Document Identifier: DSP2060

Date: 2024-08-01

Version: 1.1.0

Redfish User Guide

Supersedes: 1.0.0

Document Class: Informational

Document Status: Published

Document Language: en-US

CONTENTS

Foreword

The Redfish User Guide was prepared by DMTF's Redfish Forum.

DMTF is a not-for-profit association of industry members dedicated to promoting enterprise and systems management and interoperability. For information about DMTF, see http://www.dmtf.org.

These sites provide more information about the Redfish standard:

Site	Description
DMTF GitHub repositories	Open-source tools and libraries for Redfish.
DMTF Redfish Forum	Organization that maintains the Redfish standard. This site provides information about member companies, future work and schedules, charter, and information about joining the working group that maintains the Redfish standard.
Redfish Developer Hub	Resources, including an interactive schema explorer, hosted schema, and other links, for developers who use Redfish to build applications.
Redfish Specification Forum	DMTF Redfish-monitored user forum that answers questions about Redfish-related topics.
Redfish standard	Redfish schemas, specifications, mockups, white papers, FAQ, educational material, and more.

Acknowledgments

DMTF acknowledges the following individuals for their contributions to this document:

- Jeff Autor — Hewlett Packard Enterprise
- Michael Raineri — Dell Technologies

1 Introduction

Redfish defines an easy-to-use and implement RESTful interface that lets users manage a wide range of devices and environments including stand-alone servers, composable infrastructures, and large-scale cloud environments.

Redfish defines a JSON-encoded data model. Because it requires JSON representation, Redfish is easy to both interpret and integrate with programming environments. To support various schema-aware tools, the Redfish data model is schema-based and is published in the following formats:

- OpenAPI YAML
- JSON Schema
- OData CSDL

Although the Redfish data model is schema-based, it's not necessary for new users to understand schema in order to use Redfish. This guide will show users how to navigate the Redfish model and perform common management operations.

2 Notes about examples in this guide

The examples in this guide assume that a Redfish-enabled system is available. Many of these examples use curl, a common Linux utility that is also available for Windows, with specific options to send Redfish requests from the command line. These examples also pipe the curl output into the Python json.tool to validate and pretty-print JSON responses. Postman is an alternative tool for performing the same operations. The Redfish Mockup Server is available for development purposes when a Redfish-enabled system is not available.

The examples in this guide use these parameters for many operations:

- `<REDFISH-HOST>` : Server name or IP address of the Redfish-enabled system.
- `<USERNAME>` : Username for the Redfish account.
- `<PASSWORD>` : Password for the Redfish account.
- `<SESSION-ID>` : ID of your session.
- `<SESSION-TOKEN>` : Session token that you use on all subsequent requests including `DELETE` .

3 Redfish resouce map

A Redfish service, or simply *service*, is a software or firmware product that implements the protocols, resources, and functions of the *Redfish Specification*. In many cases, a baseboard management controller (BMC) implements Redfish to provide remote management capabilities of a system.

A service implements resources in the Redfish data model at different URIs. All services support an entry point called the service root. The service root contains references to the supported top-level resource collections and services supported by the implementation. The service root is always located at the URI `/redfish/v1/`.

Hyperlinks in responses from the service are used to guide clients to other resources in the service. These hyperlinks are represented as `@odata.id` properties where the value of the property is the URI of the referenced resource. There are many examples of these hyperlinks in this guide, such as the Redfish service root section that shows a sample service root with hyperlinks to the top-level resource collections and services.

Figure 1 shows a simplified resource map for managing a server.

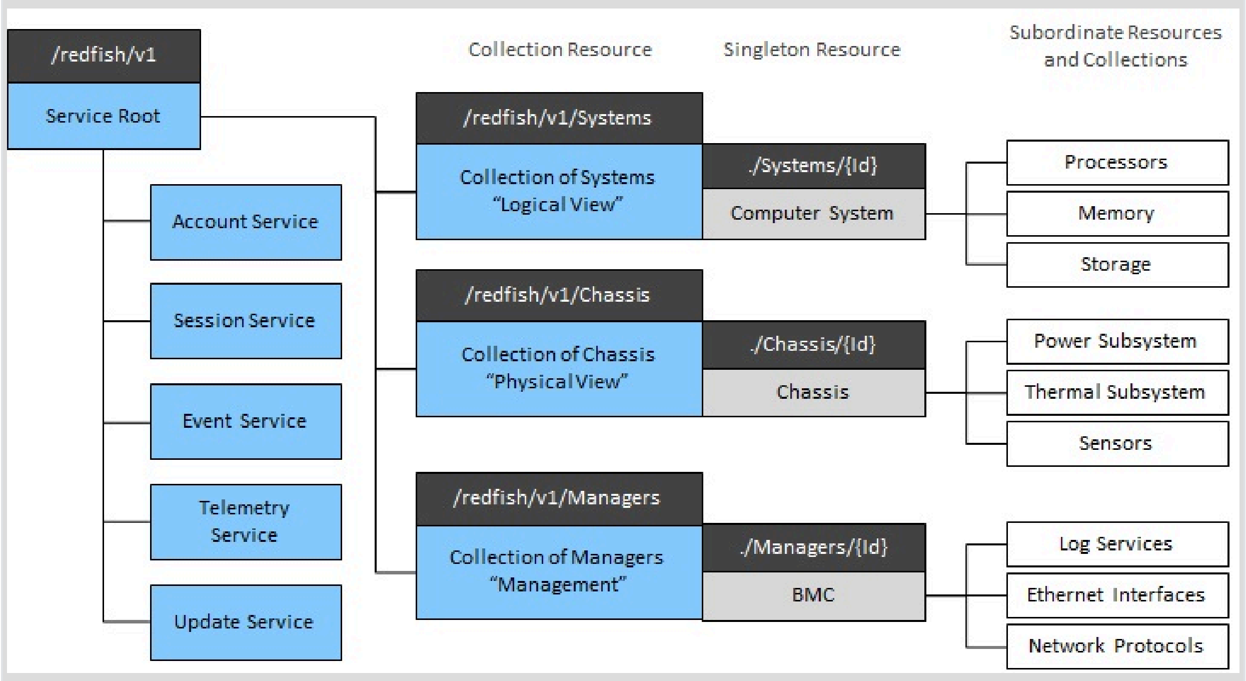

Figure 1

In the previous diagram, there are three commonly used sets of resources for managing a server:

- `Chassis`: Represents the physical view of a container of equipment, which can be a card, a server blade, an enclosure in a rack, an entire rack, or other types of containers.
- `Managers`: Represents BMCs, enclosure managers, or any other components that manage the infrastructure.
- `Systems`: Represents the logical view of any subsystem accessible from the host CPU.

