



BSI Standards Publication

Information technology — Redfish scalable platforms management API specification

National foreword

This British Standard is the UK implementation of ISO/IEC 30115:2018.

The UK participation in its preparation was entrusted to Technical Committee IST/-/99, Participating IST members.

A list of organizations represented on this committee can be obtained on request to its secretary.

This publication does not purport to include all the necessary provisions of a contract. Users are responsible for its correct application.

© The British Standards Institution 2018
Published by BSI Standards Limited 2018

ISBN 978 0 580 99408 1

ICS 35.020

Compliance with a British Standard cannot confer immunity from legal obligations.

This British Standard was published under the authority of the Standards Policy and Strategy Committee on 30 April 2018.

Amendments/corrigenda issued since publication

Date	Text affected
------	---------------

Information technology — Redfish scalable platforms management API specification

Technologies de l'information — Spécification API (interface de programmation d'applications) relative à la gestion des plates-formes évolutives Redfish



Reference number
ISO/IEC 30115:2018(E)

AVA Documents
Document Sharing Hub
© ISO/IEC 2018



COPYRIGHT PROTECTED DOCUMENT

© ISO/IEC 2018

All rights reserved. Unless otherwise specified, or required in the context of its implementation, no part of this publication may be reproduced or utilized otherwise in any form or by any means, electronic or mechanical, including photocopying, or posting on the internet or an intranet, without prior written permission. Permission can be requested from either ISO at the address below or ISO's member body in the country of the requester.

ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva, Switzerland
Tel. +41 22 749 01 11
Fax +41 22 749 09 47
copyright@iso.org
www.iso.org

Published in Switzerland

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular the different approval criteria needed for the different types of document should be noted (see www.iso.org/directives).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation on the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT) see the following URL: www.iso.org/iso/foreword.html.

This document was prepared by the Distributed Management Task Force, Inc. (DMTF) (as DSP0266) and drafted in accordance with its editorial rules. It was adopted, under the JTC 1 PAS procedure, by Joint Technical Committee ISO/IEC JTC 1, *Information technology*.

CONTENTS

1. Abstract	8
2. Normative references	8
3. Terms and definitions	9
4. Symbols and abbreviated terms.....	12
5. Overview	12
5.1. Scope.....	13
5.2. Goals	13
5.3. Design tenets.....	14
5.4. Limitations.....	14
5.5. Additional design background and rationale.....	15
5.5.1. REST-based.....	15
5.5.2. Follow OData conventions	15
5.5.3. Model-oriented	16
5.5.4. Separation of protocol from data model.....	16
5.5.5. Hypermedia API service endpoint.....	16
5.6. Service elements	16
5.6.1. Synchronous and asynchronous operation support.....	16
5.6.2. Eventing mechanism.....	17
5.6.3. Actions.....	17
5.6.4. Service entry point discovery	17
5.6.5. Remote access support	18
5.7. Security.....	18
6. Protocol details.....	18
6.1. Use of HTTP	19
6.1.1. URIs	19
6.1.2. HTTP methods	20
6.1.3. HTTP redirect.....	21
6.1.4. Media types.....	21
6.1.5. ETags	21
6.2. Protocol version	22
6.3. Redfish-defined URIs and relative URI rules.....	23
6.4. Requests.....	24
6.4.1. Request headers	24
6.4.2. Read requests (GET)	26
6.4.3. HEAD	28
6.4.4. Data modification requests.....	29
6.5. Responses.....	32
6.5.1. Response headers	33
6.5.2. Status codes	35
6.5.3. Metadata responses.....	38
6.5.4. Resource responses	41

6.5.5. Resource Collection responses	49
6.5.6. Error responses.....	50
7. Data model and Schema.....	53
7.1. Schema repository.....	53
7.1.1. Programmatic access to schema files	53
7.2. Type identifiers.....	54
7.2.1. Type identifiers in JSON.....	54
7.3. Common naming conventions	54
7.4. Localization considerations	55
7.5. Schema definition	55
7.5.1. Common annotations	55
7.5.2. Schema documents	56
7.5.3. Resource type definitions.....	58
7.5.4. Resource properties.....	58
7.5.5. Reference properties.....	62
7.5.6. Resource actions	64
7.5.7. Resource extensibility	65
7.5.8. Oem property examples.....	67
7.6. Common Redfish resource properties.....	69
7.6.1. Id	69
7.6.2. Name.....	69
7.6.3. Description	55
7.6.4. Status	70
7.6.5. Links	70
7.6.6. Members	70
7.6.7. RelatedItem.....	70
7.6.8. Actions.....	17
7.6.9. OEM	70
7.7. Redfish resources.....	71
7.7.1. Current configuration.....	71
7.7.2. Settings	71
7.7.3. Services	71
7.7.4. Registry	72
7.8. Special resource situations.....	72
7.8.1. Absent resources	72
7.8.2. Schema variations.....	72
8. Service details.....	73
8.1. Eventing.....	73
8.1.1. Event message subscription	74
8.1.2. Event message objects	74
8.1.3. Subscription cleanup.....	75
8.2. Asynchronous operations	75
8.3. Resource tree stability	76

DSP0266	Redfish Scalable Platforms Management API Specification
8.4. Discovery	77
8.4.1. UPnP compatibility	77
8.4.2. USN format.....	77
8.4.3. M-SEARCH response	77
8.4.4. Notify, alive, and shutdown messages	78
9. Security	18
9.1. Protocols.....	78
9.1.1. TLS.....	78
9.1.2. Cipher suites	78
9.1.3. Certificates	79
9.2. Authentication	79
9.2.1. HTTP header security	79
9.2.2. Extended error handling	80
9.2.3. HTTP header authentication	80
9.2.4. Session Management	80
9.2.5. AccountService	83
9.2.6. Async tasks	83
9.2.7. Event subscriptions	83
9.2.8. Privilege model/Authorization	83
9.2.9. Redfish Service Operation to Privilege Mapping	84
10. Redfish Host Interface.....	92
11. Redfish Composability.....	92
11.1. Composition Requests.....	92
11.1.1. Specific Composition.....	92
12. ANNEX A (informative)	93
12.1. Change log	93

Foreword

The Redfish Scalable Platforms Management API ("Redfish") was prepared by the Scalable Platforms Management Forum of the DMTF.

