Usage Guide for Rack Management v1.2

Date: 2025-03-20

Version: 1.2.0

Supersedes: 1.1.0

Document Class: Informational

Document Status: Draft

Document Language: en-US

CONTENTS

1 License

2 Scope

This document references requirements and provide the usage examples for the OpenRMC northbound API v1.2.0 for a rack management controller.

3 Requirements

As a Redfish-based interface, the required Redfish interface model elements are specified in a profile document. The profile is located at –

https://github.com/opencomputeproject/OCP-Profiles/blob/master/OCPRackManagerController.v1_2_0_WIP.json

The OCPRackManagerController.v1.2.0 profile extends from the OCPBaselineHardwareManagement.v1.0.1 profile. The profile is located at –

https://github.com/opencomputeproject/OCP-Profiles/blob/master/OCPBaselineHardwareManagement.v1_0_1.json

A profile is read by the Redfish Interop Validator which autogenerates and executes the tests against an implementation. The Interop Validator generates a test report – in text or HTML format.

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

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

4 Capabilities

The OpenRMC API extends from the OCP Baseline Hardware Management v1.0 capabilities. The following table lists those capabilities are extracted from The "Usage Guide and Requirements for the OCP Baseline Hardware Management Profile v1.0.1" document

Use Case	Management Task	Requirement
Account Management	Get accounts	Mandatory
Service Management	Get sessions	Mandatory
Hardware Inventory	Get FRU info	Mandatory
	Get and Set the Asset Tag	Recommended
Hardware Location	Get location LED	Recommended
	Set location LED	Recommended
Status	Get Chassis status	Mandatory
Power	Get power state	If implemented, mandatory
	Get power usage	Recommended
	Get power limit	Recommended
Temperature	Get temperature	If implemented, mandatory
	Get temperature thresholds	If implemented, recommended
Cooling	Get fan speeds	If implemented, mandatory
	Get fan redundancy	If implemented, recommended
Log	Get log entry	Mandatory
	Clear system log	Recommended
Management Controller	Get firmware version	Mandatory
	Get controller status	Mandatory
	Get network info	Mandatory
	Reset controller	Mandatory

- Baseline Capabilities

The following are the usages and capabilities of the OpenRMC interface which are incremental to the OCP Baseline Hardware Management capabilities.

Use Case	Management Task	Requirement
Rack Power Status	Get power state of rack	Mandatory
	Get power usage of rack	Recommended
Rack Power Control	Set power usage of reack	Mandatory
PSU State/Health	Get state of PSU	Mandatory
Node inventory	Get FRU Info of node	Mandatory
Node Power Status	Get power state of node	Mandatory
	Get Power usage of node	Mandatory
Node Power Control	Set Power usage of node	Mandatory
Node Temperature	Get temperature of node	Mandatory
Node Status/Health	Get status/health of node	Mandatory
	Get status of node CPUs	Mandatory
	Get status of node memory	Mandatory
	Get state of node LEDs	Mandatory
	Get logs from node	Mandatory
Firmware Update	Pull FW Update on Rack Manager	Mandatory
	Push FW Update on Rack Manager	Mandatory
	Pull FW Update on node	Mandatory
	Push FW Update on node	Mandatory
Group Operations	Reset a temp group of nodes	Mandatory
	Reset a persistent group of nodes	Mandatory
	Create a persistent group of nodes	Mandatory
	Set boot order of aggregate	Mandatory
Authorization	Get certificate from node	Mandatory
	Place certificate on node	Mandatory
	Place token on node	Mandatory

Use Case	Management Task	Requirement
	Place certificat on rack manager	Mandatory
	Place token on rack manager	Mandatory
	Place manifest on rack manager	Mandatory

4.1 Group Operations in Redfish

Group operations are performed using the AggregationService resource. Actions can be performed on temporary groups or persistent groups. A temporary group is passed within the action. A persistent group is create prior to action and referenced within the action. A persistent group is modelled as an Aggregate resource. The AggregateService resource contains the Aggregates collection resource whose members are Aggregate resources.

The following command

```
**GET** /redfish/v1/AggregationService
```

The response contains the following fragment. The fragment shows the URI for the Aggregate collection resource.

