Usage Guide and Requirements for OpenRMC Northbound API v1.0.0

Author: John Leung (Intel Corporation)

June 2021

Table of Contents

1. License 3

2. Scope 4

3. Requirements 4

4. Capabilities 4

5. Use Cases 5

5.1. Hardware Inventory 5

5.1.1. Obtain inventory information for the rack manager 5

5.1.2. Obtain inventory information for each node in the rack 5

5.2. Rack Power Status 6

5.2.1. Obtain power state of the rack 6

5.2.2. Obtain power usage for the rack 6

5.3. Rack Power Control 7

5.3.1. Set to limit for power usage for the rack 7

5.4. PSU Status/Health 7

5.4.1. Obtain the status and health of the PSU 7

5.5. Node Power Status 8

5.5.1. Obtain power state of a node 8

5.5.2. Obtain power usage for a node 8

5.6. Node Power Control 9

5.7. Node Temperature 9

5.8. Node Health and Status 10

5.8.1. Obtain the status and health of the node 10

5.8.2. Status and health of the CPUs 11

5.8.3. Status and health of the memory 11

5.8.4. Obtain the state of the LED 11

5.8.5. Retrieve the RMC log 12

5.8.6. Retrieve the System logs 12

5.9. Obtain the firmware revision 13

5.9.1. Obtain the revision of Rack Manager firmware 13

5.9.2. Obtain the revision of the BIOS firmware on each system 13

5.9.3. Obtain the revision of the BMC firmware on each system 13

5.9.4. Obtain the revision of PSU firmware 14

5.10. Update the Rack Manager Firmware 14

5.10.1. Pull Method 14

5.10.2. Push Method 15

5.11. Account Management 15

6. References 16

7. Revision 16

1. License

2. Scope

This document references requirements and provide the usage examples for the OpenRMC northbound API v1.0.0 for a rack management controller.

3. Requirements

As a Redfish-based interface, the required Redfish interface model elements are specified in a profile document.   For the OpenRMC northbound API v1.1.0, the profile is located at –

https://github.com/opencomputeproject/OCP-Profiles/blob/master/OCPRackManagerController.v1_0_0.json

The Redfish Interop Validator is an open source conformance test which reads the profile, executes the tests against an implementation and generates a test report – in text or HTML format.

```
$> python3 RedfishInteropValidator.py profileName --ip host:port
```

The Redfish Interop Validator is located at *https://github.com/DMTF/Redfish-Interop-Validator*.

4. Capabilities

The following use cases and associated resources have been identified to allow RMC interface to expose the rack level manageable capabilities.

Use Case	**Manageable Capabilities**	
Hardware inventory	• Get the position and detail of the managed nodes • FRU information of rack manager • FRU information of each node	Section 5.1
Rack Power Status	• Obtain the rack power readings • Voltage • Current	Section 5.2
Rack Power Control	• Set the rack power usage limit	Section 5.3
PSU Status/Health	• Obtain the status and health of the PSU	Section 5.4
Node Power Status	• Determine the power status of the node • On or Off • Obtain the node power readings • Voltage • Current	Section 5.5
Node Power Control	Power profile	Section 5.6
Node Temperature	• Obtain the node temperature • Celsius or Fahrenheit	Section 5.7
Node Status/Health	• Obtain the status and health of the node • Status and health of the CPUs • Status and health of the memory • Obtain the state of the LED • Retrieve the logs	Section 5.8
Firmware Versions	• Obtain the firmware revision of • Rack Management firmware • BIOS firmware of each node • BMC firmware of each node • PSU firmware	Section 5.9
Firmware Update	Update the firmware on the • Rack Management firmware	Section 5.10

Account Management	Admin/user accounts	Section 5.11

5. Use Cases

This section describes how each capability is accomplished by interacting via the Redfish Interface.

5.1. Hardware Inventory

The Redfish client obtains the hardware inventory information for the rack and for each node.