The Chassis example, Computer system example, and Manager example sections show how users can perform management operations on the previous resources.

4 Authentication

4.1 Authentication overview

To authenticate access to Redfish resources, user credentials are provided to the Redfish service through either HTTP Basic authentication or sessions. When attempting to access a resource without valid credentials or with insufficient privileges, the service returns the HTTP `401 Unauthorized` or `403 Forbidden` status code.

4.2 HTTP Basic authentication

With HTTP Basic authentication, credentials are provided as a Base64-encoded string of the username and password as `<USERNAME>:<PASSWORD>`. This string is provided in the `Authorization` request header with the `Basic` scheme:

`Authorization: Basic <Base64-encoded credentials>`

For example, if the username is `admin` and the password is `password`, the following header would be used:

`Authorization: Basic YWRtaW46cGFzc3dvcmQ=`

When using HTTP Basic authentication, users will need to provide this header in each request to the service. This means that the service will need to verify the username and password on every request. This can have a negative impact on timing and latency, so this method is not optimal for most circumstances when multiple requests will be issued to the service.

4.3 Sessions

Redfish sessions allow a user to exchange a username and password for a session token, and the session token is used in susbsequent requests. This is a more optimal approach when performing a multiple operations so that the username and password only need to be verified by the service on the initial session creation request.

To create a session, perform a `POST` operation on the `/redfish/v1/SessionService/Sessions` URI:

```
curl -k -D - -X POST 'https://<REDFISH-HOST>/redfish/v1/SessionService/Sessions' \
-H "Content-Type: application/json" -d '{ "UserName": "<USERNAME>", "Password": "<PASSWORD>" }'
```

The response from the service, if successful, will contain the session ID and session token in the `Location` and `X-`

`Auth-Token` response headers respectively. These values need to be saved for future operations with the session. The following is an example response where a session was successfully created:

```
HTTP/1.1 201 Created
Location: /redfish/v1/SessionService/Sessions/<SESSION-ID>
X-Auth-Token: <SESSION-TOKEN>
Content-Type: application/json

{
    "@odata.type": "#Session.v1_1_1.Session",
    "@odata.id": "/redfish/v1/SessionService/Sessions/<SESSION-ID>",
    "Id": "1",
    "Name": "User Session",
    "UserName": "<USERNAME>",
    "Password": null
}
```

In subsequent requests to the service, provide the `X-Auth-Token` request header with the session token received in the response from the session creation. The following example shows a request to the URI `/redfish/v1/Chassis/1U` with the `X-Auth-Token` header provided:

```
curl -k 'https://<REDFISH-HOST>/redfish/v1/Chassis/1U' -H 'X-Auth-Token: <SESSION-TOKEN>'
```

When the session is no longer needed, perform a `DELETE` operation on the session URI that represents the session:

```
curl -k -X DELETE 'https://<REDFISH-HOST>/redfish/v1/SessionService/Sessions/<SESSION-ID>' \
-H 'X-Auth-Token: <SESSION-TOKEN>'
```

If successful, the session token is invalidated. The service will reject future requests that contain the invalidated token.

5 Redfish service root

To get the Redfish service root, perform a `GET` operation on the Redfish service root URI. Authentication is not needed because this resource is not protected.

```
curl -k 'https://<REDFISH-HOST>/redfish/v1/' | python -m json.tool
```

The following example shows a sample Redfish service root:

```
{
    "@odata.id": "/redfish/v1/",
    "@odata.type": "#ServiceRoot.v1_14_0.ServiceRoot",
    "Id": "RootService",
    "Name": "Root Service",
    "RedfishVersion": "1.15.0",
    "UUID": "92384634-2938-2342-8820-489239905423",
    "ProtocolFeaturesSupported": {
        "ExpandQuery": {
            "ExpandAll": true,
            "Levels": true,
            "MaxLevels": 6,
            "Links": true,
            "NoLinks": true
        },
        "SelectQuery": false,
        "FilterQuery": false,
        "OnlyMemberQuery": true,
        "ExcerptQuery": true
    },
    "Systems": {
        "@odata.id": "/redfish/v1/Systems"
    },
    "Chassis": {
        "@odata.id": "/redfish/v1/Chassis"
    },
    "Managers": {
        "@odata.id": "/redfish/v1/Managers"
    },
    "Tasks": {
        "@odata.id": "/redfish/v1/TaskService"
    },
    "SessionService": {
        "@odata.id": "/redfish/v1/SessionService"
    },
    "AccountService": {
        "@odata.id": "/redfish/v1/AccountService"
    },
    "EventService": {
        "@odata.id": "/redfish/v1/EventService"
    },
    "Registries": {
        "@odata.id": "/redfish/v1/Registries"
    },
    "UpdateService": {
        "@odata.id": "/redfish/v1/UpdateService"
    },
    "CertificateService": {
        "@odata.id": "/redfish/v1/CertificateService"
    },
    "Links": {
        "Sessions": {
            "@odata.id": "/redfish/v1/SessionService/Sessions"
        }
    }
}
```

The service root includes hyperlinks to the top-level resource collections and services supported by the implementation. For example, the `AccountService` property contains a hyperlink to the account service, shown by the value `/redfish/v1/AccountService` for its `@odata.id` property.

The following table lists the top-level resource collections and services that are commonly found in Redfish services. This table is not exhaustive for every possible property in the service root.

Property	Description
`AccountService`	Service to manage user accounts.
`CertificateService`	Service to manage certificates installed on the service.
`Chassis`	Collection of physical containers managed by the service.
`EventService`	Service to manage asynchronous event notifications.
`JobService`	Service to manage operations that are scheduled to run at particular points in time.
`Managers`	Collection of management entities managed by the service.
`SessionService`	Service to manage sessions.
`Systems`	Collection of systems managed by the service.
`Tasks`	Service to manage outstanding operations with the service.

Property	Description
`TelemetryService`	Service to manage collection and reporting of metrics tracked by the service.
`UpdateService`	Service to perform firmware or software updates.

6 Resource collections

A common pattern found in Redfish is the usage of resource collections. These types of resources act as containers for a set of resources of the same type. Depending on the specific resource collection and capabilities of the service, resource collections can grow or shrink over time.

Resource collections contain a property called `Members` to show the hyperlinks to the members of the collection. Resource collections can also contain a property called `Members@odata.nextLink` for cases where a response cannot show hyperlinks to all of the members of the collection. In these cases, users will need to follow this hyperlink to find additional members of the collection.

