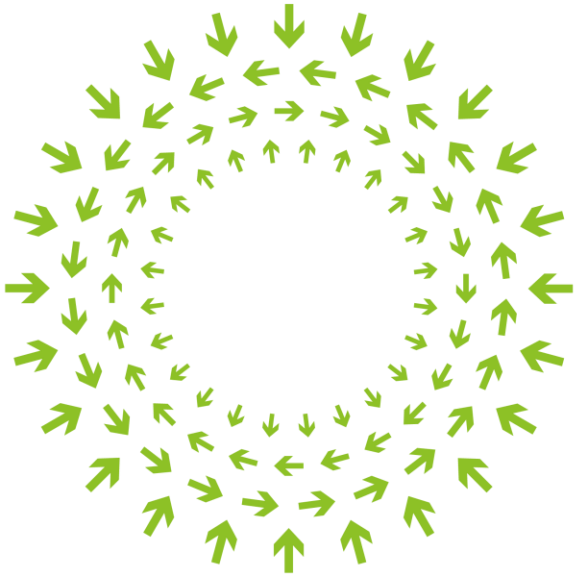


OPEN

Compute Project

Edgecore’s Device Management for TelCo Operators

V2.10

Author: Edgecore

Table of Contents

1. License

Contributions to this proposal are made under the terms and conditions set forth in Open Compute Project Contribution License Agreement (“OCP CLA”) (“Contribution License”) by:

Edgecore Networks

Usage of this Specification is governed by the terms and conditions set forth in Open Compute Project Hardware License – Permissive (“OCPHL Permissive.)

Note: The following clarifications, which distinguish technology licensed in the Contribution License and/or Specification License from those technologies merely referenced (but not licensed), were accepted by the Incubation Committee of the OCP:

[None].

NOTWITHSTANDING THE FOREGOING LICENSES, THIS SPECIFICATION IS PROVIDED BY OCP "AS IS" AND OCP EXPRESSLY DISCLAIMS ANY WARRANTIES (EXPRESS, IMPLIED, OR OTHERWISE), INCLUDING IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, FITNESS FOR A PARTICULAR PURPOSE, OR TITLE, RELATED TO THE SPECIFICATION. NOTICE IS HEREBY GIVEN, THAT OTHER RIGHTS NOT GRANTED AS SET FORTH ABOVE, INCLUDING WITHOUT LIMITATION, RIGHTS OF THIRD PARTIES WHO DID NOT EXECUTE THE ABOVE LICENSES, MAY BE IMPLICATED BY THE IMPLEMENTATION OF OR COMPLIANCE WITH THIS SPECIFICATION. OCP IS NOT RESPONSIBLE FOR IDENTIFYING RIGHTS FOR WHICH A LICENSE MAY BE REQUIRED IN ORDER TO IMPLEMENT THIS SPECIFICATION. THE ENTIRE RISK AS TO IMPLEMENTING OR OTHERWISE USING THE SPECIFICATION IS ASSUMED BY YOU. IN NO EVENT WILL OCP BE LIABLE TO YOU FOR ANY MONETARY DAMAGES WITH RESPECT TO ANY CLAIMS RELATED TO, OR ARISING OUT OF YOUR USE OF THIS SPECIFICATION, INCLUDING BUT NOT LIMITED TO ANY LIABILITY FOR LOST PROFITS OR ANY CONSEQUENTIAL, INCIDENTAL, INDIRECT, SPECIAL OR PUNITIVE DAMAGES OF ANY CHARACTER FROM ANY CAUSES OF ACTION OF ANY KIND WITH RESPECT TO THIS SPECIFICATION, WHETHER BASED ON BREACH OF CONTRACT, TORT (INCLUDING NEGLIGENCE), OR OTHERWISE, AND EVEN IF OCP HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

2. Scope & Overview

Scope:

Edgecore's Device Management can provide a method for identification and management of target devices. It can also provide data that can be used to analyze and detect possible future device health or out of state issues. This enables the entire infrastructure devices (including networking, storage, server) to be monitored to provide High Availability.

The work, described in this paper, provides means to manage devices though a well-defined data model. It uses the Edgecore's Device Manager to collect and manage device status from all devices, and sends to an Output Bus, using RESTFUL APIS. The functionality also provides support for monitoring certain device states and sending event notifications.

All APIs and functionality set forth in this document will be available on both CPU and BMC sides, unless otherwise noted. The document also introduces Edgecore BMC daughter-card and motherboard reference design providing support to the new features, which is be discussed in detail separately. At the current time, the proposed changes are only planned for vOLT devices.