The hardware inventory use case is supported by:
- The ability to obtain inventory information for the rack manager
- The ability to obtain inventory information for the nodes in the rack

5.1.1. Obtain inventory information for the rack manager

The hardware inventory for the rack in obtained from the Chassis resource representing the rack management hardware.

```
GET /redfish/v1/Chassis/RackManager
```

The response contains the following fragment, which includes hardware inventory properties for manufacturer, model, SKU, serial number and part number.   The AssetTag properties is a client writeable property.

```
{
      "@odata.type": "#Chassis.v1_2_0.Chassis",
      "@odata.id": "/redfish/v1/Chassis/RackManager",
      "Id": "RackManager",
      . . .
      "ChassisType": "Rack",
      "Name": "Rack Manager Hardware",
      "Manufacturer": "…"
      "Model": "RackScale_Rack",
      "SKU": "…"
      "SerialNumber": "…",
      "PartNumber": "…",
      "AssetTag": null,
}
```

5.1.2. Obtain inventory information for each node in the rack

The hardware inventory for the rack in obtained from the Chassis resource representing each node's hardware.

```
GET /redfish/v1/Chassis/{id}
```

The response contains the following fragment, which includes the hardware inventory properties for manufacturer, model, SKU, serial number and part number.   The AssetTag properties is a client writeable property.

```
{
      "@odata.type": "#Chassis.v1_2_0.Chassis",
      "@odata.id": "/redfish/v1/Chassis/Node1",
      "Id": "Node1",
      . . .
      "ChassisType": "Node",
      "Name": "Rack Manager Hardware",
      "Manufacturer": "…"
      "Model": "RackScale_Rack",
      "SKU": "…"
      "SerialNumber": "…",
      "PartNumber": "…",
      "AssetTag": null,
}
```

5.2. Rack Power Status

In the rack power status use case, the Redfish Client obtains the rack's power state and the power usage reading.

5.2.1. Obtain power state of the rack

The power state for the rack in obtained from the Chassis resource representing the rack hardware.

```
GET /redfish/v1/Chassis/Rack
```

The response contains the following fragment, which contains the PowerState property.

```
{
    "@odata.type": "#Chassis.v1_2_0.Chassis",
    "@odata.id": "/redfish/v1/Chassis/Rack",
    "Id": "Node1",
    . . .
    "ChassisType": "Rack",
    "PowerState": "On"
}
```

5.2.2. Obtain power usage for the rack

The power usage for the rack is obtained from the Power resource associated with the rack hardware.

```
GET /redfish/v1/Chassis/Rack/Power
```

The response contains the following fragment, which includes the PowerControl array properties. The PowerConsumedWatts property contains the value of instantaneous power usage. The PowerMetrics objects contains statistics (min, max, avg) power usage over a duration.

```
{
  "@odata.id": "/redfish/v1/Chassis/Rack/Power",
  "@odata.type": "#Power.v1_1_0.Power",
  "Id": "Power",
  "PowerControl": [ {
    "@odata.id": "/redfish/v1/Chassis/Zone1/Power#/PowerControl/0",
    "MemberId": "0",
    "Name": "System Power Control",
    "PowerConsumedWatts": 8000,
    "PowerMetrics": {
      "IntervalInMin": null,
      "MinConsumedWatts": null,
      "MaxConsumedWatts": null,
      "AverageConsumedWatts": null
    }
    }]
}
```

5.3. Rack Power Control

In the rack power control use case, the Redfish Client sets a power limit on the rack.

5.3.1. Set to limit for power usage for the rack

The power usage for the rack is modifying the PowerLimit object within the Power resource associated with the rack hardware.

The properties are writeable, so they can be PATCH'ed directly.

```
PATCH /redfish/v1/Chassis/Rack/Power
```

With the message

```
{
        "PowerLimit": {
            "LimitInWatts": 300
        }
}
```

Note that the PowerLimit complex properties has other properties that may be set during the same patch.