The fragment shows two actions that can be perform on an aggregate, either temporary or persistent. The action is invoke by performing a POST to the URI contained in Target property. The contents of the request body in desribed in resource whose URI is contained in the @Redfish.ActionInfo property.

```
    {
        "@odata.id": "/redfish/v1/AggregationService",
        "Id": "AggregationService",
        "ServiceEnabled": true,
        "Status": {
            "Health": "OK",
            "State: "Enabled"
        },
        "Aggregates": {
            "@odata.id": "/redfish/v1/AggregationService/Aggregates"
        },
        "Actions": {
            "#AggregationService.Reset": {
                "target": "/redfish/v1/AggregationService/Actions/AggregationService.Reset",
                "@Redfish.ActionInfo": "/redfish/v1/AggregationService/ResetActionInfo"
            },
```

```
            "#AggregationService.SetDefaultBootOrder": {
                "target": "/redfish/v1/AggregationService/Actions/
AggregationService.SetDefaultBootOrder",
                "@Redfish.ActionInfo": "/redfish/v1/AggregationService/SetDefaultBootOrderActionInfo"
            }
        }
    }
```

4.2 Update Firmware

The firmware can be updated with a pull or push method. The "Redfish Firmware Update Whitepaper"[3] has detail discussion of the firmware update process.

The main process is for the firmware package to be delivered opaquely, and the Redfish Service interprets the firmware package to determine the components that are updated. The Targets property can be used to guide and constrain this behavior.

5 Use Cases

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

5.1 Get FRU Info of node

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 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,
    }
```

5.1.1 Get 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 PowerState properties.

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

5.2 Get power usage for 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 Voltage 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 Set power usage of 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 Get status of 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 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 Get power state of 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 PowerState properties.

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

5.6 Get power usage of 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

Which responds with the following message. 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 PowerMetricsobject contains statistics regarding the power usage over a time interval (minimum, maximum, average).

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

5.7 Set power usage of Node

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. 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.

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

With the message

```
{
  "PowerLimit": {
    "Limit InWatts": 300
  }
}
```

5.8 Get temperature of Node

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 message is shown below. 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": \[ {
    "ReadingCelsius": 21
    "PhysicalContext": "Intake"
  } \]
}
```

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": \[ {
    "PhysicalContext": "Intake"
    "UpperThresholdNonCritical": 42,
    "UpperThresholdCritical": 42,
    "UpperThresholdFatal": 42,
    "LowerThresholdNonCritical": 42,
    "LowerThresholdCritical": 5,
    "LowerThresholdFatal": 42,
    "MinReadingRangeTemp": 0,
    "MaxReadingRangeTemp": 200
  } \]
}
```

5.9 Get status of 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

Which responds with the following message. The PowerConsumedWatts property contains the value of instantaneous power usage.


```
{
  "@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 following message is the response. The System's Status object contains an additional property, HealthRollup.


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


Which responds with the following message. The PowerConsumedWatts property contains the value of instantaneous power usage.

5.10 Get status of node 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 following message is the response. 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"
    }
  }
}
```


5.11 Get status of node 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 following message is the response. 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"
    }
  }
}
```


5.12 Get state of node 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 contain the following fragment. The information of interest is the value of the IndicatorLED property.

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

5.13 Get log from rack manager

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"
  }
}
```

5.14 Get log from node

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"
  }
}
```


5.15 Get FW version on rack manager

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"
}
```

5.16 Get FW version on node

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)"
}
```

5.17 Get BMC FW version on node

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"
}
```

5.18 Get FW version of PSU

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

5.19 Update Firmware on Rack Manager via Pull Method

To update the firmware on the rack manager via the pull method, the client invokes the following command.

POST /redfish/v1/UpdateService/Actions/SimpleUpdate

The POST command includes the following message. The value of the ImageURI property is the path to the new rack manager firmware image. The message may also include the TransferProtocol, Username and Password properties.

POST /redfish/v1/UpdateService/Actions/UpdateService.SimpleUpdate HTTP/1.1 Content-Type: application/json Content-Length:

```
{
  "ImageURI": "https://192.168.1.250/images/rmc_update.bin",
  "Target": \[
    "/redfish/v1/managers/RMC"
  \]
```

