DMTF is a not-for-profit association of industry members dedicated to promoting enterprise and systems management and interoperability. Members and non-members may reproduce DMTF specifications and documents, provided that correct attribution is given. As DMTF specifications may be revised from time to time, the particular version and release date should always be noted.

Implementation of certain elements of this standard or proposed standard may be subject to third party patent rights, including provisional patent rights (herein “patent rights”). DMTF makes no representations to users of the standard as to the existence of such rights, and is not responsible to recognize, disclose, or identify any or all such third party patent right, owners or claimants, nor for any incomplete or inaccurate identification or disclosure of such rights, owners or claimants. DMTF shall have no liability to any party, in any manner or circumstance, under any legal theory whatsoever, for failure to recognize, disclose, or identify any such third party patent rights, or for such party’s reliance on the standard or incorporation thereof in its product, protocols or testing procedures. DMTF shall have no liability to any party implementing such standard, whether such implementation is foreseeable or not, nor to any patent owner or claimant, and shall have no liability or responsibility for costs or losses incurred if a standard is withdrawn or modified after publication, and shall be indemnified and held harmless by any party implementing the standard from any and all claims of infringement by a patent owner for such implementations.

For information about patents held by third-parties which have notified DMTF that, in their opinion, such patent may relate to or impact implementations of DMTF standards, visit http://www.dmtf.org/about/policies/disclosures.php.

This document's normative language is English. Translation into other languages is permitted.
CONTENTS

Foreword ................................................................. 5
  Acknowledgments .................................................. 5
1 Abstract ...................................................................... 6
2 Overview ...................................................................... 7
3 Normative references .................................................. 8
4 Terms and definitions .................................................. 9
5 Design tenets .................................................................. 10
6 Profile tools ................................................................... 11
7 Profile repository ........................................................ 12
8 Profile document definition .......................................... 13
  8.1 File name conventions ............................................. 13
  8.2 Basic functions ...................................................... 13
    8.2.1 Required profiles ............................................. 14
    8.2.2 Example ........................................................ 14
  8.3 Protocol requirements .............................................. 15
    8.3.1 Example ........................................................ 16
    8.3.2 Requirement values .......................................... 16
  8.4 Resource (schema) requirements ............................... 17
    8.4.1 Schema-level functions ..................................... 18
      8.4.1.0.1 URI patterns .......................................... 19
      8.4.1.1 Example .................................................. 19
    8.4.2 Resource use cases ............................................ 20
      8.4.2.1 Use case-level functions ............................... 21
      8.4.2.2 Use case example ....................................... 21
    8.4.3 Property-level functions ................................... 22
      8.4.3.1 Example .................................................. 23
      8.4.3.2 Comparison ............................................. 24
      8.4.3.3 Read requirement ....................................... 25
      8.4.3.4 Write requirement ..................................... 26
      8.4.3.5 Conditional requirements ............................. 26
        8.4.3.5.1 Parent and subordinate resources .............. 27
        8.4.3.5.2 Example ............................................ 28
        8.4.3.5.3 Compare property .................................. 29
        8.4.3.5.4 Examples ........................................... 29
      8.4.3.6 Handling deprecated properties .................... 31
        8.4.3.6.1 Examples ........................................... 31
    8.4.4 Action requirements ......................................... 32
      8.4.4.1 Parameters ............................................... 33
      8.4.4.2 Example .................................................. 33
  8.5 Registry-level requirements ...................................... 34
    8.5.1 Messages ........................................................ 35
Foreword

The Redfish Interoperability Profile specification 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.

Acknowledgments

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

- Richelle Ahlvers — Broadcom Inc.
- Jeff Autor — Hewlett Packard Enterprise
- George Ericson — Dell Technologies
- Tomas Gonzalez — Majec Systems, Inc.
- Jon Hass — Dell Technologies
- Jeff Hilland — Hewlett Packard Enterprise
- John Leung — Intel Corporation
- Michael Raineri — Dell Technologies
- Paul Vancil — Dell Technologies
1 Abstract

As schema definitions for the Redfish Specification ("Redfish") are designed to provide significant flexibility, and allow conforming implementations on a wide variety of products, very few properties within the schemas are required by the Redfish Specification. But consumers and software developers need a more rigidly defined set of required properties (features) in order to accomplish management tasks. This set allows users to compare implementations, specify needs to vendors, and allows software to rely on the availability of data. To provide that "common ground", a Redfish interoperability profile allows the definition of a set of schemas and property requirements, which meet the needs of a particular class of product or service.