Overview:

Scalability is one of the paramount problems in today's modern infrastructures. The growing needs of the end customers forces the infra operators to employ horizontal, scale-out solutions, often requiring them to deploy additional devices to their systems. This ever-growing infrastructure requires non-traditional thinking in which a more automated device management functionality must be employed.

Designed to manage large systems through a modern platform, DMTF's Redfish is an open industry standard specification enabling customers to monitor their infrastructure through a well-defined interface.

In its core, each device is monitored and managed through an agent, PSME running on OLT (or OpenBMC running on BMC daughter card, if available), to

1- retrieve specific parameters to be sent through its northbound interfaces.

2- receive notifications on certain device state changes.[1]

3- manage device state and reboot options.

[1] Device Manager will not support notifications setup through BMC.

Edgecore's Device Manager uses the ONLP driver (or BMC when available) to manage and retrieve information about device, and makes data available at a configurable Output Bus. It uses a RESTFUL API to communicate with PSME at certain intervals to retrieve the data .

Other devices on the system, such as Aggregation switches and ONUs, can also use Redfish to report to Edgecore's Device Manager, the same way vOLT does.

To provide uniform functionality across an infrastructure employing a non-uniform device list provided by multiple vendors, Redfish (and OCP) exchanges information using a hardware management specification, employing a JSON schema. This proposal uses standard OCP baseline profile. If needed, we may enhance the baseline profile with attributes specific for TelCo in the future.

This will also work on BMC controller running OpenBMC standard interface.

The following is an example of OCP defined schema

```
Redfish >> v1 >> Chassis

       "@odata.type": "#ChassisCollection.ChassisCollection",
       "Name": "Chassis Collection",
       "Members@odata.count": 2,
       "Members": [
              {
              "@odata.id": "/redfish/v1/Chassis/Switch1"
              } ,
              {
              "@odata.id": "/redfish/v1/Chassis/Switch2"
              }
       ],
       "odata.context":
       "redfish/v1/$metadata#ChassisCollection.ChassisCollection",
       "odata.id"; "redfish/v1/Chassis"
```

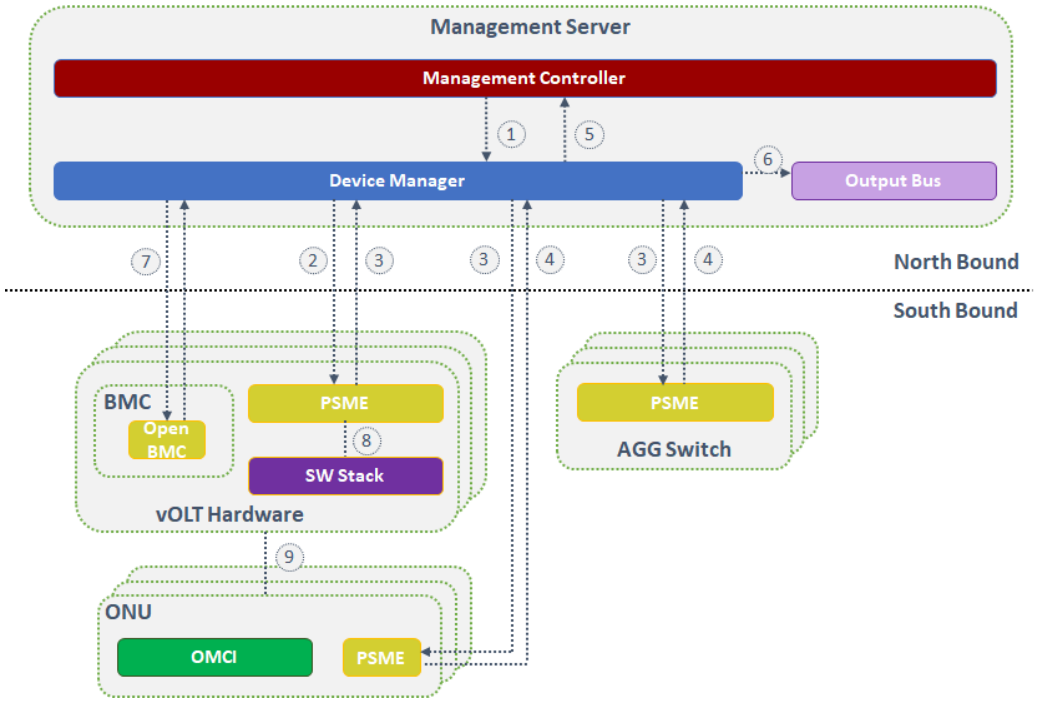
3. Architecture

Edgecore's Device Manager will use a standard interface and dataflow to manage and monitor for deployment in the field. Each device in the system will communicate North Bound with Device Manager to transmit device data and notifications, using PSME (or OpenBMC in BMC), adhering Redfish standards. Device Manager will collect device data and notifications from each device, and make the data available on a predetermined Output Bus for consumers. The consumers of this data could then use it for various purposes, including

- inventory management
- status monitoring
- device updates

- reboots

Following diagram illustrates the overall architecture and the interaction between the components at a high level.