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

Acknowledgments

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

- Jeff Autor - Hewlett Packard Enterprise
- Patrick Boyd - Dell Inc.
- David Brockhaus - Emerson Network Power
- Richard Brunner - VMware Inc.
- Lee Calcote - Seagate Technology
- P Chandrasekhar - Dell Inc.
- Chris Davenport - Hewlett Packard Enterprise
- Gamma Dean - Emerson Network Power
- Daniel Dufresne - EMC
- Samer El-Haj-Mahmoud - Lenovo, Hewlett Packard Enterprise
- George Ericson - EMC
- Wassim Fayed - Microsoft Corporation
- Mike Garrett - Hewlett Packard Enterprise
- Steve Geffin - Emerson Network Power
- Joe Handzik - Hewlett Packard Enterprise
- Jon Hass - Dell Inc.
- Jeff Hilland - Hewlett Packard Enterprise
- Chris Hoffman - Emerson Network Power
- Steven Krig - Intel Corporation
- John Leung - Intel Corporation
- Jagan Molleti - Dell Inc.
- Milena Natanov - Microsoft Corporation
- Michael Pizzo - Microsoft Corporation
- Chris Poblete - Dell Inc.
- Michael Raineri - EMC
- Irina Salvan - Microsoft Corporation
- Hemal Shah - Broadcom Limited
- Jim Shelton - Emerson Network Power
- Tom Slaughter - Intel Corporation
- Donnie Sturgeon - Emerson Network Power
- Pawel Szymanski - Intel Corporation
- Paul Vancil - Dell Inc.
- Linda Wu - Super Micro Computer, Inc.

1. Abstract

The Redfish Scalable Platforms Management API ("Redfish") is a new specification that uses RESTful interface semantics to access data defined in model format to perform out-of-band systems management. It is suitable for a wide range of servers, from stand-alone servers to rack mount and bladed environments but scales equally well for large scale cloud environments.

There are several out-of-band systems management standards (defacto and de jour) available in the industry. They all either vary widely in implementation, were developed for single server embedded environments or have their roots in antiquated software modeling constructs. There is no single industry standard that is simple to use, based on emerging programming standards, embedded friendly and capable of meeting large scale data center & cloud needs.

2. Normative references

The following referenced documents are indispensable for the application of this document. For dated or versioned references, only the edition cited (including any corrigenda or DMTF update versions) applies. For references without a date or version, the latest published edition of the referenced document (including any corrigenda or DMTF update versions) applies.

- [IETF RFC 3986](http://www.ietf.org/rfc/rfc3986.txt) T. Berners-Lee et al, Uniform Resource Identifier (URI): Generic Syntax, <http://www.ietf.org/rfc/rfc3986.txt>
- [IETF RFC 4627](http://www.ietf.org/rfc/rfc4627.txt), D. Crockford, The application/json Media Type for JavaScript Object Notation (JSON), <http://www.ietf.org/rfc/rfc4627.txt>
- [IETF RFC 5789](http://www.ietf.org/rfc/rfc5789.txt), L. Dusseault et al, PATCH method for HTTP, <http://www.ietf.org/rfc/rfc5789.txt>
- [IETF RFC 5280](http://www.ietf.org/rfc/rfc5280.txt), D. Cooper et al, Web linking, <http://www.ietf.org/rfc/rfc5280.txt>
- [IETF RFC 5988](http://www.ietf.org/rfc/rfc5988.txt), M. Nottingham, Web linking, <http://www.ietf.org/rfc/rfc5988.txt>
- [IETF RFC 6901](http://www.ietf.org/rfc/rfc6901.txt), P. Bryan, Ed. et al, JavaScript Object Notation (JSON) Pointer, <http://www.ietf.org/rfc/rfc6901.txt>
- [IETF RFC 6906](http://www.ietf.org/rfc/rfc6906.txt), E. Wilde, The 'profile' Link Relation Type, <http://www.ietf.org/rfc/rfc6906.txt>
- [IETF RFC 7230](http://www.ietf.org/rfc/rfc7230.txt), R. Fielding et al., Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing, <http://www.ietf.org/rfc/rfc7230.txt>
- [IETF RFC 7231](http://www.ietf.org/rfc/rfc7231.txt), R. Fielding et al., Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content, <http://www.ietf.org/rfc/rfc7231.txt>
- [IETF RFC 7232](http://www.ietf.org/rfc/rfc7232.txt), R. Fielding et al., Hypertext Transfer Protocol (HTTP/1.1): Conditional Requests, <http://www.ietf.org/rfc/rfc7232.txt>
- [IETF RFC 7234](http://www.ietf.org/rfc/rfc7234.txt), R. Fielding et al., Hypertext Transfer Protocol (HTTP/1.1): Caching, <http://www.ietf.org/rfc/rfc7234.txt>
- [IETF RFC 7235](http://www.ietf.org/rfc/rfc7235.txt), R. Fielding et al., Hypertext Transfer Protocol (HTTP/1.1): Authentication, <http://www.ietf.org/rfc/rfc7235.txt>