A Redfish interoperability profile is a JSON document that contains schema-level, property-level, and registry-level requirements. At the property level, these requirements can include a variety of conditions under which the requirement applies.
2 Overview

The Redfish Specification separates the definition of the protocol from the data model (schema), and in addition, allows each resource defined in the data model to be revised independently. While this creates significant flexibility and extensibility, it can cause confusion when developers and end users attempt to answer the question “What version of Redfish does your product support?” The answer is not a simple one, because fully describing a Redfish implementation would require listing each property supported in each schema implemented, as well as the protocol version and supported features. That level of detail and version reporting would be extremely cumbersome to create or maintain, and difficult to use to compare implementations across products or vendors.

The Redfish interoperability profile concept was created to simplify that process, by providing a means to communicate the functionality provided with a single statement - that an implementation meets the requirements set forth in a Redfish interoperability profile.

A profile is constructed in a machine-readable (JSON) document that serves two purposes. First, it enables the creation of a human-readable document that merges the profile requirements with the Redfish schema into a single document for developers or users. Second, it allow a conformance test utility to test a Redfish service implementation for conformance with the profile.
3 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.


Redfish Interoperability Profiles

Published Version 1.6.0
4 Terms and definitions

Some terms and phrases in this document have specific meanings beyond their typical English meanings. This clause defines those terms and phrases.

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, Clause 7. 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, Clause 7 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 6.

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 term "deprecated" in this document is to be interpreted as material that is not recommended for use in new development efforts. Existing and new implementations may use this material, but they should move to the favored approach. Deprecated material may be implemented in order to achieve backwards compatibility. Deprecated material should contain references to the last published version that included the deprecated material as normative material and to a description of the favored approach. Deprecated material may be removed from the next major version of the specification.
5 Design tenets

All profile entries, at the profile, resource, or property level, are "additive". That is, each requirement can only apply more rigid requirements that override less rigid requirements.

Profile requirements do not allow for exclusions of data. Implementations are able to provide more data in their resources than required by a profile, as an implementation likely addresses multiple use cases or profiles. This includes both standard properties and OEM extensions.
6 Profile tools

A free, open source utility has been created by the Redfish Forum to verify that a Redfish service implementation conforms to the requirements included in a Redfish interoperability profile. The Redfish Interop Validator is available for download from DMTF's organization on GitHub at: https://github.com/DMTF/Redfish-Interop-Validator

A documentation generator has also been created by the Redfish Forum that will create a "guide" using the Redfish schema and the profile document. The output is intended for use by both developers and end users to understand the implementation requirements of a profile. The Redfish Documentation Generator is available for download from DMTF's organization on GitHub at: https://github.com/DMTF/Redfish-Tools
7 Profile repository

Redfish interoperability profiles published or re-distributed by DMTF are available for download from the Redfish profile repository located at: http://redfish.dmtf.org/profiles
8 Profile document definition

A Redfish interoperability profile is specified in a JSON document. The JSON objects and properties contained in the document are described in this specification, and are also available in a JSON-schema form (RedfishInteroperabilityProfile.v1_x_x.json) from DMTF’s Redfish schema repository at http://redfish.dmtf.org/profiles for download. The json-schema can be used to validate a profile document to ensure compatibility with automated conformance tools or utilities.

The JSON document structure is intended to align easily with JSON payloads retrieved from Redfish service implementations, to allow for easy comparisons and conformance testing. Many of the properties defined within this structure have assumed default values that correspond with the most common use case, so that those properties can be omitted from the document for brevity.