```
}
```

If the Redfish service starts a task to handle the firmware update, it will respond with a task pointer, TaskMonitorURI. The client monitors the task by performing GETs on the TaskMonitorURI and inspects the response.

5.20 Push FW Update on Rack Manager

To update the firmware on the rack manager via the push method, the client invokes the following command.

POST /redfish/v1/UpdateService/upload

The POST command includes the following multi-part message

```
Content-Type: multipart/form-data;
boundary=--------------------------d74496d66958873e
Content-Length:
----------------------------d74496d66958873e
Content-Disposition: form-data; name="UpdateParameters"
Content-Type: application/json
{
  "Target": \[
    "/redfish/v1/managers/RMC"
  \]
}
----------------------------d74496d66958873e
Content-Disposition: form-data; name="UpdateFile";
filename="bmc_update.bin"
Content-Type: application/octet-stream
\<software image binary\>
```

Redfish service starts a task to handle the firmware update, it will respond with a task pointer, TaskMonitorURI. The client monitors the task by performing GETs on the TaskMonitorURI and inspects the response.

5.21 Pull FW update on Node

The node firmware maybe updated with a pull or push method.

To update the firmware on a node, the process described above for the rack manager firmware can be used with minor changes. The primary change is the Target property, if it is used.

The Target property can specify the components, of interest.

```
{
  "Targets": \[
    "/redfish/v1/Systems/CS-3"
    "/redfish/v1/Managers/BMC_3"
  \]
}
```

The Target property can specify the node, of interest.

```
{
  "Targets": \[
    "/redfish/v1/systems/CS-3"
  \]
}
```

The Targets property can specify the nodes, of interest.

```
{
  "Target": \[
    "/redfish/v1/systems/CS-1",
    "/redfish/v1/systems/CS-3"
  \]
}
```

5.22 Reset a temporary group of nodes

To perform a reset of a temporary group, a HTTP POST is invoked. The resource URI to use for the POST is determined by inspecting the AggregateService resource. The resource URI is the 'target' property of the within the #Aggregate.Reset property.

To perform a reset of the group, a POST is invoked to the value of the Target property within the #Aggregate.Reset property.

POST /redfish/v1/AggregationService/Actions/Aggregate.Reset

The POST request shall contain a request body. The contents of the request body are described by resource specified by the @Reddfish.ActionInfo property. The TargetURIs property specifies the group to be used. After the group is used, it is forgotten.

```
{
  "BatchSize": 10,
  "DelayBetweenBatchesInSeconds": 15,
  "ResetType": "ForceRestart",
  "TargetURIs": \[
    "/redfish/v1/Systems/cluster-node3",
    "/redfish/v1/Systems/cluster-node4"
  \]
}
```

5.23 Reset a persistent group of nodes

To update a persistent set of nodes, the client invokes the following command.

POST /redfish/v1/AggregationService/Aggregates/Agg1/Actions/Aggregate.Reset

The POST command contains a request body. The ResetType property specifies what type of reset to perform and is mandatory. The BatchSize and DelayBetweenBatechesInSeconds specifies that the reset be done in batches, instead of all at the same time.

```
{
  "BatchSize": 10,
  "DelayBetweenBatchesInSeconds": 15,
  "ResetType": "ForceRestart"
}
```

5.24 Create a Persistent Set of Nodes

The previous usage model assumes that the aggregate, Agg1, already exists in the Aggregates collection.

To create an aggregate, the client invokes the following command.

POST /redfish/v1/AggregationService/Aggregates/Agg1

The response contains the following fragment. The Elements property contains the members of the group. The Actions property contains the actions that can be performed on the aggregate. An action is invoked by POST'ing to the URI value of the Target property with a request body containing the properties described in the ActionInfo resource.

```
{
  "@odata.id": "/redfish/v1/AggregationService/Aggregates/Agg1",
  "Id": "Agg1",
  "Name": "Aggregate One",
  "ElementsCount": 2,
  "Elements": \[
    { "@odata.id": "/redfish/v1/Systems/cluster-node3" },
    { "@odata.id": "/redfish/v1/Systems/cluster-node4" }
  \]
}
```

5.25 Set the Boot Order to their defaults a persistent group of nodes

To set the boot order of a persistent group of nodes to their default boot order, the client invokes the following command.

POST /redfish/v1/AggregationService/Aggregates/Agg1/Actions/Aggregate.SetDefaultBootOrder

The POST command has no request message.

5.26 Authorization between rack manager and manage node

The use cases specified below is the support the process for authorization between the rack manager and the managed node as described in section 6.

5.27 Get certificate from node

The certificate for a node is retrieved as member of the Certificates collection for the node.

GET /redfish/v1/Systems/Node-1/Certificates/Cert-1

The response contains the following fragment.