① : The first step is the notification from Management Controller to add the IP address of the device to be monitored to the device list (Device Manager can also use Ref ⑤ to refresh all its data.) Device Manager can also subscribe to events from the device by sending a HTTP POST to the URL of the Resource Collection for "Subscriptions" in the Event Service. At this point, Device Manager can query status parameters with HTTP GET method to PSME (Pooled System Management Engine) periodically. The default polling interval is set to five seconds.

② : Device Manager is an alert Receiver. When the Redfish service interface is responding to Device Manager, it uses RESTful API. If PSME detects a change in one of the monitored hardware states, it will send an alert for the subscribed event to the Device Manager. A particular issue is reported only once until the state of the hardware changes.

③ : Device Manager retrieves data using HTTP GET method from devices periodically.

④ : Switch responds using HTTP RESPONSE to Device Manager.

⑤ : The Management Controller will provide the active device list to Device Manager which needs to be monitored. When the device becomes available, the Device Manager uses this IP address to connect PSME (or OpenBMC) using HTTP protocol.

⑥ : Device Manager publishes the device alarm/status into Output Bus.

⑦ : If exists, BMC can also be used to retrieve device status and manage the target device. However, BMC may not provide the full functionality offered by PSME, including periodic data collection or alarms.

⑧ : PSME inspects hardware status at regular intervals to detect hardware failure/status through Network OS's SW stack.

4. API Details

4.1 Authentication

Redfish Service uses session management to implement authentication.

4.1.1 Enable/Disable Session Management

Enable secure access to resources.

POST: /redfish/v1/SessionService

```
{
"ServiceEnabled": true,
"SessionTimeout": 300
}

- Response : 200 OK
```

Property	Requirement	Value
ServiceEnabled	Mandatory	true, false
SessionTimeout	Mandatory	≥1 [in seconds]

Following is a mockup of get the Session Service State

GET: /redfish/v1/SessionService

```
{
"ServiceEnabled": true,
"SessionTimeout": 300
}

- Response : 200 OK
```

Device Management gRPC definition	
gRPC API name	**SetSessionService** (DeviceAccount)
gRPC Arguments	**IpAddress** (String): device IP Address and port. **UserToken** (String): device user (Administrator) login permission code.

	SessionEnabled (Bool): true(enable session service), false(disable session service). **SessionTimeout** (uint64): What are the many timeouts for device session service?
gRPC API return	None

Create a New Session

Create a new session (Login) by using the login credential of an existing account (default username "admin" and Password "redfish"), enable Session Management, first. Include the token is generated in the https request header to perform any operations on the secured resources.

POST: /redfish/v1/SessionService/Sessions

```
{
"UserName": "admin",
"Password": "redfish"
}


- Response : 201 CREATED
```

Property	Requirement	Value
UserName	Mandatory	
Password	Mandatory	

Device Management gRPC definition	
gRPC API name	**LoginDevice** (DeviceAccount)
gRPC Arguments	**IpAddress** (String): device IP Address and port. **ActUsername** (String): login this user name. **ActPassword** (String): login this user password.
gRPC API return	DeviceAccount (message): **Httptoken**: device user login permission code.

Following is a mockup of the session token generated

```
{
        "X-Auth-Token": "5HHYVsfNOb9Dm4X0PVmQcK9aJnWUMQXU"
}
```

4.1.2 Delete an Existing Session

Delete an existing session (Logout) by using default UserName "admin" and Password "redfish". Following example assumes a session with ID "1" has already been created.

DELETE: /redfish/v1/SessionService/Sessions/<session_id>

```
- Response : 200 OK
```

Device Management gRPC definition	
gRPC API name	**LogoutDevice** (DeviceAccount)
gRPC Arguments	**IpAddress**(String): device IP Address and port **UserToken**(String): device user login permission code. The user logout follows the permission by this rule. . Administrator: logout all users. . Operator: logout “Operator” and “ReadOnlyUser” users. . ReadOnlyUser: only logout “ReadOnlyUser” user. **ActUsername**(String): logout this user name.
gRPC API return	None

4.2 Account Service

Account Service contains properties common to all user accounts, such as password requirements, and control features such as account lockout. It also contains links to the collections of Manager Accounts and Roles.

