Device Management Interface Profile and Requirements

Version 1.0info, 1st of October 2018

TR-545

ONF TR-545 Device Management Interface Profile and Requirements

Disclaimer

Important Note

This Technical Recommendation has been approved by the OTCC Project TST under the ONF publishing guidelines for publications that allow Project technical steering teams (TSTs) to authorize publication of project documents, and accordingly carries the designation of "-info" in the document ID. In accordance to these guidelines this document has not been reviewed or approved by the ONF Board.

Open Networking Foundation

1000 El Camino Real, Suite 100, Menlo Park, CA 94025

www.opennetworking.org

Table of Content

List of Figures

List of Tables

Definitions and Abbreviation

Term	Definition
Technology Specific Interface Definition	The Technology Specific Interface Definition shall comprise: - one or several UML classes, which are related to a specific network layer and attached to the ONF Core information model - one or several YANG files, which are derived from the UML classes - descriptions of elements, which are in need of explanation - descriptions of technology specific semantics

1 Introduction

Even well-defined information models are not sufficient for managing a multi-vendor network with third party applications. Further specification is required on the interfaces between the network domain controller and the devices (e.g. router, switch, microwave indoor unit, or mediator) shown in Figure 1 below with orange arrows.

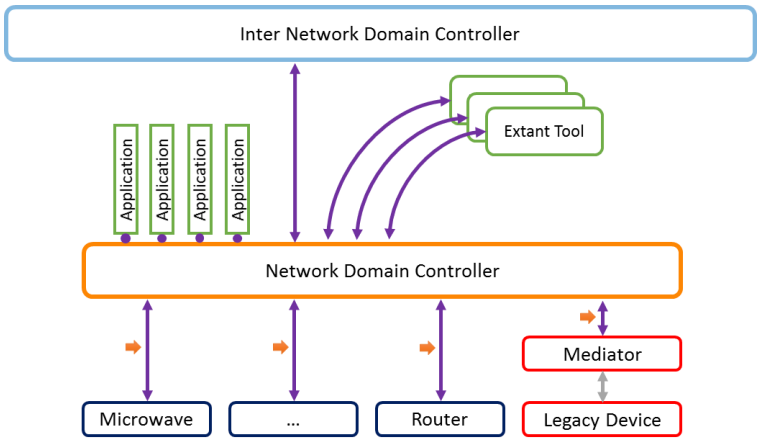

Figure 1: Management Interface

Generic semantics and interface behavior shall be the same across all implementations of management interfaces (e.g. with or without mediator) and independent of the Technology Specific Interface Definition.

A set of mandatory requirements for these generic semantics and behaviors needs to be defined in order to assure compatibility and actual applicability of the respective implementations.

Particularly, if there are several equally good, but different ways of implementing the management interface, specification is required for being able to use a simple design on the application layer.

Whenever possible, specification is done by referencing standards or optional components of standards. Whenever necessary, a textual description is used.

2 Device Management Interface Definition

2.1 Unified Semantic Rules

Even complete and error-free transmitted configuration requests might lead to different actions and responses as a consequence of different interpretation within the respective implementations. Even if there is no clinical right or wrong, a unified way of interpreting the information from the data modeling must be defined.

The following generic semantic rules shall be valid and mandatory for all management interface implementations independently from the Technology Specific Interface Definition.

2.1.1 Successful Configuration

1 The device configuration process shall start with choosing and instantiating an Expected Hardware (e.g. specific board in a slot) from the schemas provided by the device. (Some devices might restrict configuration on components, which are already active. In such case, the device shall either offer just

those types of components, which are already active, to be chosen for being the Expected Hardware or the active hardware shall be automatically considered to be the Expected Hardware.)

2 The Netconf server shall create technology specific layer termination point objects according to the Expected Hardware.

3 <commit> operations for changing the configuration attributes of these layer termination point objects shall be denoted successful, whenever they are in accordance with the Capabilities of the Expected Hardware and pass the validation tests, which are specific for the Expected Hardware.

4 The changed configuration shall be put into operation, or be stored for later operation, in case the Expected Hardware is not active.

5 Hardware differing from the Expected Hardware shall be treated just like missing hardware.

Example: xpicIsAvail expresses, whether the device is capable of XPIC. xpicIsOn expresses, whether XPIC is configured. xpicIsUp expresses, whether XPIC is currently running. In case XPIC would be configured on a microwave device, which is supporting it, but currently holding no radio unit, xpicIsAvail and xpicIsOn would be true, but xpicIsUp would be false.

