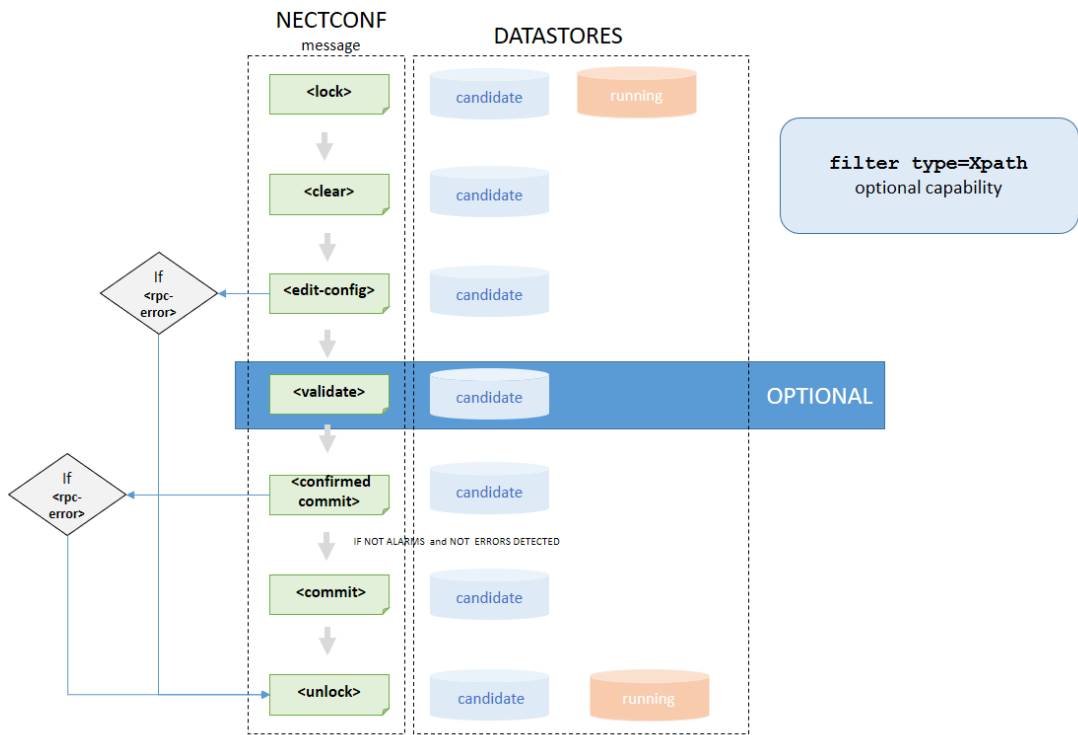
```

        <oc-oper-mode:supported-application-codes>20</oc-oper-
mode:supported-application-codes>
    </oc-oper-mode:supported-modes>
    <config/>
    <state>
        <mode-id>100</mode-id>
        <description>MODE100</description>
        <vendor-id>TIPMUST</vendor-id>
        <oc-oper-mode:operational-mode-capabilities xmlns:oc-
oper-mode="http://example.net/yang/operational-mode-draft-tipmust">
            <oc-oper-mode:description>Additional details</oc-oper-
mode:description>
            <oc-oper-mode:modulation-format>oc-oper-mode:MODULATION-
FORMAT-8QAM</oc-oper-mode:modulation-format>
            <oc-oper-mode:baud-rate>37.5</oc-oper-mode:baud-rate>
            <oc-oper-mode:bit-rate>oc-opt-types:TRIB_RATE_10G</oc-
oper-mode:bit-rate>
            <oc-oper-mode:central-frequency-min>191000000</oc-oper-
mode:central-frequency-min>
            <oc-oper-mode:central-frequency-max>196000000</oc-oper-
mode:central-frequency-max>
            <oc-oper-mode:grid-type>FLEX</oc-oper-mode:grid-type>
            <oc-oper-mode:fec>
                <oc-oper-mode:fec-coding>oc-oper-mode:FEC-G</oc-oper-
mode:fec-coding>
            </oc-oper-mode:fec>
            <oc-oper-mode:min-output-power>0.00</oc-oper-mode:min-
output-power>
            <oc-oper-mode:max-output-power>2.00</oc-oper-mode:max-
output-power>
            <oc-oper-mode:input-power-sensitivity>-10.00</oc-oper-
mode:input-power-sensitivity>
            <oc-oper-mode:min-q-value>0.00</oc-oper-mode:min-q-value>
            <oc-oper-mode:chromatic-dispersion-tolerance>0.00</oc-
oper-mode:chromatic-dispersion-tolerance>
            <oc-oper-mode:differential-group-delay-
tolerance>0.00</oc-oper-mode:differential-group-delay-tolerance>
            <oc-oper-mode:filter>
                <oc-oper-mode:shape>ROLL_Shape</oc-oper-mode:shape>
            </oc-oper-mode:filter>
            <oc-oper-mode:sop>Some SOP</oc-oper-mode:sop>
        </oc-oper-mode:operational-mode-capabilities>
    </state>
</mode>
</operational-modes>
```