- [ISO/IEC Directives](http://isotc.iso.org/livelink/livelink.exe?func=ll&objId=4230456&objAction=browse&sort=subtypeH), Part 2, Rules for the structure and drafting of International Standards, <http://isotc.iso.org/livelink/livelink.exe?func=ll&objId=4230456&objAction=browse&sort=subtypeH>
- [JSON Schema, Core Definitions and Terminology, Draft 4](http://tools.ietf.org/html/draft-zyp-json-schema-04.txt) <http://tools.ietf.org/html/draft-zyp-json-schema-04.txt>
- [JSON Schema, Interactive and Non-Interactive Validation, Draft 4](http://tools.ietf.org/html/draft-fge-json-schema-validation-00.txt) <http://tools.ietf.org/html/draft-fge-json-schema-validation-00.txt>
- [OData Version 4.0 Part 1: Protocol](http://docs.oasis-open.org/odata/odata/v4.0/os/part1-protocol/odata-v4.0-os-part1-protocol.html). 24 February 2014. <http://docs.oasis-open.org/odata/odata/v4.0/os/part1-protocol/odata-v4.0-os-part1-protocol.html>
- [OData Version 4.0 Part 2: URL Conventions](http://docs.oasis-open.org/odata/odata/v4.0/os/part2-url-conventions/odata-v4.0-os-part2-url-conventions.html). 24 February 2014. <http://docs.oasis-open.org/odata/odata/v4.0/os/part2-url-conventions/odata-v4.0-os-part2-url-conventions.html>
- [OData Version 4.0 Part 3: Common Schema Definition Language \(CSDL\)](http://docs.oasis-open.org/odata/odata/v4.0/os/part3-csdl/odata-v4.0-os-part3-csdl.html). 24 February 2014. <http://docs.oasis-open.org/odata/odata/v4.0/os/part3-csdl/odata-v4.0-os-part3-csdl.html>
- [OData Version 4.0: Core Vocabulary](http://docs.oasis-open.org/odata/odata/v4.0/os/vocabularies/Org.OData.Core.V1.xml). 24 February 2014. <http://docs.oasis-open.org/odata/odata/v4.0/os/vocabularies/Org.OData.Core.V1.xml>
- [OData Version 4.0 JSON Format](http://docs.oasis-open.org/odata/odata-json-format/v4.0/os/odata-json-format-v4.0-os.html). 24 February 2014. <http://docs.oasis-open.org/odata/odata-json-format/v4.0/os/odata-json-format-v4.0-os.html>
- [OData Version 4.0: Units of Measure Vocabulary](http://docs.oasis-open.org/odata/odata/v4.0/os/vocabularies/Org.OData.Measures.V1.xml). 24 February 2014. <http://docs.oasis-open.org/odata/odata/v4.0/os/vocabularies/Org.OData.Measures.V1.xml>
- [Simple Service Discovery Protocol/1.0](http://tools.ietf.org/html/draft-cai-ssdp-v1-03). 28 October 1999. <http://tools.ietf.org/html/draft-cai-ssdp-v1-03>
- [The Unified Code for Units of Measure](http://www.unitsofmeasure.org/ucum.html). <http://www.unitsofmeasure.org/ucum.html>
- [W3C Recommendation of Cross-Origin Resource Sharing](http://www.w3.org/TR/cors/). 16 January 2014. <http://www.w3.org/TR/cors/>
- [SNIA TLS Specification for Storage Systems](http://www.snia.org/tls/). 20 November 2014. <http://www.snia.org/tls/>
- [DMTF DSP0270 Redfish Host Interface Specification](http://www.dmtf.org/sites/default/files/standards/documents/DSP0270_1.0.pdf), http://www.dmtf.org/sites/default/files/standards/documents/DSP0270_1.0.pdf

3. Terms and definitions

In this document, some terms have a specific meaning beyond the normal English meaning. Those terms are defined in this clause.

The terms "shall" ("required"), "shall not", "should" ("recommended"), "should not" ("not recommended"), "may", "need not" ("not required"), "can" and "cannot" in this document are to be interpreted as described in ISO/IEC Directives, Part 2, Annex H. The terms in parenthesis are alternatives for the preceding term, for use in exceptional cases when the preceding term cannot be used for linguistic reasons. Note that ISO/IEC Directives, Part 2, Annex H specifies additional alternatives. Occurrences of such additional alternatives shall be interpreted in their normal English meaning.

The terms "clause", "subclause", "paragraph", and "annex" in this document are to be interpreted as described in ISO/IEC Directives, Part 2, Clause 5.

The terms "normative" and "informative" in this document are to be interpreted as described in ISO/IEC Directives, Part 2, Clause 3. In this document, clauses, subclauses, or annexes labeled "(informative)" do not contain normative content. Notes and examples are always informative elements.

The following additional terms are used in this document.

Term	Definition
Baseboard Management Controller	An embedded device or service, typically an independent microprocessor or System-on-Chip with associated firmware, within a Computer System used to perform systems monitoring and management-related tasks, which are commonly performed out-of-band.
Collection	See Resource Collection.
CRUD	Basic intrinsic operations used by any interface: Create, Read, Update and Delete.
Event	A record that corresponds to an individual alert.
Managed System	In the context of this specification, a managed system is a system that provides information or status, or is controllable, via a Redfish-defined interface.
Member	A Member is a single resource instance contained in a Resource Collection
Message	A complete request or response, formatted in HTTP/HTTPS. The protocol, based on REST, is a request/response protocol where every Request should result in a Response.
Operation	The HTTP request methods that map generic CRUD operations. These are POST, GET, PUT/PATCH, HEAD and DELETE.
OData	The Open Data Protocol, as defined in OData-Protocol .
OData Service Document	The name for a resource that provides information about the Service Root. The Service Document provides a standard format for enumerating the resources exposed by the service that enables generic hypermedia-driven OData clients to navigate to the resources of the Redfish Service.
Redfish Alert Receiver	The name for the functionality that receives alerts from a Redfish Service. This functionality is typically software running on a remote system that is separate from the managed system.
Redfish Client	Name for the functionality that communicates with a Redfish Service and accesses one or more resources or functions of the Service.
Redfish Protocol	The set of protocols that are used to discover, connect to, and inter-communicate with a Redfish Service.