The LimitException property specifies the action if the power limit cannot be enforced.   The possible values are: "NoAction", "HardPowerOff", "LogEventOnly".

```
{
        "PowerLimit": {
            "LimitInWatts": 300,
            "LimitException": "LogEventOnly",
            "CorrectionInMs": 100
        }
}
```

5.4. PSU Status/Health

In the PSU Status/Health use case, the Redfish Client gets the health and status of the PSU (Power Supply Unit)

5.4.1. Obtain the status and health of the PSU

The status and health of the power supply unit is obtained from the Power resource associated with the rack hardware.

```
GET /redfish/v1/Chassis/Rack/Power
```

The response contains the following fragment. The status and health of the power supply is obtained from the PowerSupplies object within the Power resource associated with the rack hardware. Specifically, the Status object contains both State and Health properties.

```
{
    "@odata.id": "/redfish/v1/Chassis/Rack/Power",
    "@odata.type": "#Power.v1_1_0.Power",
    "Id": "Power",
    "PowerSupplies": [ {
        "@odata.id": "/redfish/v1/Chassis/Zone1/Power#/PowerSupplies/0",
        "MemberId": "0",
        "Name": "Power Supply Bay 1",
        "Status": {
            "State": "Enabled",
            "Health": "Warning"
        },
       . . .
        "RelatedItem": [ {
            "@odata.id": "/redfish/v1/Chassis/Rack"
        } ]
    } ]
}
```

5.5. Node Power Status

In the node power status use case, the Redfish Client obtains a node's power state and the power usage reading.

5.5.1. Obtain power state of a node

The power state for the node in obtained from the Chassis resource representing the node chassis or hardware.

```
GET /redfish/v1/Chassis/Node-1
```

The response contains the following fragment, which includes the PowerState properties.

```
{
       "@odata.id": "/redfish/v1/Chassis/Node-1,
       "ChassisType": "Node",
       "PowerState": "On"
}
```

5.5.2. Obtain power usage for a node

The power usage for a node is obtained from the Power resource associated with the node chassis or hardware.

```
GET /redfish/v1/Chassis/Node-1/Power
```

The response contains the following fragment. The PowerConsumedWatts property contains the value of instantaneous power usage.

```
{
       "@odata.id": "/redfish/v1/Chassis/Node-1/Power",
       "PowerControl": [
              {
                     "Name": "System Power Control",
                     "PowerConsumedWatts": 200
              }
       ]
       . . .
}
```

Note, the response also contains a PowerMetrics object.   The PowerMetrics object contains statistics regarding the power usage over a time interval (minimum, maximum, average).

```
{
        "@odata.id": "/redfish/v1/Chassis/Node-1/Power",
        "PowerControl": [
                {
                        "MemberId": "0",
                        "PowerMetrics": {
                                "IntervalInMin": 1,
                                "MinConsumedWatts": 197,
                                "MaxConsumedWatts": 202,
                                "AverageConsumedWatts": 199
                        }
                }
        ]
}
```

5.6. Node Power Control

The power usage limit for the node is modifying the PowerLimit object within the Power resource associated with the node's chassis or hardware.

The property is PATCH'ed directly.

```
PATCH /redfish/v1/Chassis/Node-1/Power
```

With the message

```
{
         "PowerLimit": {
             "LimitInWatts": 300
         }
}
```

The PATCH is similar to set the power limit on the rack, except the URI specifies the node's Power resource, instead of the rack's Power resource.