4.2.1 View Accounts

List default accounts.

GET: /redfish/v1/AccountService/Accounts

Following is a mockup of the accounts list

```
{
        "@odata.context": "/redfish/v1/$metadata#AccountService.AccountService",
        "@odata.id": "/redfish/v1/AccountService/Accounts",
        "@odata.type": "#ManagerAccountCollection.ManagerAccountCollection",
        “Name": "Accounts Collection",
        "Members@odata.count": 1,
        "Members": [
        {
                "@odata.id": "/redfish/v1/AccountService/Accounts/admin"
        } ]
}

- Response : 200 OK
```

Device Management gRPC definition	
gRPC API name	**ListDeviceAccounts** (DeviceAccount)
gRPC Arguments	**IpAddress** (String): device IP Address and port **UserToken**(String): device user (Administrator) login permission code.
gRPC API return	DeviceAccountList (message): **Account**: device user list.

4.2.2 Create a New Account

Create a new account.

POST: /redfish/v1/AccountService/Accounts

```
   {
   "Name":"Name_1",
   "UserName":"User_Name_1",
   "Password":"User_Password_1",
   "RoleId":"Administrator",
   "Enabled":true ,
   "Locked":false
   }

- Response : 201 CREATED
```

Property	Requirement	Value
UserName	Mandatory	
Password	Mandatory	
RoleId	Mandatory	Administrator, Operator, ReadOnlyUser
Enabled	Mandatory	true, false
Locked	Mandatory	true, false

Following is an account generated

GET: /redfish/v1/AccountService/Accounts/User_Name_1

```
{
"@odata.context": "/redfish/v1/$metadata#ManagerAccount.ManagerAccount",
"@odata.id": "/redfish/v1/AccountService/Accounts/User_Name_1",
"@odata.type": "#ManagerAccount.v1_0_0.ManagerAccount",
"Id": "User_Name_1",
"Name": "Name_1",
"Description": null,
"UserName": "User_Name_1",
"Password": null,
"Locked": false,
"Enabled": true,
"RoleId": "Administrator",
"Links":{
"Role":{
"@odata.id": "/redfish/v1/AccountService/Roles/1"
}
}
}

- Response : 200 OK
```

Device Management gRPC definition	
gRPC API name	**CreateDeviceAccount** (DeviceAccount)
gRPC Arguments	**IpAddress**(String): device IP Address and port. **UserToken**(String): device user (Administrator) login permission code. **ActUsername**(String): add this user name. **ActPassword** (String): add this user password. **Privilege** (String): add supported privilege (Administrator/Operator/ReadOnlyUser).
gRPC API return	None

4.2.3 Delete an Account

Delete an existing account.

DELETE: /redfish/v1/AccountService/Accounts/<user_name>

```
- Response : 200 OK
```

Device Management gRPC definition	
gRPC API name	**RemoveDeviceAccount** (DeviceAccount)
gRPC Arguments	**IpAddress**(String): device IP Address and port. **UserToken**(String): device user (Administrator) login permission code. **ActUsername**(String): delete this user name.
gRPC API return	None

4.2.4 Chang Account Password

Administrator can change user's password

PATH: redfish/v1/AccountService/Accounts/admin

```
{
"Password": "redfish_new_password"
}

- Response : 200 OK
```

Device Management gRPC definition	
gRPC API name	**ChangeDeviceUserPassword** (DeviceAccount)
gRPC Arguments	**IpAddress**(String): device IP Address and port. **UserToken**(String): device user (Administrator) login permission code.

	ActUsername(String): delete this user name. **ActPassword**(String): new user password.
gRPC API return	None

4.3 Hardware Information

Hardware inventory and health state information, such as information about temperature sensors, fans, PSU, HDD, NIC, can be obtained using Redfish API's.

4.3.1 Chassis information

Retrieve chassis information.