Term	Definition
Redfish Schema	The Schema definitions for Redfish resources. It is defined according to OData Schema representation that can be directly translated to a JSON Schema representation.
Redfish Service	Also referred to as the "Service". The set of functionality that implements the protocols, resources, and functions that deliver the interface defined by this specification and its associated behaviors for one or more managed systems.
Redfish Service Entry Point	Also referred to as "Service Entry Point". The interface through which a particular instance of a Redfish Service is accessed. A Redfish Service may have more than one Service Entry Point.
Request	A message from a Client to a Server. It consists of a request line (which includes the Operation), request headers, an empty line and an optional message body.
Resource	A Resource is addressable by a URI and is able to receive and process messages. A Resource can be either an individual entity, or a Collection that acts as a container for several other entities.
Resource Collection	A Resource Collection is a Resource that acts as a container of other Resources. The Members of a Resource Collection usually have similar characteristics. The container processes messages sent to the container. The Members of the container process messages sent only to that Member without affecting other Members of the container.
Resource Tree	A Resource Tree is a tree structure of JSON encoded resources accessible via a well-known starting URI. A client may discover the resources available on a Redfish Service by following the resource links from the base of the tree. NOTE for Redfish client implementation: Although the resources are a tree, the references between resources may result in graph instead of a tree. Clients traversing the resource tree must contain logic to avoid infinite loops.
Response	A message from a Server to a Client in response to a request message. It consists of a status line, response headers, an empty line and an optional message body.
Service Root	The term Service Root is used to refer to a particular resource that is directly accessed via the service entry point. This resource serves as the starting point for locating and accessing the other resources and associated metadata that together make up an instance of a Redfish Service.
Subscription	The act of registering a destination for the reception of events.

4. Symbols and abbreviated terms

The following additional abbreviations are used in this document.

Term	Definition
BMC	Baseboard Management Controller
CRUD	Create, Replace, Update and Delete
CSRF	Cross-Site Request Forgery
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol over TLS
IP	Internet Protocol
IPMI	Intelligent Platform Management Interface
JSON	JavaScript Object Notation
KVM-IP	Keyboard, Video, Mouse redirection over IP
NIC	Network Interface Card
PCI	Peripheral Component Interconnect
PCIe	PCI Express
TCP	Transmission Control Protocol
XSS	Cross-Site Scripting

5. Overview

The Redfish Scalable Platforms Management API ("Redfish") is a management standard using a data model representation inside of a hypermedia RESTful interface. Because it is based on REST, Redfish is easier to use and implement than many other solutions. Since it is model oriented, it is capable of expressing the relationships between components in modern systems as well as the semantics of the services and components within them. It is also easily extensible. By using a hypermedia approach to REST, Redfish can express a large variety of systems from multiple vendors. By requiring JSON representation, a wide variety of resources can be created in a denormalized fashion not only to improve scalability, but the payload can be easily interpreted by most programming environments as well as being relatively intuitive for a human examining the data. The model is exposed in terms of an interoperable

Redfish Schema, expressed in an OData Schema representation with translations to a JSON Schema representation, with the payload of the messages being expressed in a JSON following OData JSON conventions. The ability to externally host the Redfish Schema definition of the resources in a machine-readable format allows the meta data to be associated with the data without encumbering Redfish Services with the meta data, thus enabling more advanced client scenarios as found in many data center and cloud environments.

5.1. Scope

The scope of this specification is to define the protocols, data model, and behaviors, as well as other architectural components needed for an inter-operable, cross-vendor, remote and out-of-band capable interface that meets the expectations of Cloud and Web-based IT professionals for scalable platform management. While large scale systems are the primary focus, the specifications are also capable of being used for more traditional system platform management implementations.

The specifications define elements that are mandatory for all Redfish implementations as well as optional elements that can be chosen by system vendor or manufacturer. The specifications also define points at which OEM (system vendor) -specific extensions can be provided by a given implementation.

The specifications set normative requirements for Redfish Services and associated materials, such as Redfish Schema files. In general, the specifications do not set requirements for Redfish clients, but will indicate what a Redfish client should do in order to access and utilize a Redfish Service successfully and effectively.

The specifications do not set requirements that particular hardware or firmware must be used to implement the Redfish interfaces and functions.

5.2. Goals

There are many objectives and goals of Redfish as an architecture, as a data representation, and of the definition of the protocols that are used to access and interact with a Redfish Service. Redfish seeks to provide specifications that meet the following goals:

- Scalable – To support stand-alone machines to racks of equipment found in cloud service environments.
- Flexible – To support a wide variety of systems found in service today.
- Extensible – To support new and vendor-specific capabilities cleanly within the framework of the data model.
- Backward Compatible - To enable new capabilities to be added while preserving investments in earlier versions of the specifications.
- Interoperable – To provide a useful, required baseline that ensures common level of functionality and implementation consistency across multiple vendors.
- System-Focused – To efficiently support the most commonly required platform hardware

management capabilities that are used in scalable environments, while also being capable of managing current server environments.

- Standards based – To leverage protocols and standards that are widely accepted and used in environments today - in particular, programming environments that are being widely adopted for developing web-based clients today.
- Simple – To be directly usable by software developers without requiring highly specialized programming skills or systems knowledge.
- Lightweight – To reduce the complexity and cost of implementing and validating Redfish Services on managed systems.

5.3. Design tenets

The following design tenets and technologies are used to help deliver the previously stated goals and characteristics:

- Provide a RESTful interface using a JSON payload and an Entity Data Model
- Separate protocol from data model, allowing them to be revised independently
- Specify versioning rules for protocols and schema
- Leverage strength of internet protocol standards where it meets architectural requirements, such as JSON, HTTP, OData, and the RFCs referenced by this document.
- Focus on out-of-band access -- implementable on existing BMC and firmware products
- Organize the schema to present value-add features alongside standardized items
- Make data definitions as obvious in context as possible
- Maintain implementation flexibility. Do not tie the interface to any particular underlying implementation architecture. "Standardize the interface, not the implementation."
- Focus on most widely used 'common denominator' capabilities. Avoid adding complexity to address functions that are only valued by a small percentage of users.
- Avoid placing complexity on the management controller to support operations that can be better done at the client.