8.1 File name conventions

The document that describes a profile follows the Redfish schema file naming conventions from the Redfish Specification. The file name format for profiles shall be:

<ProfileName>.v<MajorVersion>_<MinorVersion>_<Errata>.json

For example, the file name of the BasicServer profile v1.2.0 is BasicServer.v1_2_0.json. The file name shall include the profile name and version, which matches those property values within the document.

8.2 Basic functions

At the top level of the JSON document are the basic properties, which describe the profile, including authorship and contact information, versioning, and other profiles to include in order to build upon previous work.

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SchemaDefinition</td>
<td>string</td>
<td>The JSON schema that defines this Redfish interoperability profile document and can be used to validate its contents.</td>
</tr>
<tr>
<td>ProfileName</td>
<td>string</td>
<td>The name of this Redfish profile.</td>
</tr>
<tr>
<td>ProfileVersion</td>
<td>string</td>
<td>The version of this Redfish profile. The version shall be represented using a &lt;major&gt;.&lt;minor&gt;.&lt;errata&gt; format.</td>
</tr>
<tr>
<td>Purpose</td>
<td>string</td>
<td>A description of the purpose of this Redfish profile, such as its intended target audience, product segments, etc.</td>
</tr>
</tbody>
</table>
### 8.2.1 Required profiles

The `RequiredProfiles` object contains object properties that are named to match the name of the profile to be included. Each of these sub-objects contains the properties listed below.

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Repository</td>
<td>string</td>
<td>A URI providing the location of the repository that contains the JSON files to be included. The filenames of the JSON files contained in the repository are expected to follow the Redfish interoperability profile filename conventions. If absent, the repository location shall be the Redfish profile repository (<a href="http://redfish.dmtf.org/profiles">http://redfish.dmtf.org/profiles</a>).</td>
</tr>
<tr>
<td>MinVersion</td>
<td>string</td>
<td>The minimum version required by this Redfish profile. The version shall be represented using a <code>&lt;major&gt;.&lt;minor&gt;.&lt;errata&gt;</code> format, including an optional errata version. If this property is absent, the minimum value shall be <code>1.0.0</code>.</td>
</tr>
</tbody>
</table>

### 8.2.2 Example

The following is an example of the top-level properties in a profile, with two required profiles included.
8.3 Protocol requirements

An object named `Protocol` contains properties which describe Redfish protocol functionality that is not related to the supported schemas or properties. Therefore, these functions cannot be validated by comparing retrieved JSON payloads.

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MinVersion</td>
<td>string</td>
<td>Indicates the minimum version of the Redfish Specification protocol support required by this profile. This version shall be reported by the Redfish service in the <code>ServiceRoot</code> resource property <code>RedfishVersion</code>. The version shall be represented using a <code>&lt;major&gt;.&lt;minor&gt;.&lt;errata&gt;</code> format, including an optional errata version. If this property is absent, the minimum value shall be <code>1.0.0</code>.</td>
</tr>
<tr>
<td>Discovery</td>
<td>string</td>
<td>Indicates support requirements for the Redfish SSDP discovery protocol. If this property is absent, there is no requirement for SSDP. See the Requirement values clause.</td>
</tr>
<tr>
<td>HostInterface</td>
<td>string</td>
<td>Indicates support requirements for the Redfish host interface. If this property is absent, there is no requirement for a host interface. See the Requirement values clause.</td>
</tr>
</tbody>
</table>
### Property Type Description

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExpandQuery</td>
<td>string</td>
<td>Indicates support requirements for the $expand query parameter. Additional $expand support requirements may be specified in the resource entry for the ProtocolFeaturesSupported object within ServiceRoot. If this property is absent, there is no requirement for support of the $expand query parameter. See the Requirement values clause.</td>
</tr>
<tr>
<td>FilterQuery</td>
<td>string</td>
<td>Indicates support requirements for the $filter query parameter. If this property is absent, there is no requirement for support of the $filter query parameter. See the Requirement values clause.</td>
</tr>
<tr>
<td>SelectQuery</td>
<td>string</td>
<td>Indicates support requirements for the $select query parameter. If this property is absent, there is no requirement for support of the $select query parameter. See the Requirement values clause.</td>
</tr>
<tr>
<td>OnlyQuery</td>
<td>string</td>
<td>Indicates support requirements for the only query parameter. If this property is absent, there is no requirement for support of the only query parameter. See the Requirement values clause.</td>
</tr>
<tr>
<td>ExcerptQuery</td>
<td>string</td>
<td>Indicates support requirements for the excerpt query parameter. If this property is absent, there is no requirement for support of the excerpt query parameter. See the Requirement values clause.</td>
</tr>
</tbody>
</table>