## 7 [Annex B] - Operations for netconf validation

We request a minimum set of capabilities and operations needed for a correct implementation of the configuration. For the edition of the configuration, a candidate and running datastore is required. The following picture describe the correct workflow required for editing the configuration of a device.



The following table define the operations requires for validation the capabilities needed for the normal operation over the datastores

Test	Objecti ve	Procedure	Comments	RPC call and reply
1	Check if the authenti cation with usernam e and passwor d is enabled.	Creating a SSH connection over port 22.	Is desirable to have SSH activated in case the device has to be accessed due to NETCONF issues.	
2	Check if the port 830 is enabled to exchang e NETC ONF message s.	Creating a SSH connection over port 830.		

3	Check if the capabilities exchange is supported	Sending a <hello> rpc to receive a message with the capabilities.	This is useful to know which YANG models are supported by the NETCONF agent.	<pre>CALL &lt;?xml version="1.0" encoding="UTF-8"?&gt; &lt;hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"&gt;   &lt;capabilities&gt;     &lt;capability&gt;urn:ietf:params:netconf:base:1.0&lt;/capability&gt;   &lt;/capabilities&gt; &lt;/hello&gt;  REPLY EXAMPLE &lt;?xml version="1.0" encoding="UTF-8"?&gt; &lt;hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"&gt; &lt;capabilities&gt; &lt;capability&gt;urn:ietf:params:netconf:base:1.0&lt;/capability&gt; &lt;capability&gt;urn:ietf:params:netconf:capability:writable-running:1.0&lt;/capability&gt; &lt;capability&gt;urn:ietf:params:netconf:capability:candidate:1.0&lt;/capability&gt; ... &lt;/capabilities&gt; &lt;session-id&gt;5&lt;/session-id&gt; &lt;/hello&gt;</pre>
4	Check if the supported NETCONF version is 1.0	When receiving the capabilities, we have to check that the NETCONF agent support v1.0	Version 1.1 can be supported too, but it won't be used.	<pre>REPLY EXAMPLE &lt;?xml version="1.0" encoding="UTF-8"?&gt; &lt;hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"&gt; &lt;capabilities&gt; &lt;capability&gt;urn:ietf:params:netconf:base:1.0&lt;/capability&gt; &lt;capability&gt;urn:ietf:params:netconf:capability:writable-running:1.0&lt;/capability&gt; &lt;capability&gt;urn:ietf:params:netconf:capability:candidate:1.0&lt;/capability&gt; ... &lt;/capabilities&gt; &lt;session-id&gt;5&lt;/session-id&gt; &lt;/hello&gt;</pre>
5	Check if the <get-config> rpc is correctly implemented.	Sending a <get-config> rpc request and receive a <rpc-reply> including the configuration parameters of the device.	Only configuration parameters must be retrieved. It will be tested over both configurations, candidate and running.	<pre>&lt;rpc message-id="1" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"&gt; &lt;get-config/&gt; &lt;/rpc&gt;</pre> <p>Note: must return only config</p>
6	Check if the <get> rpc is correctly implemented.	Sending a <get> rpc request and receive a <rpc-reply> including the configuration and running state parameters of the device.	Both type of parameters, configuration and running state, must be retrieved.	<pre>&lt;rpc message-id="1" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"&gt; &lt;get/&gt; &lt;/rpc&gt;</pre> <p>Note: must return full configuration (config+state)</p>
7	Check if the <edit-config> rpc is correctly implemented.	Sending a <edit-config> rpc request including the parameters to change to receive a <rpc-reply> including a OK message.	Only parameters which are allowed to be modified can be modified by using this rpc. It will be tested over both configurations, candidate and running.	<p>Example of configuration of output power. Test over candidate and running</p> <pre>CALL &lt;edit-config&gt;   &lt;target&gt;     &lt;{{target}}/&gt;   &lt;/target&gt;   &lt;config&gt;     &lt;components xmlns="http://openconfig.net/yang/platform"&gt;       &lt;component&gt;         &lt;name&gt;{{och_component_name}}&lt;/name&gt;         &lt;optical-channel xmlns="http://openconfig.net/yang/terminal-device"&gt;           &lt;config&gt;             &lt;target-output-power&gt;{{output_power}}&lt;/target-output-power&gt;           &lt;/config&gt;         &lt;/optical-channel&gt;       &lt;/component&gt;     &lt;/components&gt;   &lt;/config&gt; &lt;/edit-config&gt;  REPLY &lt;rpc-reply message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"&gt;   &lt;ok/&gt; &lt;/rpc-reply&gt;</pre>