6 Attributes, which are not supported by the hardware, cannot be changed successfully.

7 All changes must be successful for the entire configuration request being identified successful.

8 As soon as another Expected Hardware gets selected, all affected layer termination point objects (incl. capability information and configuration) plus dependent objects (like e.g. cross connections) shall be deleted.

9 The values, which are shown in the following table, have the following meaning “y”=yes, “n”=no and “0”=no physical hardware at all.

Expected hardware is defined	Configuration changes are valid for expected hardware	Physical hardware is as expected		Configuration is to be denoted "successful"
y	y	y	->	y
y	y	n	->	y
y	y	0	->	y
y	n	y	->	n
y	n	n	->	n
y	n	0	->	n
n	n	n	->	n

Table 1: Successful Configuration

10 In addition, the following status and alarm information shall be given (“y”=yes, “n”=no, “0”=no physical hardware at all, “up”=up and “d”=down):

Expected hardware is defined	Configuration is valid for expected hardware	Physical hardware is as expected		Operational status of hardware incl. configuration	Hardware missmatch alarm
y	y	y	->	up	n
y	y	n	->	d	y
y	y	0	->	d	n

Table 2: Operational Status after Configuration

2.2 Interface Behavior

The applications and controllers need to have correct and up-to-date information about the devices all the time. This requires not just presence of status and performance values in the information model, but also a predictable and unambiguous “behavior” of the management interface. This applies on the ordinary processes and situations, but also on edge cases such as just partly executable configuration requests or loss of the management connection during the processing.

2.2.1 Network Protocol and Connection

[11] Netconf protocol over SSH in accordance with IETF RFC 6242 shall be used on the device management interface.

[12] The SSH shall be implemented in accordance with IETF RFC 4253. [Ruling of the optional components of RFC 4253 will be added in the next release of the document.]

2.2.2 Management Protocol

[13] Netconf shall be implemented according to IETF RFC 6241.

[14] RFC 6241 contains optional elements. RFC 6241 allows to proprietarily amend the Netconf implementation. This results in the implicit allowance of divergent implementations of optional elements. For preventing incompatible Netconf implementations, this document prescribes that Netconf protocol (incl. optional elements of it) shall be implemented as described in RFC 6241, even if RFC 6241 would allow individual modifications or amendments.

The optional Netconf capabilities, which are defined in chapter 8 of RFC 6241, shall be implemented according to the following list:

Capability	Impl.
[15] Writable-Running	No[1]
[16] Candidate Configuration	Yes[2]
[17] Confirmed Commit	Yes[3]
Rollback-on-Error	optional
[18] Validate	Yes[4]
[19] Distinct Startup	No[5]
URL	optional
XPath	optional

Table 3: Netconf Capabilities

[20] During writing into the candidate configuration data store, the single attribute's value changes shall not be subject to validation testing. The <test-option> element provided by the <edit-config> operation shall always have "set" as value to implement this behavior.

[21] Validation testing shall solely be performed on the entire candidate configuration during <commit> operation or <validate> operation.

[22] Notification provisioning shall be implemented according to IETF RFC 5277. All optional components of RFC 5277 shall be supported.

2.2.3 Initial Connection

[23] The device shall implement a DHCP client according to IETF RFC 2131, which also supports the DHCP Options according to IETF RFC 2132.

[24] This DHCP client shall be activated as a factory default.

[25] The graphical management user interface of the device shall facilitate the option to manually deactivate the DHCP client.

The device's DHPC client's DHCPREQUEST shall also contain the following information:

[1] It shall not be possible to make <edit-, <copy- or <delete-config> operations directly on the running configuration data store.
[2] It shall be possible to write single or multiple changes into a candidate configuration data store and initiate actual configuring of the device by a single <commit> operation.
[3] It shall also be possible to expand the <commit> to a confirmed <commit> operation including the <persist> parameter. (This shall allow optionally requesting for automated roll-back of configuration tasks, which might potentially break the management connection.)
[4] The <validate> operation shall check the content of the candidate configuration data store for syntax errors, missing or invalid parameters, references to undefined configuration data, or any other violations of rules established by the underlying data model and generic or technology specific semantic errors.
[5] It shall not be possible to make <edit-, <copy- or <delete-config> operations directly on the distinct startup configuration data store.