### 8.3.1 Example

```json
{
    "Protocol": {
        "MinVersion": "1.6",
        "Discovery": "Mandatory",
        "HostInterface": "Recommended",
        "ExpandQuery": "Mandatory",
        "SelectQuery": "None",
        "FilterQuery": "Recommended",
        "OnlyQuery": "Mandatory",
        "ExcerptQuery": "Recommended"
    }
}
```

### 8.3.2 Requirement values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mandatory</td>
<td>This protocol feature is required for this profile.</td>
</tr>
<tr>
<td>Recommended</td>
<td>It is recommended, but not required, that this protocol feature be supported.</td>
</tr>
</tbody>
</table>
### 8.4 Resource (schema) requirements

The primary content in a Redfish profile is the set of supported property requirements. As Redfish is organized and defined by schema-backed JSON resources, these requirements are also organized by schema.

For each schema, an object is created in the JSON document, named to match the schema name. Within this object, properties describe the location of the schema file, and schema-level requirements. Within each schema-level object is a `PropertyRequirements` object that describes the property-level requirements for that schema. The definition of both the schema/resource-level and property-level requirements are accomplished using the same mechanisms, which are described in the next clause.

The structure of the resource and property requirements is:

```json
{
    "<schema-name>": {
        "MinVersion": "<version>",
        "CreateResource": "<boolean>",
        "DeleteResource": "<boolean>",
        "UpdateResource": "<boolean>",
        "URIs": ["<uri-pattern>", "<uri-pattern>"],
        "PropertyRequirements": {
            "<property-name>": {
                "<property-requirements>": "<property-requirements-value>"
            }
        },
        "ActionRequirements": {
            "<action-name>": {
                "<action-requirements>": "<action-requirements-value>"
            }
        }
    }
}
```

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>This feature is not required by this profile. It is listed here for clarity.</td>
</tr>
</tbody>
</table>
8.4.1 Schema-level functions

The following options are available at the schema level:

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Repository</td>
<td>string</td>
<td>A URI providing the location of the repository that contains the JSON files to be included. The filenames of the JSON files contained in the repository are expected to follow the Redfish schema filename conventions. If absent, the repository location shall be the Redfish schema repository (<a href="http://redfish.dmtf.org/schemas">http://redfish.dmtf.org/schemas</a>).</td>
</tr>
<tr>
<td>MinVersion</td>
<td>string</td>
<td>The minimum version required by this Redfish profile. The version shall be represented using a &lt;major&gt;.&lt;minor&gt;.&lt;errata&gt; format, including an optional errata version. If this property is absent, the minimum value shall be 1.0.0.</td>
</tr>
<tr>
<td>ReadRequirement</td>
<td>string</td>
<td>Resource-level requirement for this schema. See the Read requirement clause.</td>
</tr>
<tr>
<td>Purpose</td>
<td>string</td>
<td>A description of the purpose of this requirement. This text can provide justification or reasoning behind the requirement for use in the profile documentation.</td>
</tr>
<tr>
<td>ConditionalRequirements</td>
<td>object</td>
<td>Resource-level conditional requirements that apply to instances of this schema. See the Conditional requirements clause.</td>
</tr>
<tr>
<td>CreateResource</td>
<td>boolean</td>
<td>Specifies a requirement that a user may create a member of this resource. This normally applies to Redfish resource collections. If this property is absent, there is no requirement to support creation of members of this resource.</td>
</tr>
<tr>
<td>DeleteResource</td>
<td>boolean</td>
<td>Specifies a requirement that a user may delete a member of this resource. This normally applies to Redfish resource collections. If this property is absent, there is no requirement to support deletion of members of this resource.</td>
</tr>
<tr>
<td>UpdateResource</td>
<td>boolean</td>
<td>Specifies a requirement that a user may update a member of this resource. This normally applies to Redfish resource collections. If this property is absent, there is no requirement to support updating of members of this resource, but individual property-level read-write requirements apply.</td>
</tr>
</tbody>
</table>
### Property Types