8	Check if the <commit> rpc is correctly implemented.	Sending a <commit> rpc to overwrite the running config with the parameters of the candidate config.	Running config will be fully overwritten with the candidate config.	<div><pre>CALL &lt;edit-config&gt;   &lt;target&gt;     &lt;candidate/&gt;   &lt;/target&gt;   &lt;config&gt;     &lt;components xmlns="http://openconfig.net/yang/platform"&gt;       &lt;component&gt;         &lt;name&gt;{{och_component_name}}&lt;/name&gt;         &lt;optical-channel xmlns="http://openconfig.net/yang/terminal-device"&gt;           &lt;config&gt;             &lt;target-output-power&gt;{{output_power}}&lt;/target-output-power&gt;           &lt;/config&gt;         &lt;/optical-channel&gt;       &lt;/component&gt;     &lt;/components&gt;   &lt;/config&gt; &lt;/edit-config&gt;  REPLY   &lt;rpc-reply message-id="101"     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"&gt;     &lt;ok/&gt;   &lt;/rpc-reply&gt;  COMMIT  &lt;rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"&gt;   &lt;commit/&gt; &lt;/rpc&gt;  COMMIT REPLY   &lt;rpc-reply message-id="101"     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"&gt;     &lt;ok/&gt;   &lt;/rpc-reply&gt;</pre></div> <div>Note: verify that new output power is configured in the running datastore</div>
9	Check if capability “Writable-running” is correctly implemented.	Sending a <edit-config> rpc request with running datastore as target.		<div><pre>CALL &lt;?xml version="1.0" encoding="UTF-8"?&gt; &lt;hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"&gt;   &lt;capabilities&gt;     &lt;capability&gt;urn:ietf:params:netconf:base:1.0&lt;/capability&gt;   &lt;/capabilities&gt; &lt;/hello&gt;  REPLY EXAMPLE &lt;?xml version="1.0" encoding="UTF-8"?&gt; &lt;hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"&gt;   &lt;capabilities&gt;     &lt;capability&gt;urn:ietf:params:netconf:base:1.0&lt;/capability&gt;     &lt;capability&gt;urn:ietf:params:netconf:capability:writable-running:1.0&lt;/capability&gt;     &lt;capability&gt;urn:ietf:params:netconf:capability:candidate:1.0&lt;/capability&gt;     ...   &lt;/capabilities&gt;   &lt;session-id&gt;5&lt;/session-id&gt; &lt;/hello&gt;</pre></div>
10	Check if capability “Candidate Config” is correctly implemented.	Sending a <edit-config> rpc request with candidate datastore as target.	Only of candidate datastore is available	<div><pre>CALL &lt;?xml version="1.0" encoding="UTF-8"?&gt; &lt;hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"&gt;   &lt;capabilities&gt;     &lt;capability&gt;urn:ietf:params:netconf:base:1.0&lt;/capability&gt;   &lt;/capabilities&gt; &lt;/hello&gt;  REPLY EXAMPLE &lt;?xml version="1.0" encoding="UTF-8"?&gt; &lt;hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"&gt;   &lt;capabilities&gt;     &lt;capability&gt;urn:ietf:params:netconf:base:1.0&lt;/capability&gt;     &lt;capability&gt;urn:ietf:params:netconf:capability:writable-running:1.0&lt;/capability&gt;     &lt;capability&gt;urn:ietf:params:netconf:capability:candidate:1.0&lt;/capability&gt;     ...   &lt;/capabilities&gt;   &lt;session-id&gt;5&lt;/session-id&gt; &lt;/hello&gt;</pre></div>