```
{
  "@odata.id": "/redfish/v1/Systems/Node-1/Certificates/Cert-1",
  "Id": "Cert-1",
  "Name": "HTTPS Certificate",
  "CertificateString": "-----BEGIN CERTIFICATE-----\n...\n-----END CERTIFICATE-----",
  "CertificateType": "PEM",
  "Issuer": {
```

```
    "Country": "US",
    "State": "Oregon",
    "City": "Portland",
    "Organization": "Contoso",
    "OrganizationalUnit": "ABC",
    "CommonName": "manager.contoso.org"
  },
  "Subject": {
    "Country": "US",
    "State": "Oregon",
    "City": "Portland",
    "Organization": "Contoso",
    "OrganizationalUnit": "ABC",
    "CommonName": "manager.contoso.org"
  },
  "ValidNotBefore": "2018-09-07T13:22:05Z",
  "ValidNotAfter": "2019-09-07T13:22:05Z",
  "KeyUsage": \[
    "ServerAuthentication"
  \]
}
```

5.28 Place certificate on node

The certificate is placed on a managed node with the following HTTP command.

POST /redfish/v1/Systems/{id}/Certificates/SystemID

The response contains the following fragment. The KeyUsage property shall have the value(s) ??.

```
{
  "@odata.id": "/redfish/v1/System/1/Certificates/SystemID",
  "@odata.type": "#Certificate.v1_1_0.Certificate",
  "Id": "1",
  "Name": "HTTPS Certificate",
  "CertificateString": "-----BEGIN CERTIFICATE-----\n...\n-----END ERTIFICATE-----",
  "CertificateType": "PEM",
  "Issuer": {
    "CommonName": "…" },
    "Organization": "…",
    "OrganizationalUnit": "…"
  },
  "Subject": {
    "CommonName": "…" ,
    "Organization": "…",
    "OrganizationalUnit": "…"
  },
```

```
  "ValidNotBefore": "2018-09-07T13:22:05Z",
  "ValidNotAfter": "2019-09-07T13:22:05Z",
  "KeyUsage": \[ "KeyCertSign" \]
}
```

5.29 Place token on node

The token is placed on a managed node with the following HTTP command.

POST /redfish/v1/Systems/{id}/Certificates/Token

The response contains the following fragment. The KeyUsage property shall have the value(s) ??.

```
{
  "@odata.id": "/redfish/v1/System/1/Certificates/Token",
  "@odata.type": "#Certificate.v1_1_0.Certificate",
  "Id": "Token",
  "Name": "HTTPS Certificate",
  "CertificateString": "-----BEGIN CERTIFICATE-----\n...\n-----END CERTIFICATE-----",
  "CertificateType": "PEM",
  "Issuer": {
    "CommonName": "…" ,
    "Organization": "…",
    "OrganizationalUnit": "…"
  },
  "Subject": {
    "CommonName": "…" },
    "Organization": "…",
    "OrganizationalUnit": "…"
  },
  "ValidNotBefore": "2018-09-07T13:22:05Z",
  "ValidNotAfter": "2019-09-07T13:22:05Z",
  "KeyUsage": \[ "KeyCertSign" \]
}
```

5.30 Place certificate on rack manager

The certificate is placed on the rack manager with the following HTTP command.

POST /redfish/v1/Managers/<RackManager>/Certificates/Certificate

Where <RackManager> is the member in which the "ManagerType" property has the value "RackManager".

The response contains the following fragment. The KeyUsage property shall have the value KeyCertSign.