The `ChassisCollection`, `ManagerCollection`, and `ComputerSystemCollection` resources are example resource collections commonly found in the service root. The following example shows a request to the `ChassisCollection` resource found in the service root at the URI `/redfish/v1/Chassis`.

```
curl -k 'https://<REDFISH-HOST>/redfish/v1/Chassis' -H 'X-Auth-Token: <SESSION-TOKEN>' \
| python -m json.tool
```

The response to the request shows the resource collection contains one member with the URI `/redfish/v1/Chassis/1U`:

```
{
    "@odata.id": "/redfish/v1/Chassis",
    "@odata.type": "#ChassisCollection.ChassisCollection",
    "Members": [
        {
            "@odata.id": "/redfish/v1/Chassis/1U"
        }
    ],
    "Members@odata.count": 1,
    "Name": "Chassis Collection"
}
```

If supported by the service, users can perform `POST` operations on a resource collection to create new members for that collection. A `DELETE` operation on a member of the resource collection will remove the member from the collection. Authentication with sessions is an example of this, where the `POST` operation on `/redfish/v1/SessionService/Sessions` will add a new member to the `SessionCollection` resource, and the `DELETE` operation on `/redfish/v1/SessionService/Sessions/<SESSION-ID>` removes the member from the `SessionCollection` resource.

7 Chassis example

To get a `Chassis` resource, perform a `GET` operation on the URI of the desired member of the `ChassisCollection` resource. The following example shows a request to retreive the `Chassis` resource named `1U`.

```
curl -k 'https://<REDFISH-HOST>/redfish/v1/Chassis/1U' -H 'X-Auth-Token: <SESSION-TOKEN>' \
| python -m json.tool
```

The following example shows a response to the previous request:

```
{
    "@odata.id": "/redfish/v1/Chassis/1U",
    "@odata.type": "#Chassis.v1_21_0.Chassis",
    "Id": "1U",
    "Name": "Computer System Chassis",
    "ChassisType": "RackMount",
    "AssetTag": "Chicago-45Z-2381",
    "Manufacturer": "Contoso",
    "Model": "3500RX",
    "SKU": "8675309",
    "SerialNumber": "437XR1138R2",
    "PartNumber": "224071-J23",
    "PowerState": "On",
    "LocationIndicatorActive": true,
    "HeightMm": 44.45,
    "WidthMm": 431.8,
    "DepthMm": 711,
    "WeightKg": 15.31,
    "Location": {
        "PostalAddress": {
            "Country": "US",
            "Territory": "OR",
            "City": "Portland",
            "Street": "1001 SW 5th Avenue",
            "HouseNumber": 1100,
            "Name": "DMTF",
            "PostalCode": "97204"
        },
        "Placement": {
            "Row": "North",
            "Rack": "WEB43",
            "RackOffsetUnits": "EIA_310",
            "RackOffset": 12
        }
    },
    "Status": {
        "State": "Enabled",
        "Health": "OK"
    },
    "ThermalSubsystem": {
        "@odata.id": "/redfish/v1/Chassis/1U/ThermalSubsystem"
    },
    "PowerSubsystem": {
        "@odata.id": "/redfish/v1/Chassis/1U/PowerSubsystem"
    },
    "EnvironmentMetrics": {
        "@odata.id": "/redfish/v1/Chassis/1U/EnvironmentMetrics"
    },
    "Sensors": {
        "@odata.id": "/redfish/v1/Chassis/1U/Sensors"
    },
    "Links": {
        "ComputerSystems": [
            {
                "@odata.id": "/redfish/v1/Systems/437XR1138R2"
            }
        ],
        "ManagedBy": [
            {
                "@odata.id": "/redfish/v1/Managers/BMC"
            }
        ],
        "ManagersInChassis": [
            {
                "@odata.id": "/redfish/v1/Managers/BMC"
            }
        ]
    }
}
```

The response contains information about the physical container the `Chassis` resource represents. It's common to find information such as status, health, and FRU information about the container. Power subsystem, thermal subsystem, and sensor information can be found in hyperlinks from the `Chassis` resource. The response can also contain hyperlinks to equipment within the container.

The Chassis section of the *Redfish Resource and Schema Guide* contains details for the properties that can be found in the `Chassis` resource.

8 Computer system example

To get a `ComputerSystem` resource, perform a `GET` operation on the URI of the desired member of the `ComputerSystemCollection` resource. The following example shows a request to retreive the `ComputerSystem` resource named `437XR1138R2`.

```
curl -k 'https://<REDFISH-HOST>/redfish/v1/Systems/437XR1138R2' \
-H 'X-Auth-Token: <SESSION-TOKEN>' | python -m json.tool
```