11	Check if capability “Confirmed-commit” is correctly implemented.	Sending a <commit> rpc with the field <confirmed>.	If a commit rpc without tag <confirmed> is not sent after the first	<div><pre>&lt;capability&gt;urn:ietf:params:netconf:capability:confirmed-commit:1.1&lt;/capability&gt;</pre></div> <div><pre>CALL &lt;rpc message-id="101"   xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"&gt;</pre></div>



	” is correctly implemented.		commit, the changes will be discarded.  A <cancel-commit> rpc can be executed to discard the changes too.	<pre>&lt;commit&gt; &lt;confirmed/&gt; &lt;/commit&gt; &lt;/rpc&gt;  REPLY &lt;rpc-reply message-id="101"   xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"&gt;   &lt;ok/&gt; &lt;/rpc-reply&gt;</pre>
12	Check if capability “Rollback-on-error” is correctly implemented.	Sending two <edit-config> rpcs at the same time to the same tree of the model.	The datastore will be rolled back to previous state with no modifications.	<pre>&lt;capability&gt;urn:ietf:params:netconf:capability:rollback-on-error:1.0&lt;/capability&gt;</pre>
13	Cancel-commit	Rollback configuration to the previous state if used confirmed commit	Cancels an ongoing confirmed commit	<pre>&lt;capability&gt;urn:ietf:params:netconf:capability:validate:1.1&lt;/capability&gt;  CALL &lt;rpc message-id="102"   xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"&gt;   &lt;cancel-commit/&gt; &lt;/rpc&gt;</pre>
14	Lock-unlock	Lock datastore to avoid configurations from another clients at the same time	The candidate configuration can be locked using the <lock> operation with the <candidate> element as the <target> parameter:  Also all the datastores	<pre>LOCK  &lt;rpc message-id="101"   xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"&gt;   &lt;unlock&gt;     &lt;target&gt;       &lt;candidate/&gt;     &lt;/target&gt;   &lt;/unlock&gt; &lt;/rpc&gt;  UNLOCK  &lt;rpc message-id="101"   xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"&gt;   &lt;unlock&gt;     &lt;target&gt;       &lt;candidate/&gt;     &lt;/target&gt;   &lt;/unlock&gt; &lt;/rpc&gt;</pre>
15	Check if capability “Validate” is correctly implemented.  Note: include the test-options	Send a <validate> rpc request to see if the datastore configuration is correct.	If it is correct, a <rpc-reply> with a OK message will be received.	<pre>&lt;capability&gt;urn:ietf:params:netconf:capability:validate:1.1&lt;/capability&gt;  CALL &lt;rpc message-id="101"   xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"&gt;   &lt;validate&gt;     &lt;source&gt;       &lt;candidate/&gt;     &lt;/source&gt;   &lt;/validate&gt; &lt;/rpc&gt;  REPLY &lt;rpc-reply message-id="101"   xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"&gt;   &lt;ok/&gt; &lt;/rpc-reply&gt;</pre>
16	Check if capability “Notification”	Sending a <hello> rpc to receive a message with the capabilities.	This is useful to know which YANG models are supported by the	<pre>CALL &lt;?xml version="1.0" encoding="UTF-8"?&gt; &lt;hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"&gt;   &lt;capabilities&gt;     &lt;capability&gt;urn:ietf:params:netconf:base:1.0&lt;/capability&gt;   &lt;/capabilities&gt; &lt;/hello&gt;  REPLY EXAMPLE &lt;?xml version="1.0" encoding="UTF-8"?&gt;</pre>