5.4. Limitations

Redfish does not guarantee that client software will never need to be updated. Examples that may require updates include accommodation of new types of systems or their components, data model updates, and so on. System optimization for an application will always require architectural oversight. However, Redfish does attempt to minimize instances of forced upgrades to clients using Schemas, strict versioning and forward compatibility rules and through separation of the protocols from the data model.

Inter-operable does not mean identical. A Redfish client may need to adapt to the optional elements that are provided by different vendors. Implementation and configurations of a particular product from a given vendor can also vary.

For example, Redfish does not enable a client to read a Resource Tree and write it to another Redfish

Service. This is not possible as it is a hypermedia API. Only the root object has a well known URI. The resource topology reflects the topology of the system and devices it represents. Consequently, different server or device types will result in differently shaped resource trees, potentially even for identical systems from the same manufacturer.

Additionally, not all Redfish resources are simple read/write resources. Implementations may follow other interaction patterns discussed later. As an example, user credentials or certificates cannot simply be read from one service and transplanted to another. Another example is the use of Setting Data instead of writing to the same resource that was read from.

Lastly, the value of links between resources and other elements can vary across implementations. Clients should not assume that links can be reused across different instantiations of a Redfish Service.

5.5. Additional design background and rationale

5.5.1. REST-based

This document defines a RESTful interface. Many service applications are exposed as RESTful interfaces.

There are several reasons to define a RESTful interface:

- It enables a lightweight implementation, where economy is a necessity (smaller data transmitted than SOAP, fewer layers to the protocol than WS-Man).
- It is a prevalent access method in the industry.
- It is easy to learn and easy to document.
- There are a number of toolkits & development environments that can be used for REST.
- It supports data model semantics and maps easily to the common CRUD operations.
- It fits with our design principle of simplicity.
- It is equally applicable to software application space as it is for embedded environments thus enabling convergence and sharing of code of components within the management ecosystem.
- It is schema agnostic so adapts well to any modeling language.
- By using it, Redfish can leverage existing security & discovery mechanisms in the industry.

5.5.2. Follow OData conventions

With the popularity of RESTful APIs, there are nearly as many RESTful interfaces as there are applications. While following REST patterns helps promote good practices, due to design differences between the many RESTful APIs there is no interoperability between them.

OData defines a set of common RESTful conventions and markup which, if adopted, provides for interoperability between APIs.

Adopting OData conventions for describing Redfish Schema, URL conventions, and naming and structure

of common properties in a JSON payload, not only encapsulate best practices for RESTful APIs but further enables Redfish Services to be consumed by a growing ecosystem of generic client libraries, applications, and tools.

5.5.3. Model-oriented

The Redfish model is built for managing systems. All resources are defined in OData Schema representation and translated to JSON Schema representation. OData is an industry standard that encapsulates best practices for RESTful services and provides interoperability across services of different types. JSON is being widely adopted in multiple disciplines and has a large number of tools and programming languages that accelerate development when adopting these approaches.

5.5.4. Separation of protocol from data model

The protocol operations are specified independently of the data model. The protocols are also versioned independently of the data model. The expectation is that the protocol version changes extremely infrequently, while the data model version is allowed to change as needed. This implies that innovation should happen primarily in the data model, not the protocols. It allows the data model to be extended and changed as needed without requiring the protocols or API version to change. Conversely, separating the protocols from the data model allows for changes to occur in the protocols without causing significant changes to the data model.

5.5.5. Hypermedia API service endpoint

Like other hypermedia APIs, Redfish has a single service endpoint URI and all other resources are accessible via opaque URIs referenced from the root. Any resource discovered through links found by accessing the root service or any service or resource referenced using references from the root service will conform to the same versions of the protocols supported by the root service.

5.6. Service elements

5.6.1. Synchronous and asynchronous operation support

While the majority of operations in this architecture are synchronous in nature, some operations can take a long time to execute, more time than a client typically wants to wait. For this reason, some operations can be asynchronous at the discretion of the service. The request portion of an asynchronous operation is no different from the request portion of a synchronous operation.

The use of HTTP Response codes enable a client to determine if the operation was completed synchronously or asynchronously. For more information, see the clause on [Tasks](#).

5.6.2. Eventing mechanism

In some situations it is useful for a service to provide messages to clients that fall outside the normal request/response paradigm. These messages, called events, are used by the service to asynchronously notify the client of some significant state change or error condition, usually of a time critical nature.

Only one style of eventing is currently defined by this specification - push style eventing. In push style eventing, when the server detects the need to send an event, it uses an HTTP POST to push the event message to the client. Clients can enable reception of events by creating a subscription entry in the Event Service, or an administrator can create subscriptions as part of the Redfish Service configuration. All subscriptions are persistent configuration settings.

The clause on [Eventing](#) further in this specification discusses the details of the eventing mechanism.

5.6.3. Actions

Operations can be divided into two sets: intrinsic and extrinsic. Intrinsic operations, often referred to as CRUD, are mapped to [HTTP methods](#). The protocol also has the ability to support extrinsic operations -- those operations that do not map easily to CRUD. Examples of extrinsic would be items that collectively would be better performed if done as a set (for scalability, ease of interface, server side semantic preservation or similar reasons) or operations that have no natural mapping to CRUD operations. One examples is system reset. It is possible to combine multiple operations into a single action. A system reset could be modeled as an update to state, but semantically the client is actually requesting a state change and not simply changing the value in the state.

In Redfish, these extrinsic operations are called **actions** and are discussed in detail in different parts of this specification.

The Redfish Schema defines certain standard actions associated with common Redfish resources. For these standard actions, the Redfish Schema contains the normative language on the behavior of the action. OEM extensions are also allowed to the Redfish [schema](#), including defining [actions](#) for existing resources.

5.6.4. Service entry point discovery

While the service itself is at a well-known URI, the service host must be discovered. Redfish, like UPnP, uses SSDP for discovery. SSDP is supported in a wide variety of devices, such as printers. It is simple, lightweight, IPv6 capable and suitable for implementation in embedded environments. Redfish is investigating additional service entry point discovery (e.g., DHCP-based) approaches.