5.7. Node Temperature

The temperature of a node is obtained from the Thermal resource subordinate to Chassis resource which represents node's chassis.

```
GET /redfish/v1/Chassis/Node-1/Thermal
```

The response contains the following fragment. In the Temperatures array element whose "PhysicalContext" property has the value of "Intake", the ReadingCelsius property contains the value of temperature.

```
{
        "@odata.id": "/redfish/v1/Chassis/Node-1/Thermal",
        "Temperatures": [
                {
                        "ReadingCelsius": 21
                        "PhysicalContext": "Intake"
                }
        ]
}
```

Note: *In the same array element, properties exists which specify the threshold values and the range of the temperature readings.*

```
{
        "@odata.id": "/redfish/v1/Chassis/Node-1/Thermal",
        "Temperatures": [
                {
                        "PhysicalContext": "Intake"
                        "UpperThresholdNonCritical": 42,
                          "UpperThresholdCritical": 42,
                          "UpperThresholdFatal": 42,
                          "LowerThresholdNonCritical": 42,
                          "LowerThresholdCritical": 5,
                          "LowerThresholdFatal": 42,
                          "MinReadingRangeTemp": 0,
                          "MaxReadingRangeTemp": 200
                }
        ]
}
```

5.8. Node Health and Status

5.8.1. Obtain the status and health of the node

Redfish models a node as it physical chassis and the logical computer system.   The relationship between the two resource and specified by references. Figure shows how a diagram of the resource tree.

To determine the status and health the node chassis is obtained by retrieving the chassis resource which represent the chassis and hardware. or the node.

```
GET /redfish/v1/Chassis/Node-1
```

The response contains the following fragment.

```
{
        "@odata.id": "/redfish/v1/Chassis/Node-1",
        "Status": {
                "State": "Enabled",
                "Health": "OK"
        }
}
```

The status and health the node computer system aspect is obtained by retrieving the System resource representing the logical aspect of the

```
GET /redfish/v1/System/Node-1
```

The response contains the following fragment. The System's Status object contains an additional property, HealthRollup.

```
{
        "@odata.id": "/redfish/v1/System/Node-1",
        "Status": {
                "State": "Enabled",
                "Health": "OK",
                "HealthRollup": "OK"
        }
}
```

5.8.2. Status and health of the CPUs

The status and health the node CPUs is obtained by retrieving the System resource which represent the node.

```
GET /redfish/v1/System/Node-1
```

The response contains the following fragment\. The information of interest is contained in the Status object, which is contained by the ProcessSummary object.

```
{
        "@odata.id": "/redfish/v1/System/Node-1",
        "ProcessorSummary": {
                "Count": 8,
                "LogicalProcessorCount": 256,
                "Model": "Multi-Core Intel(R) Xeon(R) processor 7xxx Series",
                "Status": {
                        "State": "Enabled",
                        "Health": "OK",
                        "HealthRollup": "OK"
                },
}
```

More details and health of the individual processors can found by inspecting the individual processor resources in the Processors collection resource.

5.8.3. Status and health of the memory

The status and health the node's memory is obtained by retrieving the System resource which represent the node.

```
GET /redfish/v1/System/Node-1
```

The response contains the following fragment. The information of interest is contained in the Status object.

```
{
        "@odata.id": "/redfish/v1/System/Node-1",
        "MemorySummary": {
              "TotalSystemMemoryGiB": 16,
                "MemoryMirroring": "System",
                "Status": {
                        "State": "Enabled",
                        "Health": "OK",
                        "HealthRollup": "OK"
                }
        }
}
```

5.8.4. Obtain the state of the LED

The state of the LED is obtained by retrieving the Chassis resource which represent the node chassis.

```
GET /redfish/v1/Chassis/Node-1
```

The response contains the following fragment.   The information of interest is the value of the IndicatorLED property.

```
{
	"@odata.id": "/redfish/v1/Chassis/Node-1",
	"IndicatorLED": "Lit"
}
```

5.8.5. Retrieve the RMC log

The RMC log is by retrieving the Log resource, which represent the RMC's log.

```
GET /redfish/v1/Managers/RMC/LogService/Log
```