```
{
  "@odata.id": "/redfish/v1/\<RackManager\>/1/Certificates/Token",
  "@odata.type": "#Certificate.v1_1_0.Certificate",
  "Id": "1",
  "Name": "HTTPS Certificate",
  "CertificateString": "-----BEGIN CERTIFICATE-----\n...\n-----END CERTIFICATE-----",
  "CertificateType": "PEM",
  "Issuer": {
    "CommonName": "…" ,
    "Organization": "…",
    "OrganizationalUnit": "…"
  },
  "Subject": {
    "CommonName": "…" },
    "Organization": "…",
    "OrganizationalUnit": "…"
  },
  "ValidNotBefore": "2018-09-07T13:22:05Z",
  "ValidNotAfter": "2019-09-07T13:22:05Z",
  "KeyUsage": \[ "KeyCertSign" \]
}
```

5.30.1 Place token on rack manager

The token is placed on the rack manager with the following HTTP command.

POST /redfish/v1/Managers/<RackManager>/Certificates/Token

Where <RackManager> is the member in which the "ManagerType" property has the value "RackManager".

The response contains the following fragment. The KeyUsage property shall have the value(s) ??.

```
{
  "@odata.id": "/redfish/v1/\<RackManager\>/1/Certificates/Token",
  "@odata.type": "#Certificate.v1_1_0.Certificate",
  "Id": "1",
  "Name": "HTTPS Certificate",
  "CertificateString": "-----BEGIN CERTIFICATE-----\n...\n-----END CERTIFICATE-----",
  "CertificateType": "PEM",
  "Issuer": {
    "CommonName": "…" ,
    "Organization": "…",
```

```
    "OrganizationalUnit": "…"
  },
  "Subject": {
    "CommonName": "…" },
    "Organization": "…",
    "OrganizationalUnit": "…"
  },
  "ValidNotBefore": "2018-09-07T13:22:05Z",
  "ValidNotAfter": "2019-09-07T13:22:05Z",
  "KeyUsage": \[ "KeyCertSign" \]
}
```

5.30.2 Place manifest on rack manager

The manifest is placed on the rack manager with the following HTTP command.

POST /redfish/v1/Managers/rmc/ManageabilityManifest

The request contains the following fragment.

```
{
  "@odata.id": "/redfish/v1/Managers/rmc/ManageabilityManifest",
  "@odata.type": "#ManageabilityManifest.v1_0_0.ManageabilityManifest",
  "Id": "ManageabilityManifest",
  "Name": "Manageability Manift\est",
  "NodesToManage": {
    {
      "NodeName": "node1",
      "NodeIDCertificateString": "-----BEGIN CERTIFICATE-----\n...\n-----END CERTIFICATE-----",
      "CertificateType": "PEM"
    },
    {
      "NodeName": "node2",
      "NodeIDCertificateString": "-----BEGIN CERTIFICATE-----\n...\n-----END CERTIFICATE-----",
      "CertificateType": "PEM"
    },
    {
      "NodeName": "switch1",
      "NodeIDCertificateString": "-----BEGIN CERTIFICATE-----\n...\n-----END CERTIFICATE-----",
      "CertificateType": "PEM"
    },
    {
      "NodeName": "nas1",
      "NodeIDCertificateString": "-----BEGIN CERTIFICATE-----\n...\n-----END CERTIFICATE-----",
      "CertificateType": "PEM"
    }
  }
```