For more information, see the clause on [Discovery](#)

5.6.5. Remote access support

A wide variety of remote access and redirection services are supported in this architecture. Critical to out-of-band environments are mechanisms to support Serial Console access, Keyboard Video and Mouse redirection (KVM-IP), Command Shell (i.e., Command Line interface) and remote Virtual Media. Support for Serial Console, Command Shell, KVM-IP and Virtual Media are all encompassed in this standard and are expressed in the Redfish Schema. This standard does not define the protocols or access mechanisms for accessing those devices and services. The Redfish Schema provides for the representation and configuration of those services, establishment of connections to enable those services and the operational status of those services. However, the specification of the protocols themselves are outside the scope of this specification.

5.7. Security

The challenge with security in a remote interface that is programmatic is to ensure both the interfaces used to interact with Redfish and the data being exchanged are secured. This means designing the proper security control mechanisms around the interfaces and securing the channels used to exchange the data. As part of this, specific behaviors are to be put in place including defining and using minimum levels of encryption for communication channels etc.

6. Protocol details

The Redfish Scalable Platform Management API is based on REST and follows OData conventions for interoperability, as defined in [OData-Protocol](#), JSON payloads, as defined in [OData-JSON](#), and a machine-readable representation of schema, as defined in [OData-Schema](#). The OData Schema representations include annotations to enable direct translation to JSON Schema representations for validation and consumption by tools supporting JSON Schema. Following these common standards and conventions increases interoperability and enables leveraging of existing tool chains.

Redfish follows the OData minimal conformance level for clients consuming minimal metadata.

Throughout this document, we refer to Redfish as having a protocol mapped to a data model. More accurately, HTTP is the application protocol that will be used to transport the messages and TCP/IP is the transport protocol. The RESTful interface is a mapping to the message protocol. For simplicity though, we will refer to the RESTful mapping to HTTP, TCP/IP and other protocol, transport and messaging layer aspects as the Redfish protocol.

The Redfish protocol is designed around a web service based interface model, and designed for network and interaction efficiency for both user interface (UI) and automation usage. The interface is specifically designed around the REST pattern semantics.

[HTTP methods](#) are used by the Redfish protocol for common CRUD operations and to retrieve header

information.

[Actions](#) are used for expanding operations beyond CRUD type operations, but should be limited in use.

[Media types](#) are used to negotiate the type of data that is being sent in the body of a message.

[HTTP status codes](#) are used to indicate the server's attempt at processing the request. [Extended error handling](#) is used to return more information than the HTTP error code provides.

The ability to send secure messages is important; the [Security](#) clause of this document describes specific TLS requirements.

Some operations may take longer than required for synchronous return semantics. Consequently, deterministic [asynchronous semantic](#) are included in the architecture.

6.1. Use of HTTP

HTTP is ideally suited to a RESTful interface. This clause describes how HTTP is used in the Redfish interface and what constraints are added on top of HTTP to assure interoperability of Redfish compliant implementations.

- A Redfish interface shall be exposed through a web service endpoint implemented using Hypertext Transfer Protocols, version 1.1 ([RFC7230](#), [RFC7231](#), [RFC7232](#)).

6.1.1. URIs

A URI is used to identify a resource, including the base service and all Redfish resources.

- Each unique instance of a resource shall be identified by a URI.
- A URI shall be treated by the client as opaque, and thus should not be attempted to be understood or deconstructed by the client outside of applying standard reference resolution rules as defined in clause 5, Reference Resolution, of [RFC3986](#).

To begin operations, a client must know a URI for a resource.

- Performing a GET operation yields a representation of the resource containing properties and links to associated resources.

The base resource URI is well known and is based on the protocol version. Discovering the URIs to additional resources is done through observing the associated resource links returned in previous responses. This type of API that is consumed by navigating URIs returned by the service is known as a Hypermedia API.

Redfish considers three parts of the URI as described in [RFC3986](#).

The first part includes the scheme and authority portions of the URI. The second part includes the root service and version. The third part is a unique resource identifier.

For example, in the following URL:

Example: `https://mgmt.vendor.com/redfish/v1/Systems/1`

- The first part is the scheme and authority portion (`https://mgmt.vendor.com`).
- The second part is the root service and version (`/redfish/v1/`).
- The third part is the unique resource path (`Systems/1`).

The scheme and authority part of the URI shall not be considered part of the unique *identifier* of the resource. This is due to redirection capabilities and local operations which may result in the variability of the connection portion. The remainder of the URI (the service and resource paths) is what *uniquely identifies* the resource, and this is what is returned in all Redfish payloads.

- The unique identifier part of a URI shall be unique within the implementation.

For example, a POST may return the following URI in the Location header of the response (indicating the new resource created by the POST):

Example: `/redfish/v1/Systems/2`

Assuming the client is connecting through an appliance named "mgmt.vendor.com", the full URI needed to access this new resource is `https://mgmt.vendor.com/redfish/v1/Systems/2`.

URIs, as described in [RFC3986](#), may also contain a query (`?query`) and a frag (`#frag`) components. Queries are addressed in the clause [Query Parameters](#). Fragments (frag) shall be ignored by the server when used as the URI for submitting an operation.

6.1.2. HTTP methods

An attractive feature of the RESTful interface is the very limited number of operations which are supported. The following table describes the general mapping of operations to HTTP methods. If the value in the column entitled "required" has the value "yes" then the HTTP method shall be supported by a Redfish interface.

HTTP Method	Interface Semantic	Required
POST	Object create, Object action, Eventing	Yes
GET	Object retrieval	Yes

HTTP Method	Interface Semantic	Required
PUT	Object replace	No
PATCH	Object update	Yes
DELETE	Object delete	Yes
HEAD	Object header retrieval	No
OPTIONS	Header retrieval, CORs preflight	No

Other HTTP methods are not allowed and shall receive a [405](#) response.