The response contains the following fragment.

```
{
	"@odata.id": "/redfish/v1/Managers/RMC/LogServices/Log",
	"Id": "Log1",
	"Name": "Rack Manager Log",
	"Description": "This log contains entries related to the operation of the BMC",
	"MaxNumberOfRecords": 100,
	"OverWritePolicy": "WrapsWhenFull",
	"DateTime": "2020-03-13T04:14:33+06:00",
	"DateTimeLocalOffset": "+06:00",
	"ServiceEnabled": true,
	"LogEntryType": "Event",
	"Status": {
		"State": "Enabled",
		"Health": "OK"
	},
	"Actions": {
		"#LogService.ClearLog": {
			"target": "/redfish/v1/Managers/RMC/LogServices/Log/Actions/LogService.ClearLog"
		}
	},
	"Entries": {
		"@odata.id": "/redfish/v1/Managers/RMC/LogServices/Log/Entries"
	}
}
```

5.8.6. Retrieve the System logs

The System's log are retrieved is obtained by retrieving the Log resource which represent the node's log.

```
GET /redfish/v1/Systems/Node-1/LogService/Log
```

The response contains the following fragment.

```
{
	"@odata.id": "/redfish/v1/Systems/Node-1/LogServices/Log",
	"Id": "Log",
	"Name": "System Log",
	"Description": "This log contains entries related to the operation of a system",
	"MaxNumberOfRecords": 1000,
	"OverWritePolicy": "WrapsWhenFull",
	"DateTime": "2015-03-13T04:14:33+06:00",
	"DateTimeLocalOffset": "+06:00",
	"ServiceEnabled": true,
        "LogEntryType": "Event",
        "Status": {
                "State": "Enabled",
                "Health": "OK"
        },
        "Actions": {
                "#LogService.ClearLog": {
                        "target": "/redfish/v1/Systems/Node-1/LogServices/Log/Actions/LogService.ClearLog"
                }
        },
        "Entries": {
                "@odata.id": "/redfish/v1/Systems/Node-1/LogServices/Log/Entries"
        }
}
```

5.9. Obtain the firmware revision

5.9.1. Obtain the revision of Rack Manager firmware

The version of firmware on the rack manager is obtained by retrieving the Manager resource which represents the rack manager.

```
GET /redfish/v1/Managers/RMC
```

The response contains the following fragment. The information of interest is the value of the FirmwareVersion property.

```
{
        "@odata.id": "/redfish/v1/Managers/RMC",
        "Id": "RMC",
        "FirmwareVersion": "1.00"
}
```

5.9.2. Obtain the revision of the BIOS firmware on each system

The version of BIOS firmware on a system is obtained by retrieving the System resource which represents the system.

```
GET /redfish/v1/Systems/{id}
```

The response contains the following fragment. The information of interest is the value of the BiosVersion property.

```
{
        "@odata.id": "/redfish/v1/System/CS_1",
        "Id": "CS_1",
        "BiosVersion": "P79 v1.00 (09/20/2013)"
}
```

5.9.3. Obtain the revision of the BMC firmware on each system

The version of firmware on the BMC on a system is obtained by retrieving the Manager resource which represents the BMC of interest.

```
GET /redfish/v1/Managers/BMC_1
```

The response contains the following fragment. The information of interest is the value of the FirmwareVersion property.

```
{
       "@odata.id": "/redfish/v1/Managers/BMC_1",
       "Id": "BMC_1",
       "FirmwareVersion": "1.00"
}
```

5.9.4. Obtain the revision of PSU firmware

The version of firmware on the PSU is obtained by retrieving the Power resource subordinate to the Chassis resource which represents the chassis of interest.

```
GET /redfish/v1/Chassis/Ch_1/Power
```

The response contains the following fragment. The information of interest is the value of the FirmwareVersion property.

```
{
      "@odata.id": "/redfish/v1/Chassis/Ch_1/Power",
      "Id": "Power",
      "PowerSupplies": {
            {
                  "@odata.id": "/redfish/v1/Chassis/Ch_1/Power#/PowerSupplies/0",
                  "MemberId": "0",
                  "FirmwareVersion": "1.00"
            }
      ]
}
```

5.10. Update the Rack Manager Firmware

The firmware on the rack manager can be updated with the pull or push method. These methods are distinguished by the method of transfer for the firmware to the rack manager. The firmware image can be pushed to the rack manager or the rack manager can be told to pull the firmware image on the system.