The following example shows a response to the previous request:

```
{
    "@odata.id": "/redfish/v1/Systems/437XR1138R2",
    "@odata.type": "#ComputerSystem.v1_18_0.ComputerSystem",
    "Id": "437XR1138R2",
    "Name": "WebFrontEnd483",
    "SystemType": "Physical",
    "AssetTag": "Chicago-45Z-2381",
    "Manufacturer": "Contoso",
    "Model": "3500",
    "SubModel": "RX",
    "SKU": "8675309",
    "SerialNumber": "437XR1138R2",
    "PartNumber": "224071-J23",
    "Description": "Web Front End node",
    "UUID": "38947555-7742-3448-3784-823347823834",
    "HostName": "web483",
    "Status": {
        "State": "Enabled",
        "Health": "OK",
        "HealthRollup": "OK"
    },
    "HostingRoles": [
        "ApplicationServer"
    ],
    "IndicatorLED": "Off",
    "PowerState": "On",
    "Boot": {
        "BootSourceOverrideEnabled": "Once",
        "BootSourceOverrideTarget": "Pxe",
        "BootSourceOverrideTarget@Redfish.AllowableValues": [
            "None",
            "Pxe",
            "Cd",
            "Usb",
            "Hdd",
            "BiosSetup",
            "Utilities",
            "Diags",
            "SDCard",
            "UefiTarget"
        ],
        "BootSourceOverrideMode": "UEFI",
        "UefiTargetBootSourceOverride": "/0x31/0x33/0x01/0x01"
    },
    "TrustedModules": [
        {
            "FirmwareVersion": "1.13b",
            "InterfaceType": "TPM1_2",
            "Status": {
                "State": "Enabled",
                "Health": "OK"
            }
        }
    ],
    "BootProgress": {
        "LastState": "OSRunning",
        "LastStateTime": "2021-03-13T04:14:13+06:00",
        "LastBootTimeSeconds": 676
    },
    "LastResetTime": "2021-03-13T04:02:57+06:00",
    "BiosVersion": "P79 v1.45 (12/06/2017)",
    "ProcessorSummary": {
        "Count": 2,
        "Model": "Multi-Core Intel(R) Xeon(R) processor 7xxx Series",
        "LogicalProcessorCount": 16,
        "CoreCount": 8,
        "Status": {
            "State": "Enabled",
            "Health": "OK",
            "HealthRollup": "OK"
        }
    },
    "MemorySummary": {
        "TotalSystemMemoryGiB": 96,
        "TotalSystemPersistentMemoryGiB": 0,
        "MemoryMirroring": "None",
        "Status": {
            "State": "Enabled",
            "Health": "OK",
            "HealthRollup": "OK"
        }
    },
    "Bios": {
        "@odata.id": "/redfish/v1/Systems/437XR1138R2/Bios"
    },
    "SecureBoot": {
        "@odata.id": "/redfish/v1/Systems/437XR1138R2/SecureBoot"
    },
    "Processors": {
        "@odata.id": "/redfish/v1/Systems/437XR1138R2/Processors"
    },
    "Memory": {
        "@odata.id": "/redfish/v1/Systems/437XR1138R2/Memory"
    },
    "EthernetInterfaces": {
        "@odata.id": "/redfish/v1/Systems/437XR1138R2/EthernetInterfaces"
    },
    "SimpleStorage": {
        "@odata.id": "/redfish/v1/Systems/437XR1138R2/SimpleStorage"
    },
    "LogServices": {
        "@odata.id": "/redfish/v1/Systems/437XR1138R2/LogServices"
    },
    "GraphicsControllers": {
        "@odata.id": "/redfish/v1/Systems/437XR1138R2/GraphicsControllers"
    },
    "USBControllers": {
        "@odata.id": "/redfish/v1/Systems/437XR1138R2/USBControllers"
    },
    "Certificates": {
        "@odata.id": "/redfish/v1/Systems/437XR1138R2/Certificates"
    },
    "VirtualMedia": {
        "@odata.id": "/redfish/v1/Systems/437XR1138R2/VirtualMedia"
    },
    "Links": {
        "Chassis": [
            {
                "@odata.id": "/redfish/v1/Chassis/1U"
            }
        ],
        "ManagedBy": [
            {
                "@odata.id": "/redfish/v1/Managers/BMC"
            }
        ]
    },
    "Actions": {
        "#ComputerSystem.Reset": {
            "target": "/redfish/v1/Systems/437XR1138R2/Actions/ComputerSystem.Reset",
            "ResetType@Redfish.AllowableValues": [
                "On",
                "ForceOff",
                "GracefulShutdown",
                "GracefulRestart",
                "ForceRestart",
                "Nmi",
                "ForceOn",
                "PushPowerButton"
            ]
        }
    }
}
```

The response contains information about the system the `ComputerSystem` resource represents. It's common to find information such as status, health, and FRU information about the system. Processor, memory, storage, and other components that comprise the system can be found in hyperlinks from the `ComputerSystem` resource. The response can also contain hyperlinks to more detailed configuration settings for the system, such as BIOS and UEFI Secure Boot settings.

The ComputerSystem section of the *Redfish Resource and Schema Guide* contains details for the properties that can be found in the `ComputerSystem` resource.

9 Manager example

To get a `Manager` resource, perform a `GET` operation on the URI of the desired member of the `ManagerCollection` resource. The following example shows a request to retreive the `Manager` resource named `BMC`.

```
curl -k 'https://<REDFISH-HOST>/redfish/v1/Managers/BMC' \
-H 'X-Auth-Token: <SESSION-TOKEN>' | python -m json.tool
```

The following example shows a response to the previous request:

```
{
    "@odata.id": "/redfish/v1/Managers/BMC",
    "@odata.type": "#Manager.v1_16_0.Manager",
    "Id": "BMC",
    "Name": "Manager",
    "ManagerType": "BMC",
    "Description": "Contoso BMC",
    "ServiceEntryPointUUID": "92384634-2938-2342-8820-489239905423",
    "UUID": "58893887-8974-2487-2389-841168418919",
    "Model": "Joo Janta 200",
    "DateTime": "2015-03-13T04:14:33+06:00",
    "DateTimeLocalOffset": "+06:00",
    "Status": {
        "State": "Enabled",
        "Health": "OK"
    },
    "PowerState": "On",
    "GraphicalConsole": {
        "ServiceEnabled": true,
        "MaxConcurrentSessions": 2,
        "ConnectTypesSupported": [
            "KVMIP"
        ]
    },
    "CommandShell": {
        "ServiceEnabled": true,
        "MaxConcurrentSessions": 4,
        "ConnectTypesSupported": [
            "Telnet",
            "SSH"
        ]
    },
    "FirmwareVersion": "1.45.455b66-rev4",
    "AdditionalFirmwareVersions": {
        "Bootloader": "v2022.01",
        "Kernel": "Linux 5.13.0-30-generic arm71"
    },
    "NetworkProtocol": {
        "@odata.id": "/redfish/v1/Managers/BMC/NetworkProtocol"
    },
    "EthernetInterfaces": {
        "@odata.id": "/redfish/v1/Managers/BMC/EthernetInterfaces"
    },
    "SerialInterfaces": {
        "@odata.id": "/redfish/v1/Managers/BMC/SerialInterfaces"
    },
    "LogServices": {
        "@odata.id": "/redfish/v1/Managers/BMC/LogServices"
    },
    "Links": {
        "ManagerForServers": [
            {
                "@odata.id": "/redfish/v1/Systems/437XR1138R2"
            }
        ],
        "ManagerForChassis": [
            {
                "@odata.id": "/redfish/v1/Chassis/1U"
            }
        ],
        "ManagerInChassis": {
            "@odata.id": "/redfish/v1/Chassis/1U"
        }
    },
    "Actions": {
        "#Manager.Reset": {
            "target": "/redfish/v1/Managers/BMC/Actions/Manager.Reset",
            "ResetType@Redfish.AllowableValues": [
                "ForceRestart",
                "GracefulRestart"
            ]
        }
    }
}
```

The response contains information about the management entity the `Manager` resource represents. It's common to find information such as status, health, and FRU information about the manager. Networking and other external interface settings for the manager can be found in hyperlinks from the `Manager` resource.

The Manager section of the *Redfish Resource and Schema Guide* contains details for the properties that can be found in the `Manager` resource.

10 Actions

Actions are used for operations that do not map easily to CRUD (create, read, update, and delete) semantics. For example, performing a reset of a device is not as simple as modifying a desired property since a reset operation will likely start various sequences and state transitions internal to the device, which will cause many other properties in the model to change.