```
}
```

6 Security

6.1 Security Model

The security model leverages the authentication, and a flow to ensure every entity (node, rack manager, CSP) has a certificate. Pushing/update certificate could follow the CSP or silicon vendor's existing flow to obtain/update the certificate. The model is RBAC security model, in which the CSP is the most privilege entity in the chain (generate token for both rack manager and nodes and distributes. CSP also sets up manifest and sends it to rack manager. Rack manager is the second in hierarchy who has CSPs' manifest and tokens of all managed nodes. The least privilege entity in the chain is node which only possess its own token.

6.1.1 The necessity for tokens and manifests

Token is the proof of ownership of a node or rack manager by a CSPs and shows which entities on rack belongs to which CSP. Manifest is the list of nodes from a particular CSP that a specific rack manager should manage. Tokens are always encrypted in transit that rogue rack manager could not manage nodes that are not designated to them.

6.1.2 Attestation

The authentication flow beyond the possession of a certificate as shown in discovery flow (6.6.1). In the discovery flow, both node and node manager will authenticate each other's token which is different than the certificate.

6.2 Process for authorization between rack manager and managed node

This section specifies the process by which the rack manager verifies that a managed node is one it is authorized to manage, and for a managed node to verify that a rack manager is authorized the manage it.

6.3 Definitions

- **Certificate** = X.509 Certificate (All entities, node, rack manager and CSP)
- **Node Certificate** = Certificate issued by a CA for a node

- **Rack Manager Certificate** = Certificate issued by a CA for a rack manager
- **Node Token** = CSP certificate + Node Certificate signed by a CSP.
  - The node token is created and distribute during initial setup by CSP and shows that the managed node belongs to a CSP.
- **Rack Manager Token** = CSP certificate + rack manager certificate signed by a CSP.
  - The rack manager token is created and distribute during initial setup by CSP and shows that the rack manager belongs to a CSP.
- **Manageability Manifest =** CSP certificate + All Managed Nodes Token.
  - The manifest lists the nodes on the rack that the rack manager can manage.
  - The manifest is signed b the CSP
  - Each managed node entry shall contain a node token.
- **Managed Node List =** Node Certificate + Time Stamp of Initial session establishment

> In the Redfish Certificate Management Whitepaper [1], the node certificate and rack manager certificate are referred to as device identity certificates.

6.4 Theory of Operations

The rack manager is provided a manageability manifest which includes a list of nodes that it can managed.

As the rack manager discovers a node, it obtains the node's certificate and verifies whether the manageability manifest contains the node. If so, it proceeds to proof to the node, that it has authority to manage the node and challenges to node.

6.5 Procedure

6.5.1 Initial conditions

- Initial conditions of node
  - Each managed node shall have a node certificate
  - Each managed node shall have a node token.
- Initial conditions of rack manager

- The rack manager shall have a rack manager certificate
- The rack manager shall have a rack manager token
- The rack manager shall have a manageability manifest

6.5.2 Node Discovery

The following procedure is followed when the rack manager first discovers the presence of a node.

The rack manager discovers the presence of the node on its rack and asks for nodes' certificate. During discovery process, the rack manager knows nothing about the managed nodes. It could broadcast a message or ping to get node's certificate.

The rack manager shall attempt to locate the entry for the node in manifest.

If the entry is found, the rack manager shall send part of this manifest and random generated challenge ([rack manager token + node token + challenge] encrypted by node's public key) to the node.

If the entry is not found, the rack manager shall remain silent

6.5.3 Node Authentication

The managed node shall decrypt the message with its own private key. It verifies the signature of its own token and rack manager token. This authentication indicates that the node manager is the legitimated rack manager assigned by CSP for this node.

If the authentication is successful, the managed node shall send the random generated challenge encrypted by rack manager's public key and nuance/time to the rack manager.

If the authentication is unsuccessful, the managed node shall remain silent

The rack manager shall decrypt and verify the challenge and check the freshness of nuance/time. This indicates that the managed node is the node that the rack manager has been configured to manage.

Both rack manager and managed node shall generate session keys based on the challenge and nuance/time to establish a connection.

Rack manager shall retain the node certificate and time stamp at which the connection was established.

6.5.4 Certificate Revocation Management

From Rack manager's perspective, CSP would be in charge of revocation of a node, and they would

update the manifest and pass it to Rack manager to inform it about the revocation status. Rack manager follows Update/Revoke flow in 6.6.2 to terminate session for any revoked nodes.

6.5.5

6.6 Flows

6.6.1 Managed Node starts up (Discovery Flow)