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>URIs</td>
<td>array</td>
<td>An array of URI references to which the ReadRequirement and WriteRequirement are applied. The values shall follow the resource URI pattern definition specified in the Redfish Specification.</td>
</tr>
<tr>
<td>RequiredResourceProfile</td>
<td>object</td>
<td>Specifies a Redfish interoperability profile file that contains requirements for this resource or resource use case. The ResourceRequirements for this resource type, contained within the specified profile, are applied. All other requirements included in the specified profile, which do not apply to this resource type, are ignored.</td>
</tr>
</tbody>
</table>

### 8.4.1.0.1 URI patterns

As the Redfish Specification version 1.6 or higher defines the set of possible URIs for each resource type, this fact can be used to easily create requirements or conditional requirements for resource types that occur at multiple locations in the resource tree. This method is preferred for any profile that will also desire OpenAPI compatibility, which requires Redfish v1.6 protocol support and therefore the URIs of any conforming implementation will match those listed in the profile.

Profiles which intend to apply to implementations conforming to Redfish Specification versions 1.0 through 1.5 cannot use this URI pattern matching in their profile definition. Profiles containing URI pattern requirements shall require a Redfish Specification version 1.6 or higher in the profile's Protocol object MinVersion property.

### 8.4.1.1 Example

This example shows a simple required schema:

```json
{
  "ComputerSystem": {
    "MinVersion": "1.2.0",
    "Purpose": "Every instance must have a logical-view ComputerSystem resource.",
    "PropertyRequirements": {
      "SerialNumber": {},
      "Manufacturer": {},
      "Model": {
        "ReadRequirement": "Recommended"
      }
    }
  }
}
```
8.4.2 Resource use cases

Some Redfish schemas are re-used for many types of equipment, and also in different parts of the resource tree. The requirements specified in a profile can differ significantly based on a particular resource's usage in the hierarchy. While slight variations in requirements can be handled using conditional requirements, larger variations can instead be expressed with use cases.

The use case resource requirement structure is employed when a particular resource (schema) has two or more significantly different sets of requirements that are dependent on a "type" of that resource, which can be specified using a single property-level comparison, by the URI pattern for those resources, or both.

For example, the Memory schema supports both DRAM (DIMM) and non-volatile (NV-DIMM) memory devices. The management requirements for a NV-DIMM device are considerably different than those for DRAM. By setting up two use cases for the Memory schema, one for "DIMM" and the other for "NV-DIMM" memory, the resulting profile is both easier to construct and easier to comprehend.

The structure for a set of resource use cases follows the resource-level structure, with a single UseCases property in place of the resource-level requirements. Those requirements are specified within each use case:

```json
{
   "<schema-name>": {
      "UseCases": [
         {
            "UseCaseTitle": "<title>",
            "UseCaseKeyProperty": "<property name>",
            "UseCaseKeyValues": [
               "<values of key property for comparison>",
            ],
            "UseCaseComparison": "<type of comparison>",
            "URIs": [
               "<uri-pattern>",
               "<uri-pattern>",
            ],
            "PropertyRequirements": {
               "<Property Requirements>": "<property requirements values>",
            },
            "ActionRequirements": {
               "<Action Requirements>": "<action requirements values>"
            }
         },
         {
            "<Second Use Case>": {}
         },
         {
            "<Third Use Case>": {}
         }
      ]
   }
}
```
8.4.2.1 Use case-level functions

Each use case may include any of the options available as a schema-level function. In addition, the following options are also available at the use case level:

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UseCaseTitle</td>
<td>string</td>
<td>A title or short name used to identify the resource use case.</td>
</tr>
<tr>
<td>UseCaseKeyProperty</td>
<td>string</td>
<td>The name of the property within the resource used by the UseCaseComparison to define the use case.</td>
</tr>
<tr>
<td>UseCaseKeyValues</td>
<td>string</td>
<td>The values of UseCaseKeyProperty used to evaluate the UseCaseComparison.</td>
</tr>
<tr>
<td>UseCaseComparison</td>
<td>string</td>
<td>Specifies the type of comparison to perform using the UseCaseKeyProperty and the values contained in UseCaseKeyValues. If the comparison is successful, this resource instance is covered by this resource use case.</td>
</tr>
</tbody>
</table>

8.4.2.2 Use case example

This example shows a use case selecting requirements by MemoryType for DRAM-based Memory resources under the Systems collection, and non-volatile (NV-DIMM) Memory resources anywhere in the resource tree.

```
{
    "Memory": {
        "UseCases": [
            {
                "UseCaseTitle": "DIMM",
                "UseCaseKeyProperty": "MemoryType",
                "UseCaseComparison": "Equal",
                "UseCaseKeyValues": [
                    "DRAM"
                ],
                "URIs": [
                    "/redfish/v1/Systems/{ComputerSystemsId}/Memory/{MemoryId}
                ],
                "MinVersion": "1.7.0",
            }...
```
8.4.3 Property-level functions

Within the `PropertyRequirements` object are additional objects that are named to match the property name in the parent object's schema definition. This object then contains the property-level requirements, which account for the bulk of a profile's definition. One additional level of JSON objects may be embedded, essentially nesting a `PropertyRequirements` object.

The following options are available at the property level:
<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReadRequirement</td>
<td>string</td>
<td>Property-level requirement for this property. See the Read requirement clause.</td>
</tr>
<tr>
<td>WriteRequirement</td>
<td>string</td>
<td>Property-level write (HTTP PATCH or PUT) requirement for this property. See the Write requirement clause.</td>
</tr>
<tr>
<td>ConditionalRequirements</td>
<td>object</td>
<td>Property-level conditional requirements that apply to instances of this property. See the Conditional Requirements clause.</td>
</tr>
<tr>
<td>MinCount</td>
<td>integer</td>
<td>For array type properties, the minimum number of non-NULL instances within the array.</td>
</tr>
<tr>
<td>MinSupportValues</td>
<td>array</td>
<td>The minimum set of enumerations that must be supported for this writable property.</td>
</tr>
<tr>
<td>Comparison</td>
<td>string</td>
<td>The condition used to compare the value of the property to Values. See the Comparison clause.</td>
</tr>
<tr>
<td>Purpose</td>
<td>string</td>
<td>A description of the purpose of this requirement. This text can provide justification or reasoning behind the requirement for use in the profile documentation.</td>
</tr>
<tr>
<td>Values</td>
<td>array</td>
<td>The values used to perform a Comparison. Multiple values are only allowed for AnyOf or AllOf comparisons. If no Comparison property is present, the comparison is assumed to be an AnyOf comparison.</td>
</tr>
<tr>
<td>ReplacedByProperty</td>
<td>string</td>
<td>A property that fulfills the requirements of the current property, if present in the resource. The value is the name of the property, evaluated at the same object level, or an RFC6901-defined JSON Pointer to the property within the resource. If the specified property is present in the resource, all requirements for the current property are ignored.</td>
</tr>
<tr>
<td>ReplacesProperty</td>
<td>string</td>
<td>A deprecated or obsolete property that this property replaces. The value is the name of the property, evaluated at the same object level, or an RFC6901-defined JSON Pointer to the property within the resource. If the current property is not present, its requirements can be met by the presence of the named property it replaces. If the current property is present, it must meet the requirements regardless of the presence of the property it replaces.</td>
</tr>
<tr>
<td>PropertyRequirements</td>
<td>object</td>
<td>For Redfish object properties, this object contains requirements for the properties contained within the specified object. This specification allows for only one level of nested objects and requirements.</td>
</tr>
</tbody>
</table>