6.1.3. HTTP redirect

HTTP redirect allows a service to redirect a request to another URL. Among other things, this enables Redfish resources to alias areas of the data model.

- All Redfish Clients shall correctly handle HTTP redirect.

NOTE: Refer to the [Security](#) clause for security implications of HTTP Redirect

6.1.4. Media types

Some resources may be available in more than one type of representation. The type of representation is indicated by the media type.

In HTTP messages the media type is specified in the Content-Type header. A client can tell a service that it wants the response to be sent using certain media types by setting the HTTP Accept header to a list of the acceptable media types.

- All resources shall be made available using the JSON media type "application/json".
- Redfish Services shall make every resource available in a representation based on JSON, as specified in [RFC4627](#). Receivers shall not reject a message because it is encoded in JSON, and shall offer at least one response representation based on JSON. An implementation may offer additional representations using non-JSON media types.

Clients may request compression by specifying an [Accept-Encoding header](#) in the request.

- Services should support gzip compression when requested by the client.

6.1.5. ETags

In order to reduce the cases of unnecessary RESTful accesses to resources, the Redfish Service should

support associating a separate ETag with each resource.

- Implementations should support returning [ETag properties](#) for each resource.
- Implementations should support returning ETag headers for each response that represents a single resource.
- Implementations shall support returning ETag headers for GET requests of ManagerAccount resources.

The ETag is generated and provided as part of the resource payload because the service is in the best position to know if the new version of the object is different enough to be considered substantial. There are two types of ETags: weak and strong.

- Weak model -- only "important" portions of the object are included in formulation of the ETag. For instance, meta-data such as a last modified time should not be included in the ETag generation. The "important" properties that determine ETag change include writable settings and changeable attributes such as UUID, FRU data, serial numbers, etc.
- Strong model -- all portions of the object are included in the formulation of the ETag.

This specification does not mandate a particular algorithm for creating the ETag, but ETags should be highly collision-free. An ETag could be a hash, a generation ID, a time stamp or some other value that changes when the underlying object changes.

If a client [PUTs](#) or [PATCHes](#) a resource, it should include an ETag in the HTTP If-Match/If-None-Match header from a previous GET.

In addition to returning the ETag property on each resource,

- A Redfish Service should return the ETag header on client PUT/POST/PATCH
- A Redfish Service should return the ETag header on a GET of an individual resource

The format of the ETag header is:

```
ETag: W/"<string>"
```

6.2. Protocol version

The protocol version is separate from the version of the resources or the version of the Redfish Schema supported by them.

Each version of the Redfish protocol is strongly typed. This is accomplished using the URI of the Redfish Service in combination with the resource obtained at that URI, called the ServiceRoot.

The root URI for this version of the Redfish protocol shall be `"/redfish/v1/"`.

While the major version of the protocol is represented in the URI, the major version, minor version and errata version of the protocol are represented in the Version property of the ServiceRoot resource, as defined in the Redfish Schema for that resource. The protocol version is a string of the form:

MajorVersion.MinorVersion.Errata

where

- *MajorVersion* = integer: something in the class changed in a backward incompatible way.
- *MinorVersion* = integer: a minor update. New functionality may have been added but nothing removed. Compatibility will be preserved with previous minor versions.
- *Errata* = integer: something in the prior version was broken and needed to be fixed.

Any resource discovered through links found by accessing the root service or any service or resource referenced using references from the root service shall conform to the same version of the protocol supported by the root service.

A GET on the resource `"/redfish"` shall return the following body:

```
{
  "v1": "/redfish/v1/"
}
```

6.3. Redfish-defined URIs and relative URI rules

Redfish is a hypermedia API with a small set of defined URIs. All other resources are accessible via opaque URIs referenced from the root service. The following Redfish-defined URIs shall be supported by a Redfish Service:

URI	Description
<code>/redfish</code>	The URI that is used to return the version
<code>/redfish/v1/</code>	The URI for the Redfish Service Root
<code>/redfish/v1/odata</code>	The URI for the Redfish OData Service Document
<code>/redfish/v1/\$metadata</code>	The URI for the Redfish Metadata Document

In addition, the following URI without a trailing slash shall be either Redirected to the Associated Redfish-defined URI shown in the table below or treated by the service as the equivalent URI to the associated Redfish-defined URI:

URI	Associated Redfish-Defined URI
/redfish/v1	/redfish/v1/

All relative URIs used by the service shall start with a double forward slash ("//") and include the authority (e.g., //mgmt.vendor.com/redfish/v1/Systems) or a single forward slash ("/") and include the absolute-path (e.g., /redfish/v1/Systems).

6.4. Requests

This clause describes the requests that can be sent to Redfish Services.

6.4.1. Request headers

HTTP defines headers that can be used in request messages. The following table defines those headers and their requirements for Redfish Services. Note that these are requirements for the Redfish Services, and not the clients sending the HTTP requests.

- Redfish Services shall understand and be able to process the headers in the following table as defined by the HTTP 1.1 specification if the value in the Required column is set to "Yes".
- Redfish Services shall understand and be able to process the headers in the following table as defined by the HTTP 1.1 specification if the value in the Required column is set to "Conditional" under the conditions noted in the description.
- Redfish Services should understand and be able to process the headers in the following tables as defined by the HTTP 1.1 specification if the value in the Required column is set to "No".