			NETCONF agent.	<pre>&lt;hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"&gt; &lt;capabilities&gt; &lt;capability&gt;urn:ietf:params:netconf:base:1.0&lt;/capability&gt; &lt;capability&gt;urn:ietf:params:netconf:capability:writable-running:1.0&lt;/capability&gt; &lt;capability&gt;urn:ietf:params:netconf:capability:notification:1.0&lt;/capability&gt; ... &lt;/capabilities&gt; &lt;session-id&gt;5&lt;/session-id&gt; &lt;/hello&gt;</pre>
--	--	--	----------------	---

## 8 [Annex C] - Preconfiguration of the telemetry behaviour of the device using OpenConfig telemetry-model

Streaming telemetry is a new paradigm in monitoring the health of a network. It provides a mechanism to efficiently stream configuration and operational data of interest from NE. This streamed data is transmitted in a structured format to remote management systems for monitoring and troubleshooting purposes.

Telemetry behaviour of the device MUST be preconfigured. The OC model to configure model-driven telemetry parametrization as defined above is “openconfig-telemetry.yang”. Previously is going to be necessary to enable gRPC server on the device to accept incoming connections from the receiver. That configuration is made by means of “openconfig-system.yang” module which has configuration data for the gRPC server.

- ✓ **Destination-groups:** Contain the details about the destinations. Include the destination address (ipv4 or ipv6), port, transport, and encoding format
- ✓ **Sensor-groups:** Contain the sensor paths. Sensor path represents the path in the hierarchy of a telemetry YANG data model, specifying the subset of the data that you want to stream from the device
- ✓ **Subscriptions:** Subscription binds the destination-group with the sensor-group and sets the streaming method. Separating the sensor-paths into different subscriptions enhances the efficiency of the device to retrieve operational data at scale. The streaming method can be cadence-driven or event-driven telemetry.
  - Cadence-driven telemetry continually streams data (operational statistics and state transitions) at a configured cadence. The higher frequency of the data that is continuously streamed helps you closely identify emerging patterns in the network
  - Event-driven telemetry optimizes data that is collected at the receiver and streams data only when a state transition occurs and thus optimizes data that is collected at the receiver. For example, EDT streams data about interface state transitions, IP route updates, and so on.

The configuration for event-driven telemetry is similar to cadence-driven telemetry, with only the sample interval as the differentiator. Configuring the sample interval value to 0, zero, sets the subscription for event-driven telemetry, while configuring the interval to any non-zero value sets the subscription for cadence-driven telemetry.