GET: /redfish/v1/Chassis/<chassis_id>

Following is a mockup of chassis info retrieved for Edgecore's GPON device x86-64-accton-asgvolt64-r0


```
{
"@odata.context": "/redfish/v1/$metadata#Chassis.Chassis",
"@odata.id": "/redfish/v1/Chassis/1",
"@odata.type": "#Chassis.v1_3_0.Chassis",
"Id": "1",
"ChassisType": "Drawer",
"Name": "Chassis",
"Description": "ASGvOLT64-O-AC-F",
"PowerState": "On",
"Manufacturer": "Accton",
"Model": "x86_64-accton_asgvolt64-r0",
"SKU": null,
"SerialNumber": "EC1921003307",
"PartNumber": "FN1EC0964000Z",
"AssetTag": “”,
"IndicatorLED": "Lit",
"Status":{
"State": "Enabled",
"Health": "OK",
"HealthRollup": "OK"
        },
                "@odata.type": "#Intel.Oem.Chassis", {
                "Location": {
                        "Id": null,
                        "ParentId": null
                },
                "Links": {
                        "@odata.type": "#Chassis.v1_2_0.Links",
                        "Contains":[],
                        "ComputerSystems": [ {
                                "@odata.id": "/redfish/v1/Systems/1",
                                "Actions": {
                                        "#Chassis.Reset":{"target":
                                        "/redfish/v1/Chassis/1/Actions/Chassis.R
                                        eset",
                                        "ResetType@Redfish.AllowableValues":
                                        ["ForceOff",¡K}, "Thermal":{"@odata.id":
                                        "/redfish/v1/Chassis/1/Thermal"},
```

```
                                                    "Power":{"@odata.id":
                                                    "/redfish/v1/Chassis/1/Power" } ]
                                    …… }
            ……
}
}


- Response : 200 OK
```

Device Management gRPC definition	
gRPC API name	**GetDeviceData** (Device)
gRPC Arguments	**IpAddress** (String): device IP Address and port **UserToken**(DeviceAccount)(String): device user login permission code. **RedfishAPI** (String): /redfish/v1/Chassis/<chassis_id>.
gRPC API return	DeviceData (message): **DeviceData**: device data by “RedfishAPI”.

4.3.2 Thermal Sensor Information

Fan speeds can be read from "Reading" property in payload of Fans object. PSU power thermal sensor can be read from "ReadingCelsius" property in payload of Temperatures object.

GET: /redfish/v1/Chassis/<chassis_id>/Thermal

Following is a mockup of thermal snsor info retrieved for Edgecore’s XGSPON device x86-64-accton-asxvolt16-r0

```
{
            "Description": "Collection of Thermal Sensors",
            "Redundancy":[],
            "Temperatures":[
            {
            "@odata.id": "/redfish/v1/Chassis/1/Thermal",
            "MemberId": "1",
            "Name": "System CPU Thermal Sensor Temperature",
            "PhysicalContext": "CPU",
            "SensorNumber": 1,
            "Status":{"HealthRollup": "OK", "State": "Enabled"},
            "ReadingCelsius": 49,
            "UpperThresholdNonCritical": 83,
            "UpperThresholdCritical": 93,
            "UpperThresholdFatal": 105,
            "RelatedItem":[{"@odata.id": "/redfish/v1/Chassis/1" }]
            },
            {
            "@odata.id": "/redfish/v1/Chassis/1/Thermal",
            "MemberId": "2",
            "Name": "Chassis Thermal Sensor Temperature",
```

```
            "PhysicalContext": "SystemBoard",
            "SensorNumber": 2,
            "Status":{"HealthRollup": "OK", "State": "Enabled"},
            "ReadingCelsius": 35,
            "UpperThresholdNonCritical": 83,
            "UpperThresholdCritical": 93,
            "UpperThresholdFatal": 105,
            "RelatedItem":[{"@odata.id": "/redfish/v1/Chassis/1" }]
            },…
}

- Response : 200 OK
```


Device Management gRPC definition	
gRPC API name	**GetDeviceTemperatures** (DeviceTemperature)
gRPC Arguments	**IpAddress** (String): device IP Address and port **UserToken**(DeviceAccount)(String): device user login permission code.
gRPC API return	DeviceTemperature (message): **TempData**: device temperature data.

4.3.3 PSU Power Information

Power consumption information can be read from the "**PowerConsumedWatts**" property in the payload of the PowerControl object. PSU presence information can be read from the “State” property in the payload of “Status” object.