The `Actions` property in a resource is an object that contains the available actions that can be performed on the resource. Each property inside of the `Actions` property represents one of the actions supported on the resource. It also contains properties that describe the supported parameters and parameter values for each action. Clients can perform a `POST` operation on the action URI to invoke the action.

The following example shows the supported actions for a `ComputerSystem` resource found at the URI `/redfish/v1/Systems/437XR1138R2`:

```
{
    "@odata.id": "/redfish/v1/Systems/437XR1138R2",
    "@odata.type": "#ComputerSystem.v1_18_0.ComputerSystem",
    "Id": "437XR1138R2",
    "Name": "WebFrontEnd483",
    "Actions": {
        "#ComputerSystem.Reset": {
            "target": "/redfish/v1/Systems/437XR1138R2/Actions/ComputerSystem.Reset",
            "ResetType@Redfish.AllowableValues": [
                "On",
                "ForceOff",
                "GracefulShutdown",
                "GracefulRestart",
                "ForceRestart",
                "Nmi",
                "ForceOn",
                "PushPowerButton"
            ],
            "@Redfish.ActionInfo": "/redfish/v1/Systems/437XR1138R2/ResetActionInfo"
        }
    },
    ... <Other ComputerSystem properties>
}
```

In the previous example, there is one action supported named `#ComputerSystem.Reset`. The `target` property contains the URI on which the client performs the `POST` operation. It also contains `ResetType@Redfish.AllowableValues` to show the supported values for the `ResetType` parameter. A user can perform a graceful restart of the system with the following request:

```
curl -k -X POST 'https://<REDFISH-HOST>/redfish/v1/Systems/437XR1138R2/Actions/ComputerSystem.Reset' \
-H "Content-Type: application/json" -H 'X-Auth-Token: <SESSION-TOKEN>' \
-d '{ "ResetType": "GracefulRestart" }'
```

The `@Redfish.ActionInfo` property in the earlier example is another method that can be used to show supported parameters and values for actions. Clients can perform a `GET` operation on this URI to retreive an `ActionInfo` resource, which contains an array of objects that describe each supported parameter for the action. The following example shows an `ActionInfo` resource for the `#ComputerSystem.Reset` action:

```
{
    "@odata.id": "/redfish/v1/Systems/437XR1138R2/ResetActionInfo",
    "@odata.type": "#ActionInfo.v1_2_0.ActionInfo",
    "Id": "ResetActionInfo",
    "Name": "Reset Action Info",
    "Parameters": [
        {
            "Name": "ResetType",
            "Required": true,
            "DataType": "String",
            "AllowableValues": [
                "On",
                "ForceOff",
                "GracefulShutdown",
                "GracefulRestart",
                "ForceRestart",
                "Nmi",
                "ForceOn",
                "PushPowerButton"
            ]
        }
    ]
}
```

In the previous example, there is a single object in the `Parameters` array, which signifies there is only one parameter for the action. It tells us the parameter is named `ResetType`, it's mandatory to be specified in the action request, it's a string, and has a set of allowable values. The ActionInfo section of the *Redfish Resource and Schema Guide* contains details for the properties that can be found in the `ActionInfo` resource.

11 Redfish responses

The HTTP status code can be used for initial determination for how to process a response from a Redfish service. The following table provides guidance for processing responses:

HTTP status code	Indication and next steps
`200`	The request was successful. The response body contains the requested resource or action results.
`201`	The request was successful. If there is a response body, it contains a newly created resource. The `Location` HTTP header contains the URI of the new resource.
`202`	The request is still in process. See the Task management section for additional handling.
`204`	The request was successful and no response body is given.
`3<XX>`	The request is redirected to another URI. Perform a `GET` on the URI in the `Location` HTTP header.
`4<XX>`	The request was rejected due to a client-side error and the response body contains error information. This is typically due to unsupported properties or values or missing HTTP headers. See the Error responses section for additional handling.
`5<XX>`	The request was rejected due to a service-side error and the response body contains error information. Retrying the request might be successful. See the Error responses section for additional handling.

11.1 Task management

A service creates a task when a request will take additional time to process. This is useful for when a single operation, such as a `POST`, will take longer than traditional HTTP timeouts. For these cases, services respond with the HTTP `202 Accepted` status code. The response will contain the `Location` HTTP header, which clients use to poll the progress of the task with the `GET` operations. The response might also contain the `Retry-After` HTTP header to indicate a recommended polling interval. When the task is complete, the subsequent `GET` operation will return the response for the initial request.

The following pseudocode can be used as a template for monitoring tasks:

```
response = session.post(target_uri)
if response.status == 202:
    task_monitor = response.getheader('Location')
    sleep_time = response.getheader('Retry-After', 5)
    while response.status == 202:
        sleep(sleep_time)
        response = session.get(task_monitor)
```

The following flow shows the end-to-end interactions over HTTP where a task is used to manage a long operation:

```
POST /redfish/v1/Systems/1/Storage/1/Volumes HTTP/1.1
X-Auth-Token: <SESSION-TOKEN>
Content-Type: application/json

{
    "CapacityBytes": "53687091200",
    "RAIDType": "None",
    "Links": {
        "Drives": [
            {
                "@odata.id": "/redfish/v1/Chassis/1/Drives/4"
            }
        ]
    }
}

HTTP/1.1 202 Accepted
Content-Type: application/json
Location: /redfish/v1/TaskService/TaskMonitors/378
Retry-After: 10

{
    "@odata.id": "/redfish/v1/TaskService/Tasks/5",
    "@odata.type": "#Task.v1_5_1.Task",
    "Id": "5",
    "Name": "Task for volume creation",
    "TaskState": "New",
    "TaskStatus": "OK"
    "PercentComplete": 0,
    "StartTime": "2022-03-08T13:10:13-05:00",
    "Messages": [],
    "TaskMonitor": "/redfish/v1/TaskService/TaskMonitors/378"
}



GET /redfish/v1/TaskService/TaskMonitors/378 HTTP/1.1
X-Auth-Token: <SESSION-TOKEN>


HTTP/1.1 202 Accepted
Content-Type: application/json
Location: /redfish/v1/TaskService/TaskMonitors/378
Retry-After: 10

{
    "@odata.id": "/redfish/v1/TaskService/Tasks/5",
    "@odata.type": "#Task.v1_5_1.Task",
    "Id": "5",
    "Name": "Task for volume creation",
    "TaskState": "Running",
    "TaskStatus": "OK"
    "PercentComplete": 68,
    "StartTime": "2022-03-08T13:10:13-05:00",
    "Messages": [],
    "TaskMonitor": "/redfish/v1/TaskService/TaskMonitors/378"
}



GET /redfish/v1/TaskService/TaskMonitors/378 HTTP/1.1
X-Auth-Token: <SESSION-TOKEN>


HTTP/1.1 201 Created
Content-Type: application/json
Location: /redfish/v1/Systems/1/Storage/1/Volumes/3

{
    "@odata.id": "/redfish/v1/Systems/1/Storage/1/Volumes/3",
    "@odata.type": "#Volume.v1_6_0.Volume",
    "Id": "3",
    "Name": "Newly created volume",
    "Status": {
        "State": "Enabled",
        "Health": "OK"
    }
    "CapacityBytes": "53687091200",
    "RAIDType": "None",
    "Encrypted": false,
    "BlockSizeBytes": 512,
    "Links": {
        "Drives": [
            {
                "@odata.id": "/redfish/v1/Chassis/1/Drives/4"
            }
        ]
    }
}
```