The **xpath** and description of the parameters used are described below:

### Destination-groups

- 1) `/telemetry-system/destination-groups/destination-group/config/group-id` → Unique identifier for the destination group
- 2) `/telemetry-system/destination-groups/destination-group/destinations/destination/config/destination-address` → IP address of the telemetry stream destination
- 3) `/telemetry-system/destination-groups/destination-group/destinations/destination/config/destination-port` → Protocol (udp or tcp) port number for the telemetry stream destination

### Sensor-groups

- 4)

*/telemetry-system/sensor-groups/sensor-group/config/sensor-group-id* → Name or identifier for the sensor group itself. Will be referenced by other configuration specifying a sensor group
- 5)

*/telemetry-system/sensor-groups/sensor-group/sensor-paths/sensor-path/config/path* → Path to a section of operational state of interest (the sensor).

Subscriptions

A telemetry subscription consists of a set of collection destinations, stream attributes, and associated paths to state information in the model (sensor data). There are two types, persistent and dynamic mapping to “dial-out” and “dial-in” modes, respectively. Although the modes to establish a telemetry session are different, both modes use the same data model and stream the same data:

- ✓

A persistent telemetry subscription is configured locally on the device through configuration, and is persistent across device restarts or other redundancy changes.
- ✓

A dynamic subscription is typically configured through an RPC channel, and does not persist across device restarts, or if the RPC channel is reset or otherwise torn down.

Accordingly, OC split subscription in two containers, one including “read-write access” parameters to configure permanent subscription and a second for dynamic subscription, where parameters are “read-only” in that case (a system generated identifier of the telemetry subscription is used to refer to each on-demand dynamic telemetry session created).

On one hand, the parameters used for persistent subscription, read-write (config tree) are described below:

- 6)

*/telemetry-system/subscriptions/persistent-subscriptions/persistent-subscription/config/name* → User configured identifier of the telemetry subscription. This value is used primarily for subscriptions configured locally on the network element.
- 7)

*/telemetry-system/subscriptions/persistent-subscriptions/persistent-subscription/config/local-source-address* → The IP address which will be the source of packets from the device to a telemetry collector destination
- 8)

*/telemetry-system/subscriptions/persistent-subscriptions/persistent-subscription/config/originated-qos-marking* → DSCP marking of packets generated by the telemetry subsystem on the network device.
- 9)

*/telemetry-system/subscriptions/persistent-subscriptions/persistent-subscription/config/protocol* → Selection of the transport protocol for the telemetry stream. The following options are admitted, with GRPC as mandatory support for iFusion Project.

PROTOCOL	DESCRIPTION
STREAM_SSH	Telemetry stream is carried over a SSH connection
STREAM_GRPC	Telemetry stream is carried over via the gRPC framework
STREAM_JSON_RPC	Telemetry stream is carried via the JSON-RPC framework
STREAM_THRIFT_RPC	Telemetry stream is carried via the Apache Thrift framework
STREAM_WEBSOCKET_RPC	Telemetry stream is carried by the WebSocket framework

- 10) */telemetry-system/subscriptions/persistent-subscriptions/persistent-subscription/config/encoding* → Selection of the specific encoding or RPC framework for telemetry messages to and from the network element for configuration and operational state data. The following options are admitted, where at least PROTO BUFFERS should be supported.

ENCODING	DESCRIPTION
ENC_XML	XML encoding
ENC_JSON_IETF	JSON encoded based on IETF draft standard
ENC_PROTO3	Protocol buffers v3 (Compact and Self-Describing GPB)