GET: /redfish/v1/Chassis/<chassis_id>/Power


```
{
"@odata.context": "/redfish/v1/$metadata#Power.Power",
"@odata.id": "/redfish/v1/Chassis/1/Power",
"Id": "Power",
"@odata.type": "#Power.v1_1_0.Power",
"Name": "Power Collection",
"Description": "Collection of Power",
"PowerControl":[
{"@odata.id": "/redfish/v1/Chassis/1/Power", "MemberId": "1", "Name": "System Power Control",…},
{
"@odata.id": "/redfish/v1/Chassis/1/Power",
"MemberId": "2",
"Name": "System Power Control",
"PowerConsumedWatts": 146,
"Status":{"Health": "OK", "State": "Enabled"},
"PowerRequestedWatts": null,
"PowerAvailableWatts": null,
"PowerCapacityWatts": 0,
"PowerAllocatedWatts": null,
"PowerLimit":{"LimitInWatts": null, "LimitException": null, "CorrectionInMs": null},
"RelatedItem":[{"@odata.id": "/redfish/v1/Chassis/1" }],
"Oem":{}
}
```

```
],
….
}
],
"Oem":{}
}

- Response : 200 OK
```

Device Management gRPC definition	
gRPC API name	**GetDeviceData** (Device)
gRPC Arguments	**IpAddress** (String): device IP Address and port **UserToken**(DeviceAccount)(String): device user login permission code. **RedfishAPI** (String): /redfish/v1/Chassis/<chassis_id>/Power.
gRPC API return	DeviceData (message): **DeviceData**: device data by “RedfishAPI”.

4.3.4 Management Port Information

4.3.4.1 Collection of Ports

Retrieve the port information

GET: /redfish/v1/ Managers/1/EthernetInterfaces/<ethernetswitch_id>

Following is a mockup of list of ports

```
{
        "@odata.context":"/redfish/v1/$metadata#EthernetInterface.EthernetInterf
ace",
        "@odata.id":"/redfish/v1/Managers/1/EthernetInterfaces/1",
        "@odata.type":"#EthernetInterface.v1_4_0.EthernetInterface",
        "Id":"1",
        "Name":"Management port","Description":"Management port settings",
        "Status":{
              "State":"Enabled",
              "Health":"OK",
              "HealthRollup":"OK"
        },
        "Oem":{},
        "MACAddress":"a8:2b:b5:e7:8e:62",
        "PermanentMACAddress":"a8:2b:b5:e7:8e:62",
        "SpeedMbps":100,
        "AutoNeg":true,
        "FullDuplex":true,
        "MTUSize":1500,
```

```
        "HostName":"localhost",
        "FQDN":"localhost",
        "MaxIPv6StaticAddresses":null,
        "InterfaceEnabled":true,
        "LinkStatus":"LinkUp",
        "IPv4StaticAddresses":[
            {
                "Address":"172.17.8.40"
            }
        ],
        "IPv4Addresses":[
            {
                "Address":"172.17.8.40",
                "SubnetMask":"255.255.252.0",
                "AddressOrigin":"Static",
                "Gateway":"172.17.10.251"
            }
        ],
        ……
}

- Response : 200 OK
```


Device Management gRPC definition	
gRPC API name	**GenericDeviceAccess** (Device)
gRPC Arguments	**IpAddress** (String): device IP Address and port **UserToken**(DeviceAccount)(String): device user login permission code. **HttpMethod** (String): “GET” **RedfishAPI** (String): /redfish/v1/Managers/1/EthernetInterfaces/1
gRPC API return	HttpData (message): **resultData**: device data by “RedfishAPI”.

4.3.4.2 Port Information

Retrieve detailed information about a port, if **present**, and if supported transceivers are detected, their detailed information.

GET: /redfish/v1/EthernetSwitches/<ethernetswitch_id>/Ports/<port_id>

Following is a mockup of port details


```
{
        "@odata.context": "/redfish/v1/$metadata#EthernetSwitchPort.EthernetSwitchPort",
        "@odata.id": "/redfish/v1/EthernetSwitches/1/Ports/1",
        "@odata.type": "#EthernetSwitchPort.v1_0_0.EthernetSwitchPort",
        "Id": "1",
```

```
            "Name": "Port1",
            "Description": "Ethernet Switch Port description",
            "PortId": "Port ID",
            "Status":{"State": "Enabled", "Health": "OK", "HealthRollup": "OK"},
            "LinkType": "Ethernet",
            "OperationalState": null,

            "AdministrativeState": null,
            "LinkSpeedMbps": null,
            "NeighborInfo":{"SwitchId": null, "PortId": null, "CableId": null},
            "NeighborMAC": null,
            "FrameSize": null,
            "Autosense": null,
            "FullDuplex": null,
            "MACAddress": null,
            "PortClass": null,
            "PortMode": null,
            "PortType": null,
            "Oem":{},
            "IPv4Addresses":[],
            "IPv6Addresses":[],
            "VLANs":{"@odata.id":"/redfish/v1/EthernetSwitches/1/Ports/1/VLANs"},
            "StaticMACs":{"@odata.id":"/redfish/v1/EthernetSwitches/1/Ports/1/StaticMACs"},
            "Links":{
            …
}

- Response : 200 OK
```