In the first HTTP request, a user is creating a new `Volume` resource by performing a `POST` operation on the `/redfish/v1/Systems/1/Storage/1/Volumes` URI. The response to the request contains a `202 Accepted` status code, which indicates the operation is still being processed. The `Location` header contains the URI `/redfish/v1/TaskService/TaskMonitors/378`, which the client uses to check the status of the operation.

The next request shows the client checking the status of the original operation by performing a `GET` operation on the `/redfish/v1/TaskService/TaskMonitors/378`. The response contains a `202 Accepted` status code, which indicates the original operation is still being processed.

The final request is another `GET` operation on the `/redfish/v1/TaskService/TaskMonitors/378`. The response contains a `201 Created` status code, which indicates the original request is now complete. The response body contains the newly created `Volume` resource.

11.2 Error responses

When a service returns an HTTP `4<XX>` or `5<XX>` status code, the response body contains an error response. This type of response allows a service to provide detailed information about the failure. The following example response shows a single message that indicates the user specified the value `Red` for the `IndicatorLED` property, but the value is not allowed by the service.

```
{
    "error": {
        "code": "Base.1.14.PropertyValueNotInList",
        "message": "The value 'Red' for the property IndicatorLED is not in the ...",
        "@Message.ExtendedInfo": [
            {
                "@odata.type": "#Message.v1_1_2.Message",
                "MessageId": "Base.1.14.PropertyValueNotInList",
                "RelatedProperties": [ "/IndicatorLED" ],
                "Message": "The value 'Red' for the property IndicatorLED is not in the ...",
                "MessageArgs": [ "Red", "IndicatorLED" ],
                "Severity": "Warning",
                "MessageSeverity": "Warning",
                "Resolution": "Choose a value from the enumeration list that the ..."
            }
        ]
    }
}
```

The `@Message.ExtendedInfo` property inside `error` is an array, which allows for the service to provide multiple messages in the response. Users and client software can make decisions for corrective measures, such as modifying the request to the service, retrying the request, or performing service operations.

12 Query parameters

Services can support query parameters in `GET` requests. Query parameters allow clients to control the amount and type of data in the response from a service. The following table contains query parameters defined in the *Redfish Specification*:

Query parameter	Description
`excerpt`	Only return properties that have the excerpt schema annotation. If no excerpt schema annotation is defined for the resource, returns the entire resource.
`$expand`	Includes the hyperlink's contents in-line with retrieved resources.
`$filter`	Members of the resource collection that do not match the criteria specified in `$filter` are removed.
`only`	If there is only one member in the collection, the member is returned instead of the collection.
`$select`	Only return properties specified in the `$select` statement.
`$skip`	Only return members of a collection starting after the specified index.
`$top`	Limits the number of members of a collection that can be returned.

Clients can discover the supported query parameters on the service from the service root's `ProtocolFeaturesSupported` property. The following example shows a service supports `$expand`, `only`, and `excerpt`, but does not support `$select` or `$filter`.

```
{
    "ProtocolFeaturesSupported": {
        "ExpandQuery": {
            "ExpandAll": true,
            "Levels": true,
            "MaxLevels": 6,
            "Links": true,
            "NoLinks": true
        },
        "SelectQuery": false,
        "FilterQuery": false,
        "OnlyMemberQuery": true,
        "ExcerptQuery": true
    }
}
```

12.1 Controlling collection responses

The `$skip` and `$top` query parameters are used to help iterate over large resource collections. `$skip` controls the number of members to skip in the collection, removing collection members from the beginning of the list. `$top` limits the number of collection members to return, dropping members at the end of the collection that do not fit. The two used together can be used to iterate over a collection in portions at a time. In the following example, `$skip` specifies that the first 100 log entries are to be ignored and `$top` specifies that at most 25 log entries are allowed to be returned.

```
curl -k 'https://<REDFISH-HOST>/redfish/v1/Managers/BMC/LogServices/Log/Entries?$skip=100&$top=25' \
-H 'X-Auth-Token: <SESSION-TOKEN>' | python -m json.tool
```

The `$filter` query parameter is used to remove unwanted members of a resource collection. Clients provide comparison statements in the `$filter` parameter that contain desired property values for members to include in the collection response. In the following example, `$filter` specifies that only members whose `ReadingType` property contains `Temperature` are allowed in the collection response.

```
curl -k "https://<REDFISH-HOST>/redfish/v1/Chassis/1U/Sensors?$filter=ReadingType eq 'Temperature'" \
-H 'X-Auth-Token: <SESSION-TOKEN>' | python -m json.tool
```

The `only` query parameter is used to avoid subsequent `GET` operations when resource collections only contain one member. If the collection contains one member, then the member of the collection is returned as if the client performed a `GET` operation on the member of the collection. However, in other cases, the collection is returned as if `only` was not specified. In the following example, `only` is used when reading the system collection.

```
curl -k 'https://<REDFISH-HOST>/redfish/v1/Systems?only=' \
-H 'X-Auth-Token: <SESSION-TOKEN>' | python -m json.tool
```

12.2 Reducing response data

The `excerpt` query parameter is used to limit the properties returned from a resource to those marked as "excerpt" properties. As of this publication, the only standard Redfish resources that contain excerpt properties are `Sensor` and `Control` resources. Excerpt properties are those considered to be of high importance to most clients, such as the `Reading` property in a `Sensor` resource. In the following example, `excerpt` is used when reading the `AmbientTemp` sensor in the `1U` chassis.

```
curl -k 'https://<REDFISH-HOST>/redfish/v1/Chassis/1U/Sensors/AmbientTemp?excerpt=' \
-H 'X-Auth-Token: <SESSION-TOKEN>' | python -m json.tool
```

The `$select` query parameter is used to limit the properties returned from a resource to those specified in the query parameter itself. In the following example, `$select` is used to specify that `Status`, `PartNumber`, and `SerialNumber` are to be returned when reading the `1U` chassis.

```
curl -k 'https://<REDFISH-HOST>/redfish/v1/Chassis/1U?$select=Status,PartNumber,SerialNumber' \
-H 'X-Auth-Token: <SESSION-TOKEN>' | python -m json.tool
```

12.3 Including data from hyperlinks

The `$expand` query parameter is used to include resources referenced from hyperlinks within resource payload.