- 11) */telemetry-system/subscriptions/persistent-subscriptions/persistent-subscription/sensor-profiles/sensor-profile/config/sensor-group* → Reference to the sensor group which is used in the profile
- 12) */telemetry-system/subscriptions/persistent-subscriptions/persistent-subscription/sensor-profiles/sensor-profile/config/sample-interval* → Time in milliseconds between the device's sample of a telemetry data source. For example, setting this to 100 would require the local device to collect the telemetry data every 100 milliseconds. There can be latency or jitter in transmitting the data, but the sample must occur at the specified interval. The timestamp must reflect the actual time when the data was sampled, not simply the previous sample timestamp + sample-interval. If sample-interval is set to 0, the telemetry sensor becomes event based. The sensor must then emit data upon every change of the underlying data source.
- 13) */telemetry-system/subscriptions/persistent-subscriptions/persistent-subscription/sensor-profiles/sensor-profile/config/heartbeat-interval* → Maximum time interval in seconds that may pass between updates from a device to a telemetry collector. If this interval expires, but there is no updated data to send (such as if suppress\_updates has been configured), the device must send a telemetry message to the collector.
- 14) */telemetry-system/subscriptions/persistent-subscriptions/persistent-subscription/sensor-profiles/sensor-profile/config/suppress-redundant* → Boolean flag to control suppression of redundant telemetry updates to the collector platform. If this flag is set to TRUE, then the collector will only send an update at the configured interval if a subscribed data value has changed. Otherwise, the device will not send an update to the collector until expiration of the heartbeat interval.
- 15) */telemetry-system/subscriptions/persistent-subscriptions/persistent-subscription/destination-groups/destination-group/config/group-id* → The destination group id references a reusable group of destination addresses and ports for the telemetry stream.

On the other hand, the following parameters are for dynamic subscription and, as stated above, are read-only (state tree):

- 16) */telemetry-system/subscriptions/dynamic-subscriptions/dynamic-subscription/state/id* → System generated identifier of the telemetry subscription.
- 17) */telemetry-system/subscriptions/dynamic-subscriptions/dynamic-subscription/state/destination-address* → IP address of the telemetry stream destination
- 18) */telemetry-system/subscriptions/dynamic-subscriptions/dynamic-subscription/state/destination-port* → Protocol (udp or tcp) port number for the telemetry stream destination
- 19) */telemetry-system/subscriptions/dynamic-subscriptions/dynamic-subscription/state/sample-interval* → ime in milliseconds between the device's sample of a telemetry data source. For example, setting this to 100 would require the local device to collect the telemetry data every 100 milliseconds. There can be latency or jitter in transmitting the data, but the sample must occur at the specified interval. The timestamp must reflect the actual time when the data was sampled, not simply the previous sample timestamp + sample-interval. If sample-interval is set to 0, the telemetry

- sensor becomes event based. The sensor must then emit data upon every change of the underlying data source.
- 20) /telemetry-system/subscriptions/dynamic-subscriptions/dynamic-subscription/state/**heartbeat-interval** → Maximum time interval in seconds that may pass between updates from a device to a telemetry collector. If this interval expires, but there is no updated data to send (such as if suppress\_updates has been configured), the device must send a telemetry message to the collector.
  - 21) /telemetry-system/subscriptions/dynamic-subscriptions/dynamic-subscription/state/**suppress-redundant** → Boolean flag to control suppression of redundant telemetry updates to the collector platform. If this flag is set to TRUE, then the collector will only send an update at the configured interval if a subscribed data value has changed. Otherwise, the device will not send an update to the collector until expiration of the heartbeat interval.
  - 22) /telemetry-system/subscriptions/dynamic-subscriptions/dynamic-subscription/state/**originated-qos-marking** → DSCP marking of packets generated by the telemetry subsystem on the network device.
  - 23) /telemetry-system/subscriptions/dynamic-subscriptions/dynamic-subscription/state/**protocol** → Selection of the transport protocol for the telemetry stream.
  - 24) /telemetry-system/subscriptions/dynamic-subscriptions/dynamic-subscription/state/**encoding** → Selection of the specific encoding or RPC framework for telemetry messages to and from the network element
  - 25) /telemetry-system/subscriptions/dynamic-subscriptions/dynamic-subscription/sensor-paths/sensor-path/state/**path** → Path to a section of operational state of interest (the sensor).