Transceiver	**Vendor**	**Part No**
QSFP	Precision	QSFP28AOC03
XFP (XGSPON)	Source Photonics	XPPXG2N1CDFA
XFP (XGSPON)	Hisense	LTH7226-PC+
SFP (GPON)	Hisense	LTE3680M-BH+

Device Management gRPC definition	
gRPC API name	**GetDeviceData** (Device)
gRPC Arguments	**IpAddress** (String): device IP Address and port **UserToken**(DeviceAccount)(String): device user login permission code. **RedfishAPI** (String): /redfish/v1/EthernetSwitches/<ethernetswitch_id>/Ports/<port_id>.
gRPC API return	DeviceData (message): **DeviceData**: device data by “RedfishAPI”.

4.4 Retrieve Device Data using General gRPC API

The gRPC API supports HTTP methods (GET, POST, PATH and DELETE). The Device Manager will forward your request with method to device and responds the data of device to you. For example:

GET: /redfish/v1/Managers/1/NetworkProtocol


```
{
"@odata.context":
"/redfish/v1/$metadata#ManagerNetworkProtocol.ManagerNetworkProtocol",
"@odata.id": "/redfish/v1/Managers/1/NetworkProtocol",
"@odata.type": "#ManagerNetworkProtocol.v1_0_2.ManagerNetworkProtocol",
"Id": "NetworkProtocol",
"Name": "Manager Network Protocol",
"Description": "Manager Network Protocol description",
"Status":{
"State": "Enabled",
"Health": "OK",
"HealthRollup": "OK"
},
"HostName": "localhost",
"FQDN": "localhost",
"HTTP":{
"ProtocolEnabled": null,
"Port": null
},
"HTTPS":{
"ProtocolEnabled": true,
"Port": 8888
},
"IPMI":{
"ProtocolEnabled": null,
"Port": null
},
"SSH":{
"ProtocolEnabled": true,
"Port": 22
},
"SNMP":{
"ProtocolEnabled": true,
"Port": 161
},
"VirtualMedia":{
"ProtocolEnabled": null,
"Port": null
},
"SSDP":{
"ProtocolEnabled": true,
"Port": 1900,
"NotifyIPv6Scope": null,
"NotifyMulticastIntervalSeconds": 0,
"NotifyTTL": 2
},
"Telnet":{
"ProtocolEnabled": null,
"Port": null
},
"KVMIP":{
"ProtocolEnabled": null,
```

```
"Port": null
},
"Oem":{}
}

- Response : 200 OK
```

Device Management gRPC definition	
gRPC API name	**GenericDeviceAccess** (Device)
gRPC Arguments	**IpAddress** (String): device IP Address and port **UserToken**(DeviceAccount)(String): device user login permission code. **HttpMethod**(String): assign the “GET” method **RedfishAPI** (String): /redfish/v1/Managers/1/NetworkProtocol.
gRPC API return	**HttpData** (message): **ResultData**: device data by “RedfishAPI”.

4.5 Attach the device

The device will present in the management network. The device manger have to know the device IP address and service port.

Device Management gRPC definition	
gRPC API name	**SendDeviceList** (DeviceList.Device)
gRPC Arguments	**IpAddress** (String): device IP Address and port **Frequency** (Uint32): giving the frequency of pulling device data time interval.
gRPC API return	None

4.6 Detach the device

The device could be in the out of network field. The Device Manager have to detach the device with IP address and port.

Device Management gRPC definition	
gRPC API name	**DeleteDeviceList** (Device)
gRPC Arguments	**IpAddress** (String): device IP Address and port. **UserToken**(DeviceAccount)(String): device user login permission code.
gRPC API return	None

4.7 Start pulling the device data

The Device Manager could start to pull the device data periodically. The pulling data will follow these Redfish APIs (/redfish/v1/Chassis) by default.

Device Management gRPC definition	
gRPC API name	**StartQueryDeviceData** (DeviceAccount)
gRPC Arguments	**IpAddress** (String): device IP Address and port. **UserToken**(DeviceAccount)(String): device user login permission code.
gRPC API return	None

4.8 Stop pulling the device data

The Device Manager could stop to pull the device data periodically.

Device Management gRPC definition	
gRPC API name	**StopQueryDeviceData** (DeviceAccount)
gRPC Arguments	**IpAddress** (String): device IP Address and port. **UserToken**(DeviceAccount)(String): device user login permission code.
gRPC API return	None

4.9 List of All Devices

The Device Manager could provide all attached devices list.