The `$expand` query parameter allows for the following options to control the types of hyperlinks to expand:

- `$expand=.` : Expand all hyperlinks not inside of the `Links` property.
- `$expand=~` : Only expand hyperlinks inside of the `Links` property.
- `$expand=*` : Expand every hyperlink in the resource.

Optionally, the `$levels` parameter specifies how deep to perform an expansion. The depth is in reference to how many hyperlinks are traversed to get to the final resource. For example, from a `ComputerSystemCollection` resource, it takes two levels to get to a `Bios` resource: the first level gets to the `ComputerSystem` resource and the second level goes into the `Bios` property from the `Computersystem` resource. If not specified, `$levels` is assumed to have the value 1. The query parameter `$expand=.($levels=2)` will perform two levels of expansion on hyperlinks not found in the `Links` property.

Special care needs to be made when performing expansion with hyperlinks found in `Links` and with multiple levels of expansion. Hyperlinks found in `Links` frequently cross reference each other, which can create expansion loops. Performing deep expansions of this type is likely to cause exponential growth in the response payload, which can easily hit memory limits of most BMCs.

In the following example, `$expand` is used to expand the system collection with two levels of depth while not expanding hyperlinks inside the `Links` property. Note that hyperlinks in the system such as `Processors` and `Memory` are expanded, but hyperlinks like `Chassis` and `ManagedBy` within `Links` are not expanded.

```
curl -k 'https://<REDFISH-HOST>/redfish/v1/Systems?$expand=.($levels=2)' \
-H 'X-Auth-Token: <SESSION-TOKEN>' | python -m json.tool
```

```
{
    "@odata.id": "/redfish/v1/Systems",
    "@odata.type": "#ComputerSystemCollection.ComputerSystemCollection",
    "Name": "System collection",
    "Members@odata.count": 1,
    "Members": [
        {
            "@odata.id": "/redfish/v1/Systems/437XR1138R2",
            "@odata.type": "#ComputerSystem.v1_18_0.ComputerSystem",
            "Id": "437XR1138R2",
            "Name": "WebFrontEnd483",
            "SystemType": "Physical",
            "AssetTag": "Chicago-45Z-2381",
            "Manufacturer": "Contoso",
            "Model": "3500",
            "SubModel": "RX",
            "SKU": "8675309",
            "SerialNumber": "437XR1138R2",
            "PartNumber": "224071-J23",
            "Description": "Web Front End node",
            "UUID": "38947555-7742-3448-3784-823347823834",
            "HostName": "web483",
            "Status": {
                "State": "Enabled",
                "Health": "OK",
                "HealthRollup": "OK"
            },
            "Processors": {
                "@odata.id": "/redfish/v1/Systems/437XR1138R2/Processors"
                "@odata.type": "#ProcessorCollection.ProcessorCollection",
                "Name": "Processor collection",
                "Members@odata.count": 1,
                "Members": [
                    {
                        "@odata.id": "/redfish/v1/Systems/437XR1138R2/Processors/CPU1"
                    }
                ]
            },
            "Memory": {
                "@odata.id": "/redfish/v1/Systems/437XR1138R2/Memory"
                "@odata.type": "#MemoryCollection.MemoryCollection",
                "Name": "Memory collection",
                "Members@odata.count": 2,
                "Members": [
                    {
                        "@odata.id": "/redfish/v1/Systems/437XR1138R2/Memory/DIMM1"
                    },
                    {
                        "@odata.id": "/redfish/v1/Systems/437XR1138R2/Memory/DIMM2"
                    }
                ]
            },
            "Links": {
                "Chassis": [
                    {
                        "@odata.id": "/redfish/v1/Chassis/1U"
                    }
                ],
                "ManagedBy": [
                    {
                        "@odata.id": "/redfish/v1/Managers/BMC"
                    }
                ]
            }
        }
    ]
}
```

12.4 Common combinations of query parameters

Query parameters can be combined with `&` in a single request.

`$filter`, `$skip`, and `$top` can be used together to perform iterative processing of a resource collection with members matching a specified criteria. Services apply the `$filter` query parameter first, followed by `$skip`, and then `$top`, regardless of the order given in the request. For example, the query parameters `$filter=Severity eq 'Critical'&$skip=100&$top=25` can be applied to a log entry collection to return 25 log entries that contain the severity `Critical`.

`$filter` and `$expand` can be used together when performing a `GET` request to a resource collection. `$filter` is used to reduce the members of the collection and `$expand` is used to show the contents of each of the members in the response instead of just their hyperlinks. For example, the query parameters `$filter=SystemType eq 'Physical'&$expand=.` can be applied to a system collection to return all physical systems to the client.

`$expand` and `excerpt` can be used together as a way to expand resources, but only return important data to the client. For example, the query parameters `$expand=.&excerpt=` can be applies to a sensor collection to return all `Sensor` resources in the collection, but only properties marked as "excerpt", such as `Reading`.

13 Managing user accounts

The `ManagerAccountCollection` resource found at `/redfish/v1/AccountService/Accounts` contains the user accounts. The following example shows a request to retreive the `ManagerAccountCollection` resource. The response shows one user account is present. While the last segment of the URI contains `1`, this is not necessarily the same as the username of the account. A subsequent `GET` on the URI for the account is required to discover its username.

```
curl -k 'https://<REDFISH-HOST>/redfish/v1/AccountService/Accounts' \
-H 'X-Auth-Token: <SESSION-TOKEN>' | python -m json.tool
```

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

13.1 Adding new user accounts

The preferred method to add new user accounts is to perform a `POST` operation on the `ManagerAccountCollection` resource URI to add a new member. The following example shows a new user named `Bob` added to the collection and assigned the URI `/redfish/v1/AccountService/Accounts/2`.

```
POST /redfish/v1/AccountService/Accounts HTTP/1.1
X-Auth-Token: <SESSION-TOKEN>
Content-Type: application/json

{
    "UserName": "Bob",
    "Password": "Secret12345",
    "RoleId": "Operator"
}


HTTP/1.1 201 Created
```

```
Content-Type: application/json
Location: /redfish/v1/AccountService/Accounts/2

{
    "@odata.id": "/redfish/v1/AccountService/Accounts/2",
    "@odata.type": "#ManagerAccount.v1_12_1.ManagerAccount",
    "Id": "2",
    "Name": "Bob's user account",
    "UserName": "Bob",
    "Password": null,
    "RoleId": "Operator",
    "Enabled": true,
    "AccountTypes": [
        "Redfish"
    ],
    "Links": {
        "Role": {
            "@odata.id": "/redfish/v1/AccountService/Roles/Operator"
        }
    }
}
```