A picture containing graphical user interface Description automatically generated

6.6.2 Manageability Manifest Updated

Graphical user interface, application Description automatically generated

6.7 Threat and Risk Model

6.7.1 Assets

Node's Token = CSP cert + Node's cert signed by CSP Rack Manager's Token = CSP cert + Rack Manager cert signed by CSP Manifest = CSP cert + All managed node’s tokens

Number	Asset	Pri/Sec	Risk
1	Node's Token & Rack Manager Token	Primary	High
2	CSP's Manifest	Primary	High
3	End user's data or rack resource	Secondary	High

Reason for Classification

1. This token shows that a node belong to a certain CSP. It is a confidential info to preserve CSP’s inventory privacy and to mitigate malicious attacks (more in thread model). Token are always encrypted in transit.
2. Manifest contains all the nodes that should be managed by a rack manager. All tokens are encrypted by rack manager’s public key to preserve confidentiality and privacy of tokens and mitigate malicious attacks (more in thread model).
3. In transit and storage

6.7.2 Adversaries

#	Persona	Motivation	Attacker Type	Starting Privilege Level	Skill and Potential Effor Level
1	Rogue Node	Intentional data exfiltration and access to resources	Network Adversary	User-level privilege	Highly skilled
2	Rogue Rack Manager	Intentional data exfiltration and managing nodes	Network Adversary	High privilege level	Highly skilled
3	Malicious Attacker	Intentional data exfiltration and managing nodes	Network Adversary	User-level privilege	Highly skilled, dedicated

#	Persona	Motivation	Attacker Type	Starting Privilege Level	Skill and Potential Effort level
1	Rogue Node	Intentional data exfiltration and access to resources	Network Adversary	User-level privileges	Highly skilled
2	Rogue Rack Manager	Intentional data exfiltration and managing nodes	Network Adversary	High privilege level	Highly skilled
3	Malicious Attacker	Intentional data exfiltration and managing nodes	Network Adversary	User-level privilege	Highly skilled, dedicated

6.7.3 Threats

#	Threat	Adversary	Asset	Rank	Mitigation
1	Attacker intercepts network traffic to gain access to CSP’s manifest to add rogue node/rack manager or to discover CSP’s nodes list for targeted attack	Network Adversary	Manifest	H	CSP’s manifest has list of nodes’ token encrypted by designated rack manager’s public key
2	Attacker set up a rogue node/rack manager to access specific CSP’s nodes	Network Adversary	End User’s Data	H	Nodes and rack managers for a particular CSP have a token (CSP’s certificate + Node/Rack Manager certificate signed by CSP). Rogue node’s and rogue rack manager’s attempt fails during discover phase without valid token.
3	Attacker sniffs the traffic during CSP’s initial token distribution to access legitimate node/rack manager token to set up a rogue node/rack manager with reply attack	Network Adversary	Node/ Rack Manager Token	H	Node/rack manager’s token are always encrypted during transition with node/rack manager’s public key.
4	Rogue node attempts to get added to a CSP’s rack manager	Network Adversary	End User’s Data	H	During discovery phase, rack manager checks if node’s certificate is part of CSP’s manifest. If not, it logs a “Unrecognized Node on Rack” event in its log (no session will establish).
5	Rogue node attempts to reply a legitimate node certificate	Network Adversary	End User’s Data	H	During discovery phase, rack manager will send a challenge encrypted by legitimate node’s public key. Node should resend back the challenge encrypted by rack manager’s public key. If rack manager could not verify the exact challenge sent, it records a “Suspicious Node on Rack” event in its log (no session will get established)
6	Revoked node continues to use resource of the rack	Network Adversary	Rack Resources	H	After revocation, CSP sends an updated manifest to rack manager. rack manager terminates the session of revoked nodes during updated manifest flow. It records “Revoked Node” event in its log.
7	Rogue rack manager attempts to manage CSP’s nodes	Network Adversary	End User’s Data	H	In discovery flow, rack manager send its token to managed node. Node verifies rack manager’s token signature to make sure rack manager belongs to the same CSPs as itself . If it does not, it logs “Suspicious Rack manager” in its log. (no session will get established).

#	Threat	Adversary	Asset	Rank	Mitigation
8	Non designated rack manager (belong to the same CSP as the node) attempts to manage nodes	Network Adversary	End User's Data	H	In discovery flow, rack manager sends node's token from its manifest as a proof that this rack manager is the designated rack manager for that node. Node verifies the node token to make sure rack manager is its designated rack manager.
					Other node managers do not have node's token (encrypted during transit)
9	A rogue rack manager attempts to reply legitimate encrypted rack manager's token	Network Adversary	End User's Data	H	In discovery flow, node decrypts rack manager's token and checks the cert in token with the cert that rogue rack manager passes down early in discovery flow. Discrepancy leads to the fact that node logs "Suspicious Node Manger" event (no session will get establish).

7 References

[1] "OpenRMC Design Specification"

[2] Usage Guide and Requirements for the OCP Baseline Hardware Management Profile v1.0.1

[3] "Redfish Firmware Update White Paper"

[4] "Redfish API Specification"

[5] "Redfish Certificate Whitepaper"

8 Revision

Revision/Version	Date	Description
1.0.0	6/14/2021	Released June 2021
1.1.0	5/26/2023	Released
1.2	3/20/2025	Draft