Device Management gRPC definition	
gRPC API name	**GetCurrentDevices** ()
gRPC Arguments	None
gRPC API return	**DeviceListByIp** (message): **IpAddress**: device IP address and service port.

4.10 Change query time interval for pulling device data

The Device Manger responses for pulling device data periodically. The query time interval could adjust on the fly.

Device Management gRPC definition	
gRPC API name	**SetFrequency** (FreqInfo)
gRPC Arguments	**IpAddress** (String): device IP Address and port. **UserToken**(DeviceAccount)(String): device user login permission code. **Frequency** (Uint32): giving the frequency of pulling device data time interval.
gRPC API return	None

4.11 List the added the Redfish APIs for pulling device data.

The Redfish APIs will use in pulling device data. The Device Manager could list already added Redfish APIs.

Device Management gRPC definition	
gRPC API name	**GetRfAPIList** (PollingRfAPI)
gRPC Arguments	**IpAddress** (String): device IP Address and port. **UserToken**(DeviceAccount)(String): device user login permission code.
gRPC API return	**RfAPIList** (message): **RfAPIList**: list of Redfish APIs

4.12 Append the Redfish APIs for pulling device data.

The list of Redfish APIs could append to Device Manager for pulling device data.

Device Management gRPC definition	
gRPC API name	**AddPollingRfAPI** (PollingRfAPI)
gRPC Arguments	**IpAddress** (String): device IP Address and port. **UserToken**(DeviceAccount)(String): device user login permission code. **RfAPI** (String): Redfish API
gRPC API return	None

4.1 Delete the Redfish APIs for pulling device data.

The list of Redfish APIs could delete by Device Manager for pulling device data.

Device Management gRPC definition	
gRPC API name	**RemovePollingRfAPI** (PollingRfAPI)
gRPC Arguments	**IpAddress** (String): device IP Address and port. **UserToken**(DeviceAccount)(String): device user login permission code. **RfAPI** (String): Redfish API
gRPC API return	None

4.2 Enable/Disable Log Service

This resource represents the log service for the resource or service to which it is associated. It records all ONL peripheral Add/Remove/Alert events, such as, fan plugging-in and out, thermal sensor exceeding fatal critical threshold temperature, …, etc.

4.2.1 Enable/Disable Log Service

PATCH: /redfish/v1/Managers/<manager_id>/LogServices/<log_id>

```
{
"ServiceEnabled": true
}

- Response : 200 OK
```

Property	Requirement	Value
ServiceEnabled	Mandatory	true, false

Device Management gRPC definition	
gRPC API name	**EnableLogServiceState** (LogService)
gRPC Arguments	**IpAddress** (String): device IP Address and port **UserToken** (String): device user (Administrator or Operator) login permission code. **LogServiceEnabled** (Bool): true: enable, false: disable
gRPC API return	None

4.2.2 Get Log Entries of the Target Device

GET : /redfish/v1/Managers/<manager_id>/LogServices/<log_id>/Entries

Following is a mockup of logs retrieved from the device

```
{
        "@odata.context":
        "/redfish/v1/$metadata#LogEntryCollection.LogEntryCollection",
        "@odata.id": "/redfish/v1/Managers/1/LogServices/1/Entries",
        "@odata.type": "#LogEntryCollection.LogEntryCollection",
        "Name": "Log Service Collection",
        "Description": "Collection of Logs for this System",
        "Members@odata.count": 3,
        "Members":[
            {"@odata.id": "/redfish/v1/Managers/1/LogServices/1/Entries/0","Id": "0"},
            {"@odata.id": "/redfish/v1/Managers/1/LogServices/1/Entries/1","Id": "1"},
            {"@odata.id": "/redfish/v1/Managers/1/LogServices/1/Entries/2","Id": "2"}
        ]
}

- Response : 200 OK
```

Device Management gRPC definition	
gRPC API name	**EnableLogServiceState** (LogService)
gRPC Arguments	**IpAddress** (String): device IP Address and port **UserToken** (String): device user (Administrator or Operator) login permission code. **LogServiceEnabled** (Bool): true: enable, false: disable

gRPC API return	None

4.2.3 Clear Log Entries of the Target Device

POST:/redfish/v1/Managers/<manager_id>/LogServices/<log_id>/Actions/LogService.Reset

```
{ }

- Response : 200 OK
```

Device Management gRPC definition	
gRPC API name	**ResetDeviceLogData** (LogService)
gRPC Arguments	**IpAddress** (String): device IP Address and port **UserToken** (String): device user (Administrator or Operator) login permission code.
gRPC API return	None