Information	Field / Option Code
[26] Device type identifier	60 : Vendor class identifier

Table 4: Option Codes in the DHCPREQUEST

The device shall also apply the following information from the DHCP server's DHCPACK:

Information	Field / Option Code
[27] Device Own IP Address	yiaddr : 'your' (client) IP address
[28] Subnet Mask	1 : Subnet Mask
[29] Default Gateway	3 : Router Option
[30] Domain Name Server	6 : Domain Server
[31] SDN Controller IP Address	43 : Vendor-specific information + 224 : SDN Controller IP Addresses

Table 5: Option Codes in the DHCPACK

[32] Option code 43 shall be used for transmitting IP addresses of SDN controllers as an encapsulated "vendor-specific option field" (see IETF RFC 2132 chapter 8.4).

[33] The "vendor-specific option field" of the SDN controller IP address shall be identified by code 224.

[34] Option field 224 shall be able to transmit several SDN controller IP addresses. These IP addresses shall be expressed as sequence of four bytes. So the length value of option field 224 shall be multiples of 4Byte.

Example:

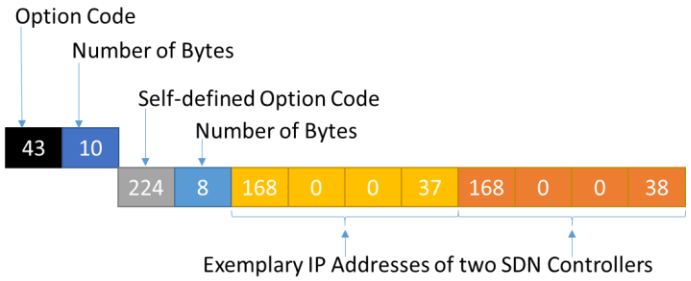

Figure 2: DHCP Option Code 43

[35] Netconf Call Home shall be implemented according to IETF RFC 8071.

[36] The device shall apply Netconf Call Home to connect to the IP address of an SDN controller.

[37] The Netconf server shall support Netconf Monitoring according to IETF RFC 6022, including retrieving the supported schemas via the <get-schema> operation.

2.2.4 Downtime of User Plane

[38] A successful <commit> operation shall be implemented in a way minimizing downtime of the user plane.

2.2.5 Failed <commit> Operation

[39] If the validation of the candidate data store fails or no sequence of configuration tasks, which implements all changes described in the candidate data store, could be found, the affected object and all corresponding underlying resources must remain unchanged.

[40] The failed <commit> operation must not affect the user plane and all related statistics must be continuously collected.

2.2.6 Persistent Configuration

[41] The device shall restart (e.g. after a power failure or reset) using the exact same configuration as before restarting. The running configuration data store shall either be persistent or be automatically backed up to a persistent start-up data store right after every successful modification.

2.2.7 Notifications

[42] Notifications shall be sent as a consequence of changes of editable attributes in the running configuration data store (after <commit> operations), but not in the candidate configuration data store (after <edit-, <copy- or <delete-config> operations).

[43] Changes in the persistent startup data store shall not be notified, because they can't be caused by direct <edit-, <copy- or <delete-config> operations, but only by backing up the running configuration data store

2.2.8 Unsupported Functionality

[44] The Netconf server shall answer with the default value from the data model, each time some information request is addressing an attribute, which is belonging to a functionality that is not supported by the device's hardware.

[45] The Netconf server shall reject <commit> operations that comprise a value change on an attribute, which is not supported by the device's hardware.

[46] The Netconf server shall answer with an <rpc-error> element (error-tag = invalid-value) on a validation operation on a candidate configuration data store, which contains a value change on an attribute, which is not supported by the device's hardware.

2.2.9 Object Naming

[47] Identifier of data objects, which have been created by the Netconf server, shall be unique within the device.

[48] The client shall not assume that the identifiers of data objects are coding any semantics.

[49] Identifiers shall be invariant.

2.3 Interface Performance

2.3.1 Key Performance Indicators

The correct and valid implementation of the modeling and the interface behavior is useless, if performance of the interface is so poor that the implementation cannot be practically used in an operator's network.