Furthermore, as already noted, to allow access to the router for dynamic dial-in subscriptions from collectors, gRPC inbound connections must be enabled in the NE. This is achieved configuring gRPC-server section in “openconfig-system” module. The **xpaths** for the related parameters are these detailed below:

- 26) /system/gRPC-server/config/**enable** → Enables the gRPC server. The gRPC server is enabled by default
- 27) /system/gRPC-server/config/**port** → TCP port on which the gRPC server should listen
- 28) /system/gRPC-server/config/**transport-security** → Enables gRPC transport security (e.g., TLS or SSL)
- 29) /system/gRPC-server/config/**certificate-id** → The certificate ID to be used for authentication
- 30) /system/gRPC-server/config/**metadata-authentication** → Enables gRPC METADATA authentication.
- 31) /system/gRPC-server/config/**listen-addresses** → The IP addresses that the gRPC server should listen on. This may be an IPv4 or an IPv6 address

The following table contains example values for each one of the parameters above described. The aim of the table is to show how the TE information retrieved from the network looks like.

Parameter	Value (example)
(1) group-id	SDN-Controller_1
(2) destination-address	192.168.1.2
(3) destination-port	57500
(4) sensor-group-id	CPU-Maximum
(5) path	/components/component/cpu/oc-



	cpu:utilization/oc-cpu:state/oc-cpu:max
(6) name	CPU-Monitor
(7) local-source-address	10.10.10.3
(8) originated-qos-marking	1
(9) protocol	STREAM_GRPC
(10) encoding	ENC_PROTO3
(11) sensor-group	CPU-Maximum
(12) sample-interval	20
(13) heartbeat-interval	30
(14) suppress-redundant	true
(15) group-id	SDN-Controller_1
(26) enabled	true
(27) port	9339
(28)transport-security	true
(29) certificate-id	ABCDG
(30) metadata-authentication	true
(31) listen-addresses	172.16.1.2



## 9 References

[RFC 6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <<https://www.rfc-editor.org/info/rfc6241>>.

[Openconfig] <https://www.openconfig.net>

## Glossary

<b>API</b>	Application Programming Interface
<b>DCI</b>	Data Center Interconnection
<b>CAPEX</b>	Capital Expenditure
<b>DCN</b>	Data Communication Network
<b>DWDM</b>	Dense Wavelength Division Multiplexing
<b>EOE</b>	Electrical-Optical-Electrical
<b>FEC</b>	Forward Error Correction
<b>GE</b>	Gigabit Ethernet
<b>HAL</b>	Hardware Abstraction Layer
<b>HW</b>	Hardware
<b>L0/L1</b>	Layer 0 and Layer 1
<b>LAN</b>	Local Area Network
<b>MAN</b>	Metropolitan Area Networks
<b>NMS</b>	Network Management System
<b>MNO</b>	Mobile Network Operator
<b>MUST</b>	Mandatory Use Case Requirements for SDN Transport
<b>NMDA</b>	Network Management Datastore Architecture
<b>NOS</b>	Network Operating System
<b>OCP</b>	Open Compute Project



<b>OLS</b>	Open Line System
<b>ONIE</b>	Open Network Install Environment
<b>OTN</b>	Optical Transport Network
<b>ROADM</b>	Reconfigurable Optical Add-Drop Multiplexer
<b>SAN</b>	Storage Area Network
<b>SDH</b>	Synchronous Digital Hierarchy
<b>SDN</b>	Software Defined Network
<b>SW</b>	Software
<b>TAI</b>	Transponder Abstraction Interface
<b>TIP</b>	Telecom Infra Project
<b>TRS</b>	Technical Requirement Specification
<b>WDM</b>	Wavelength Division Multiplexing
<b>ZTP</b>	Zero Touch provisioning