The Redfish service may start a task to handle the firmware update. The task is represented by a Task resource in the TaskService. When a task is started, the response is 202 (Accepted) with a TaskMonitorURI. The Redfish client then performs GETs on the TaskMonitorURI and inspects the value of the TaskState property.

```
GET /redfish/v1/TaskService/Tasks/FWU1
```

The response contains the following fragment.

```
{
       "@odata.id": "/redfish/v1/TaskService/Tasks/FWU1",
       "Id": "FWU1",
       "Name": "Task",
       "StartTime": "2020-05-05T14:44-05:00",
       "TaskState": "Completed",
       "TaskStatus": "OK",
       "Messages": [
              { . . . }
       ]
}
```

5.10.1. Pull Method

For a pull method, the Redfish client performs a SimpleUpdate action with a message which includes the location of the firmware image. The Redfish service transfers the firmware image from the specified location, before proceeding with the firmware update.

```
POST /redfish/v1/UpdateService/Actions/SimpleUpdate
```

The response contains the following fragment.

```
{
      "ImageURI": "...",
      "TransferProtocol": "HTTPS"
}
```

5.10.2. Push Method

For the push method, the Redfish client performs a POST with a message in a HTTP multi-part format. The format includes the firmware image.

The Redfish clients determines the URI to use for the POST, by using the value MultiipartHttpPushUri property in the UpdateService resource. In the example below, the property value is "/redfish/v1/UpdateService/upload".

```
POST /redfish/v1/UpdateService/upload HTTP/1.1
Host: <host-path>
Content-Type: multipart/form-data; boundary=---------------------------d74496d66958873e
Content-Length: <computed-length>
Connection: keep-alive
X-Auth-Token: <session-auth-token>

-----------------------------d74496d66958873e
Content-Disposition: form-data; name="UpdateParameters"
Content-Type: application/json

{
    "Targets": [ "/redfish/v1/Managers/1" ],
    "@Redfish.OperationApplyTime": "OnReset"
}

-----------------------------d74496d66958873e
Content-Disposition: form-data; name="UpdateFile"; filename="flash.bin"
Content-Type: application/octet-stream

<software image binary>
```

The multi-part format allows the UpdateFile to contain one or more images, each going to one or more targets.   In this case, the Redfish service is expected to understand how to interpret the UpdateFile which is included.

5.11. Account Management

The Redfish server has an account for each user that uses the Redfish interface.

```
POST /redfish/v1/AccountService/Accounts/1
```

The following is an example of an Account resource. The Redfish service has three mandatory resources in Roles resource collection: Administrator, Operator, ReadOnly.

```
{
       "@odata.id": "/redfish/v1/AccountService/Accounts/1",
       "Id": "1",
       "Name": "User Account",
       "Enabled": true,
       "Password": null,
       "PasswordChangeRequired": false,
       "UserName": "Administrator",
       "RoleId": "Administrator",
       "Locked": false,
       "Links": {
               "Role": {
                       "@odata.id": "/redfish/v1/AccountService/Roles/Administrator"
               }
       }
}
```

The following is the Role resource for the operator role.

```
{
        "@odata.id": "/redfish/v1/AccountService/Roles/Operator",
        "Id": "Operator",
        "Name": "User Role",
        "IsPredefined": true,
        "AssignedPrivileges": [
                "Login",
                "ConfigureSelf",
                "ConfigureComponents"
        ]
}
```

6. References

[1] "OpenRMC Design Specification"
http://www.opencompute.org/

[2] “Redfish API Specification”
https://www.dmtf.org/dsp/DSP0266

7. Revision

Revision	Date	Description
1.0.0	6/1/2021	Contributed version