Any implementation being in compliance with this Device Management Interface Definition must meet the following response times[6] for at least one exemplary attribute of the model, but not necessarily all attributes:

[50] Reading out a single attribute[7] from the device's or mediator's candidate configuration data store to the Interface Validator application < 55ms.

[51] Reading out a single attribute from the device's or mediator's running configuration data store to the Interface Validator application < 60ms [8].

[52] Writing a single attribute's value from the Interface Validator application to the device's or mediator's candidate configuration data store < 62ms.

[53] Writing a single attribute's value from the Interface Validator application to the device's running configuration data store < 150ms [9].

2.3.2 Test Environment

The aforementioned key performance indicators have to be measured under the following conditions:

- The Interface Validator shall be used for measuring the key performance indicators.
- The Interface Validator virtual machine shall run on a KVM hypervisor.
- The KVM hypervisor shall run on Ubuntu Server 18.04 LTS operating system.
- The Ubuntu Server 18.04 LTS operating system shall run on a PC with an Intel I5 or better performing.
- In order to avoid left over effects from previous tests, a new out of the box installation shall be used.
- In order to eliminate effects from limited computing resources, a process monitoring tool (e.g. top) shall be used to verify resource situation prior and after the performance test. CPU load shall not exceed 50% before and after the test execution.
- More than 1.5 times the amount of specified as needed free memory shall be available.
- The network connections operating at the highest possible speed (e.g. Gigabit Ethernet) is to be verified prior to the test execution.
- If any networking equipment would be required on the interface between controller and device under test (mediator or device), only equipment supporting wire speed shall be used.
- For the measurement of response times, a protocol analyzer (e.g. Wireshark) has to be inserted into the connection between network domain controller and the device or mediator. It has to be located as close as possible to the north bound interface of the device or mediator. Response times shall be measured by time stamp differences of commands from network controller to the mediator and answers from the mediator to the controller.

In case of mediator-supported interfaces, the following conditions shall be additionally applied:

- A single mediator instance shall be run.
- A Linux virtual machine shall run the mediator.

---

[6] 10 measurements have to be performed. The minimum and the maximum value should be disregarded and the average of the remaining 8 values must keep the time limit.
[7] Single information element which is natively provided by the network element and only the protocol needs to be transformed in case a mediator is applied.
[8] Reading from the running configuration data store might take longer, because two interfaces are daisy-chained in case of mediator-based implementations.
[9] Please, be aware that this is a combination of two operations. First, <edit-config> on the candidate configuration data store. Second, <commit> for copying into the running configuration data store. In case of mediator-based implementations, two interfaces are daisy-chained during the <commit> operation.

- Hypervisor, host operating system and further infrastructure shall be separate, but equal to the infrastructure used for the Interface Validator.
- There shall be a direct connection between mediator and device.

2.4 Mediator Behavior

2.4.1 Real-time Data Provisioning

Mediator implementations shall not represent outdated information.

[54] Running configuration, status, alarm and current performance information, which is provided by the mediator must represent the current status of the device.

[55] Candidate configuration, device capability and historical performance information might be buffered.

[56] Values, which are required for providing Notifications, must be polled, if the underlying Legacy Interface does not support Notifications.

2.5 Mediator Resource Consumption

2.5.1 Indicators

The correct and valid implementation of the modeling and the interface behavior is useless, if resource consumption of a mediator is so high that it cannot be scaled according to the requirements of a real operator's network.

Any mediator implementation being in compliance with this Device Management Interface Definition, must meet the following resource consumption indicators:

[57] A virtual machine consuming 8 virtual CPUs, 16GByte random access memory and 64GByte disc space shall be able to run 256 instances of the mediator.

2.5.2 Test Environment

Aforementioned resource consumption indicators have to be measured under the following conditions:

- A Linux virtual machine (8 virtual CPUs, 16GByte random access memory and 64GByte disc space) shall run the 256 mediator instances.
- Hypervisor, host operating system and further infrastructure shall be separate, but equal to the infrastructure used for the Interface Validator.
- Corresponding devices shall be connected via Ethernet Switches with wire speed throughput.
- Test cases shall be executed as synchronized requests from the network controller to the mediators.

2.6 Interface Simulator

No description could ever be as complete as an example. Thus, application development is supported by provisioning of a reference implementation, which is simulating the interface.