There are some implementations that have restrictions where they are unable to support adding new user accounts with a `POST` operation. Instead, these implementations model fixed account slots, and the client is required to find an empty account slot and perform a `PATCH` operation to add a new user account. This can be discovered by inspecting the `Allow` HTTP response header when performing a `GET` on the `ManagerAccountCollection` resource. If the header does not contain `POST`, then the client will need to find an empty account slot and `PATCH` the new user's information. The following example shows a response that only contains `GET` in the `Allow` header.

```
GET /redfish/v1/AccountService/Accounts HTTP/1.1
X-Auth-Token: <SESSION-TOKEN>


HTTP/1.1 200 OK
Content-Type: application/json
Allow: GET

{
    "@odata.id": "/redfish/v1/AccountService/Accounts",
    "@odata.type": "#ManagerAccountCollection.ManagerAccountCollection",
    "Name": "Accounts Collection",
    "Members@odata.count": 4,
    "Members": [
        {
            "@odata.id": "/redfish/v1/AccountService/Accounts/1"
        },
        {
            "@odata.id": "/redfish/v1/AccountService/Accounts/2"
        },
        {
            "@odata.id": "/redfish/v1/AccountService/Accounts/3"
        },
        {
            "@odata.id": "/redfish/v1/AccountService/Accounts/4"
        }
    ]
}
```

To find an empty user account slot, the client will need to iterate over the members of the `ManagerAccountCollection` resource collection. An empty user account will contain an empty string for its `UserName` property and the value `false` for its `Enabled` property. The following pseudocode shows how to discover an empty user account slot.

```
account_collection = session.get("/redfish/v1/AccountService/Accounts")
empty_account_slot_uri = None
for member in account_collection.dict["Members"]
    account_slot = session.get(member["@odata.id"])
    if account_slot.dict["UserName"] == "" and not account_slot.dict["Enabled"]:
        empty_account_slot_uri = member["@odata.id"]
        break
```

Once an empty user account slot is found, the account slot can be updated with a `PATCH` operation with the new account information.

```
curl -k -D - -X PATCH 'https://<REDFISH-HOST>/redfish/v1/AccountService/Accounts/3' \
-H "Content-Type: application/json" \
-d '{ "UserName": "Fred", "Password": "P@ssw0RD", "RoleId": "ReadOnly", "Enabled": true }'
```

13.2 Removing user accounts

The preferred method to remove a user account is to perform a `DELETE` operation on the URI of the `ManagerAccount` resource that represents the account to remove. The following example shows a request to remove the user account found at the URI `/redfish/v1/AccountService/Accounts/2`.

```
curl -k -D - -X DELETE 'https://<REDFISH-HOST>/redfish/v1/AccountService/Accounts/2'
```

As described previously in the Adding new user accounts section, there are implementations that model fixed account slots. To remove a user account for these types of implementations, a `PATCH` operation is performed to set the account slot to be empty. This can be done by setting the `UserName` property to an empty string, the `Enabled` property to `false`, and the `RoleId` property to `ReadOnly`.

```
curl -k -D - -X PATCH 'https://<REDFISH-HOST>/redfish/v1/AccountService/Accounts/2' \
-H "Content-Type: application/json" -d '{ "UserName": "", "RoleId": "ReadOnly", "Enabled": false }'
```

14 ANNEX A (informative) Change log

Version	Date	Description
1.1.0	2024-08-01	Corrected the example `RelatedProperties` property in message objects throughout the guide to remove the leading `#` to meet syntax specified by RFC6901.
		Clarified the `Actions` property example in the **Actions** section to show how the information is obtained.
		Added the **Managing user accounts** section to give guidance on how to add and remove users.
1.0.0	2022-08-30	Initial release

15 ANNEX B (informative) curl command options

The example requests in this guide use curl commands with the following options:

Option	Description
`-d` , `--data <DATA>`	Sends specified `<DATA>` in a request to the server.
`-D` , `--dump-header <FILENAME>`	Dumps response headers. If `<FILENAME>` is `-` , dumps headers to the console.
`-H` , `--header <HEADER>`	Sends specified request header.
`-k` , `--insecure`	Proceeds even for insecure TLS connections.
`-L` , `--location`	Follows redirects.
`-s` , `--silent`	Runs in silent or quiet mode.
`-X` , `--request <COMMAND>`	Uses the `<COMMAND>` method instead of the default `GET` method.

16 Bibliography

- curl: command line tool and library for transferring data with URLs, https://curl.haxx.se/
- DMTF Redfish Mockup Server, https://github.com/dmtf/Redfish-Mockup-Server
- DMTF *Redfish Sessions*, https://www.dmtf.org/sites/default/files/Redfish_School-Sessions.pdf
- DMTF DSP0266, *Redfish Specification*, https://www.dmtf.org/dsp/DSP0266
- DMTF DSP2046, *Redfish Resource and Schema Guide*, https://www.dmtf.org/dsp/DSP2046
- Internet Engineering Task Force (IETF) RFC2616, R. Fielding et al, *Hypertext Transfer Protocol -- HTTP/1.1*, https://www.ietf.org/rfc/rfc2616.txt
- IETF RFC7230, R. Fielding, Ed. et al, *Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing*, https://www.ietf.org/rfc/rfc7230.txt
- IETF RFC7231, R. Fielding, Ed. et al, *Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content*, https://www.ietf.org/rfc/rfc7231.txt
- IETF RFC7617, J. Reschke, *The 'Basic' HTTP Authentication Scheme*, https://www.ietf.org/rfc/rfc7617.txt
- IETF RFC8259, T. Bray, Ed., *The JavaScript Object Notation (JSON) Data Interchange Format*, https://tools.ietf.org/html/rfc8259
- *JSON Schema: A Media Type for Describing JSON Documents draft-handrews-json-schema-01*, https://tools.ietf.org/html/draft-handrews-json-schema-01
- *OData Version 4.0 Part 3: Common Schema Definition Language (CSDL)*, https://docs.oasis-open.org/odata/odata/v4.0/os/part3-csdl/odata-v4.0-os-part3-csdl.html
- Postman, https://www.postman.com/
- Python json.tool, https://docs.python.org/3/library/json.html#module-json.tool
- *OpenAPI Specification*, https://swagger.io/specification/