### 8.4.3.1 Example

This example shows property-level requirements, including one object property containing further requirements on that object's properties. For each Power resource, the PowerSupplies and Voltages array properties are required. Voltages has no further requirements. By default, this Voltages is
mandatory, and as an array type, must have at least one item in the array. The `PowerSupplies` array must contain at least two items. Within the array, at least one item's `PowerSupplyType` property must have a value of `AC` or `DC`.

```
{
    "Power": {
        "PropertyRequirements": {
            "PowerSupplies": {
                "ReadRequirement": "Mandatory",
                "MinCount": 2,
                "PropertyRequirements": {
                    "Status": {},
                    "PowerSupplyType": {
                        "Comparison": "AnyOf",
                        "Purpose": "Need to know AC vs. DC to know expected input.",
                        "Values": [
                            "AC",
                            "DC"
                        ]
                    },
                    "LineinputVoltage": {},
                    "PowerCapacityWatts": {},
                    "InputRanges": {
                        "ReadRequirement": "Recommended"
                    }
                }
            },
            "Voltages": {}
        }
    }
}
```

### 8.4.3.2 Comparison

The `Comparison` function uses the following enumerations to represent the various comparisons available:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Absent</td>
<td>The property is not present in this resource.</td>
</tr>
<tr>
<td>AnyOf</td>
<td>At least one instance of the property in this resource must be equal to one of the values listed.</td>
</tr>
<tr>
<td>AllOf</td>
<td>At least one instance of the property in this resource must be equal to each of the values listed.</td>
</tr>
<tr>
<td>Equal</td>
<td>The value of the property must be equal to the value in the profile.</td>
</tr>
</tbody>
</table>
### Value Description

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NotEqual</td>
<td>The value of the property must not be equal to the values listed in the profile.</td>
</tr>
<tr>
<td>GreaterThan</td>
<td>The value of the property must be greater than the values listed in the profile. This comparison is only valid for numeric properties.</td>
</tr>
<tr>
<td>GreaterThanOrEqualTo</td>
<td>The value of the property must be greater than or equal to the values listed in the profile. This comparison is only valid for numeric properties.</td>
</tr>
<tr>
<td>LessThan</td>
<td>The value of the property must be less than the values listed in the profile. This comparison is only valid for numeric properties.</td>
</tr>
<tr>
<td>LessThanOrEqualTo</td>
<td>The value of the property must be less than or equal to the values listed in the profile. This comparison is only valid for numeric properties.</td>
</tr>
<tr>
<td>Present</td>
<td>The property is present in this resource.</td>
</tr>
<tr>
<td>LinkToResource</td>
<td>The object contains a link to a resource with a type equal to one of the schema names listed in the values of the profile. The type is the unversioned schema name, such as <code>Thermal</code> or <code>Memory</code>.</td>
</tr>
</tbody>
</table>

Many of these comparison types are simple arithmetic, boolean, or string value comparisons. In addition, `Absent` and `Present` allow for comparisons concerning the existence or absence of a property. The `LinkToResource` comparison specifies that the object property contains an `@odata.id` property to a resource whose schema name (type) is listed in the `Values` array.

### 8.4.3.3 Read requirement

The `ReadRequirement` function specifies the level of basic read (HTTP GET) requirement applied to the resource or property. The default value, or if no `ReadRequirement` is present, is `Mandatory`. For object properties, requirements of the embedded properties will apply only if the object is present.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mandatory</td>
<td>This property is required in all instances of this resource. For array properties, the property is required in all non-null array items. If <code>Values</code> is defined, at least one instance of each enumeration value is required among instances of this property.</td>
</tr>
<tr>
<td>Supported</td>
<td>This property is required to be supported by the service, but may not appear in all instances of this resource. The requirement is met if the property appears in at least one instance of this resource.</td>
</tr>
<tr>
<td>Recommended</td>
<td>It is recommended, but not required, that this property be supported.</td>
</tr>
<tr>
<td>IfImplemented</td>
<td>This property is required if the underlying functionality is implemented. For object properties, requirements on embedded properties within the object will only apply if the object is present.</td>
</tr>
</tbody>
</table>
### 8.4.3.4 Write requirement