Header	Required	Supported Values	Description
Accept	Yes	RFC 7231	Indicates to the server what media type(s) this client is prepared to accept. Services shall support requests for resources with an Accept header including application/json or application/json; charset=utf-8. Services shall support requests for metadata with an Accept header including application/xml or application/xml; charset=utf-8.
Accept-Encoding	No	RFC 7231	Indicates if gzip encoding can be handled by the client. If an Accept-Encoding header is present in a request and the service cannot send a response which is acceptable according to the Accept-Encoding header, then the service should respond

Header	Required	Supported Values	Description
			with status code 406 . Services should not return responses gzip encoded if the Accept-Encoding header is not present in the request.
Accept-Language	No	RFC 7231	This header is used to indicate the language(s) requested in the response. If this header is not specified, the appliance default locale will be used.
Content-Type	Conditional	RFC 7231	Describes the type of representation used in the message body. Content-Type shall be required in requests that include a request body. Services shall accept Content-Type values of application/json or application/json;charset=utf-8.
Content-Length	No	RFC 7231	Describes the size of the message body. An optional means of indicating size of the body uses Transfer-Encoding: chunked, which does not use the Content-Length header. If a service does not support Transfer-Encoding and needs Content-Length instead, the service will respond with status code 411 .
OData-MaxVersion	No	4.0	Indicates the maximum version of OData that an odata-aware client understands
OData-Version	Yes	4.0	Services shall reject requests which specify an unsupported OData version. If a service encounters a version that it does not support, the service should reject the request with status code [412] (#status-412). If client does not specify an Odata-Version header, the client is outside the boundaries of this specification.
Authorization	Conditional	RFC 7235 , Section 4.2	Required for Basic Authentication
User-Agent	Yes	RFC 7231	Required for tracing product tokens and their version. Multiple product tokens may be listed.
Host	Yes	RFC 7230	Required to allow support of multiple origin hosts at a single IP address.

Header	Required	Supported Values	Description
Origin	Yes	W3C CORS , Section 5.7	Used to allow web applications to consume Redfish Service while preventing CSRF attacks.
Via	No	RFC 7230	Indicates network hierarchy and recognizes message loops. Each pass inserts its own VIA.
Max-Forwards	No	RFC 7231	Limits gateway and proxy hops. Prevents messages from remaining in the network indefinitely.
If-Match	Conditional	RFC 7232	If-Match shall be supported on PUT and PATCH requests for resources for which the service returns ETags, to ensure clients are updating the resource from a known state.
If-None-Match	No	RFC 7232	If this HTTP header is present, the service will only return the requested resource if the current ETag of that resource does not match the ETag sent in this header. If the ETag specified in this header matches the resource's current ETag, the status code returned from the GET will be 304 .

- Redfish Services shall understand and be able to process the headers in the following table as defined by this specification if the value in the Required column is set to "yes".

Header	Required	Supported Values	Description
X-Auth-Token	Yes	Opaque encoded octet strings	Used for authentication of user sessions. The token value shall be indistinguishable from random.

6.4.2. Read requests (GET)

The GET method is used to retrieve a representation of a resource. The service will return the representation using one of the media types specified in the Accept header, subject to requirements in the Media Types clause [Media Types](#). If the Accept header is not present, the service will return the resources representations as application/json.

- The HTTP GET method shall be used to retrieve a resource without causing any side effects.
- The service shall ignore the content of the body on a GET.
- The GET operation shall be idempotent in the absence of outside changes to the resource.

6.4.2.1. Service root request

The root URL for Redfish version 1 services shall be `"/redfish/v1/"`.

The root URL for the Service returns a ServiceRoot resource as defined by this specification.

Services shall not require authentication in order to retrieve the service root and `"/redfish"` documents.

6.4.2.2. Metadata document request

Redfish Services shall expose a [metadata document](#) describing the service at the `"/redfish/v1/$metadata"` resource. This metadata document describes the resources available at the root, and references additional metadata documents describing the full set of resource types exposed by the service.

Services shall not require authentication in order to retrieve the metadata document.

6.4.2.3. OData service document request

Redfish Services shall expose an [OData Service Document](#), at the `"/redfish/v1/odata"` resource. This service document provides a standard format for enumerating the resources exposed by the service, enabling generic hypermedia-driven OData clients to navigate to the resources of the service.

Services shall not require authentication in order to retrieve the service document.

6.4.2.4. Resource retrieval requests

Clients request resources by issuing GET requests to the URI for the individual resource. The URI for a resource may be obtained from a [resource identifier property](#) returned in a previous request (for example, within the [links](#) clause of a previously returned resource). Services may, but are not required to, support the convention of retrieving individual properties of a Resource by appending a segment containing the property name to the URI of the resource.

6.4.2.4.1. Query parameters

When the resource addressed is a Resource Collection, the client may use the following paging query options to specify that a subset of the Members of that Resource Collection be returned. These paging query options apply specifically to the "Members" array property within a ResourceCollection.

Attribute	Description	Example
\$skip	Integer indicating the number of	<code>http://resourcecollection?\$skip=5</code>

Attribute	Description	Example
	Members in the Resource Collection to skip before retrieving the first resource.	
\$top	Integer indicating the number of Members to include in the response. The minimum value for this parameter is 1. The default behavior is to return all Members.	<code>http://resourcecollection?\$top=30</code>

- Services should support the \$top and \$skip query parameters.
- Implementation shall return the 501, Not Implemented, status code for any query parameters starting with "\$" that are not supported, and should return an [extended error](#) indicating the requested query parameter(s) not supported for this resource.
- Implementations shall ignore unknown or unsupported query parameters that do not begin with "\$".

6.4.2.4.2. Retrieving Resource Collections

Retrieving a Resource Collection is done by sending the HTTP GET method to the URI for that resource. The response includes properties of the Resource Collection including an array of its Members. A subset of the Members can be returned using [client paging query parameters](#).

No requirements are placed on implementations to return a consistent set of Members when a series of requests using paging query parameters are made over time to obtain the entire set of members. It is possible that this could result in missed or duplicate elements being retrieved if multiple GETs are used to retrieve the Members array instances using paging.

- Clients shall not make assumptions about the URIs for the Members of a Resource Collection.
- Retrieved Resource Collections shall always include the [count](#) property to specify the total number of entries in its "Members" array.
- Regardless of paging, see [partial results](#), the total number of resources referenced by the Members array shall be returned in the [count](#) property.

6.4.3. HEAD

The HEAD method differs from the GET method in that it MUST NOT return message body information. However, all of the same meta information and status codes in the HTTP headers will be returned as though a GET method were processed, including authorization checks.

- Services may support the HEAD method in order to return meta information in the form of HTTP response headers.