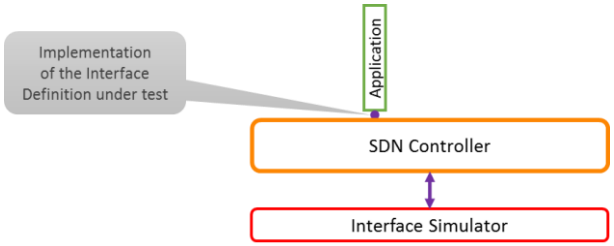

Figure 3: Interface Simulator

A reference for downloading the Interface Simulator can be found in the Technology Specific Interface Definition.

[58] Error-free testing with the Interface Simulator is one of the prerequisites for an application being in compliance with both the Technology Specific Interface Definition and the Device Management Interface Definition.

2.7 Interface Validator

The Interface Validator supports device vendors in hardware and mediator development, and network operators in automating the acceptance testing.

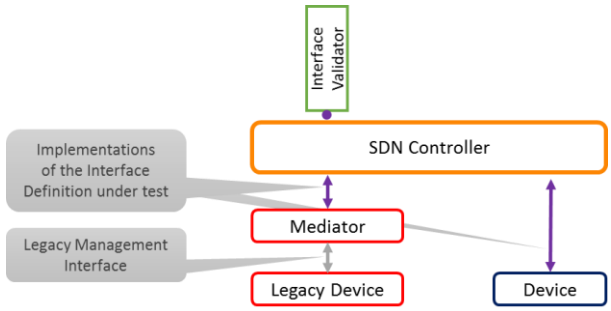

Figure 4: Interface Validator

A reference for downloading the Interface Validator can be found in the Technology Specific Interface Definition.

The Interface Validator shall at least comprise the following testing:

- Completeness and correctness of the implementation of the information model (e.g. Capabilities of the device shall be listed, presence and data type of the attributes shall be verified …)
- Completeness and correctness of the interface behavior (e.g. candidate configuration data store is available, confirmed <commit> operation is supported …)
- Compliance to the performance requirements
- Technology specific correctness of the implementation (e.g. the size of the reduction of the receive level (on the remote site), which is caused by increasing the modulation, corresponds to the difference of the values of the maximum transmit power, which are stated in the device capabilities)

[59] Error-free testing with the Interface Validator is one of the prerequisites for an interface implementation being in compliance with both the Technology Specific Interface Definition and the Device Management Interface Definition.

3 References

Reference	Comment
IETF RFC 2131	Dynamic Host Configuration Protocol, March 1997 (https://tools.ietf.org/html/rfc2131)
IETF RFC 2132	DHCP Options and BOOTP Vendor Extensions, March 1997 (https://tools.ietf.org/html/rfc2132)
IETF RFC 4253	The Secure Shell (SSH) Transport Layer Protocol, January 2006 (https://tools.ietf.org/html/rfc4253)
IETF RFC 5277	NETCONF Event Notifications, July 2008 (https://tools.ietf.org/html/rfc5277)
IETF RFC 6022	YANG Module for NETCONF Monitoring, October 2010 (https://tools.ietf.org/html/rfc6022)
IETF RFC 6241	Network Configuration Protocol (NETCONF), June 2011 (https://tools.ietf.org/html/rfc6241)
IETF RFC 6242	Using the NETCONF Protocol over Secure Shell (SSH), June 2011 (https://tools.ietf.org/html/rfc6242)
IETF RFC 7950	The YANG 1.1 Data Modeling Language, August 2016 (https://tools.ietf.org/html/rfc7950)
IETF RFC 8071	NETCONF Call Home and RESTCONF Call Home, February 2017 (https://tools.ietf.org/html/rfc8071)

4 Back Matter

Editor

Thorsten Heinze, Telefonica Germany GmbH & Co. OHG

Contributors

Many thanks to Michael Binder, Giorgio Cazzaniga, Nigel Davis, Paul Doolan, Floyd Goldstein, Petr Jurcik, Wolfgang Kluge, Xiaobing Niu, Danilo Pala, James Ries, Roberto Servadio, Daniela Spreafico, Alexandru Stancu, Beatriz Ugrinovic, Yossi Victor and Nader Zein for contributing, reviewing and commenting the Device Management Interface Profile and Requirements ONF TR-545.

- end of document -