The `WriteRequirement` function specifies the level of write support (HTTP PATCH or PUT) applied to a property. The default value, or if no `WriteRequirement` is present, is `None`. The `WriteRequirement` applies to a property once the `ReadRequirement` is met.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mandatory</td>
<td>This property is required to be writable in all instances of this resource.</td>
</tr>
<tr>
<td>Supported</td>
<td>This property is required to be writable in some instances of this resource. A service meets the requirement if the property is writable in at least one resource instance.</td>
</tr>
<tr>
<td>Recommended</td>
<td>It is recommended, but not required, that this property be writable.</td>
</tr>
<tr>
<td>None</td>
<td>This property is not required to be writable by this profile. It is listed here for clarity, and is the default value used if <code>WriteRequirement</code> is not present.</td>
</tr>
</tbody>
</table>

### 8.4.3.5 Conditional requirements

The most flexible aspect of the Redfish profile definition is the ability to make resource or property-level requirements that are dependent on one or more conditional requirements within the resource and the parent resources in the resource tree.

The `ConditionalRequirements` array function specifies these conditional requirements, which add to any requirements also defined for the resource or property. Note that a condition cannot override or weaken a requirement already specified. For example, if a property requirement is marked as `Mandatory`, no conditional requirement could mark the property as `None`. Instead, the property would be specified with a `None` requirement, and with one or more `ConditionalRequirements` that would specify when the property requirement becomes `Mandatory`.
The following options are available for each conditional requirement:

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReadRequirement</td>
<td>string</td>
<td>The requirement to apply to the resource or property if the condition is met.</td>
</tr>
<tr>
<td>WriteRequirement</td>
<td>string</td>
<td>Property-level write (HTTP PATCH or PUT) requirement for this property. See the Write requirement clause.</td>
</tr>
<tr>
<td>Purpose</td>
<td>string</td>
<td>Text describing the purpose of this conditional requirement.</td>
</tr>
<tr>
<td>URIs</td>
<td>array</td>
<td>An array of URI references to which the ReadRequirement and WriteRequirement is applied. The values shall follow the resource URI pattern definition specified in the Redfish Specification.</td>
</tr>
<tr>
<td>SubordinateToResource</td>
<td>array</td>
<td>An ordered list, from top of hierarchy to bottom, of resources where this resource is linked as a subordinate resource. The conditional requirements listed for the resource apply only to instances which are subordinate to the listed parent resource list. See the Parent and subordinate resources clause.</td>
</tr>
<tr>
<td>Comparison</td>
<td>string</td>
<td>The condition used to compare the value of the property to Values. See the Comparison clause.</td>
</tr>
<tr>
<td>Values</td>
<td>array</td>
<td>The values used to perform a comparison. Multiple values are only allowed for AnyOf or AllOf comparisons. If no Comparison property is present, the comparison is assumed to be an AnyOf comparison.</td>
</tr>
<tr>
<td>CompareProperty</td>
<td>string</td>
<td>The name or path to the property in this resource whose value is used to test this condition. If the value begins with a / character, the value shall represent an RFC6901-defined JSON Pointer, specifying an explicit path from the root level of the resource to a property within the resource. Otherwise, the property name will be evaluated at the current object level within the resource, and if it is not found, upper levels will be searched until the root level is reached. See the Compare property clause.</td>
</tr>
<tr>
<td>CompareValues</td>
<td>array</td>
<td>Values of the CompareProperty used to test this condition. See the Compare property clause.</td>
</tr>
<tr>
<td>CompareType</td>
<td>string</td>
<td>The condition used to compare the value of the property named by CompareProperty to the value of CompareValues. If the comparison is true, this conditional requirement applies. See the Compare property clause.</td>
</tr>
</tbody>
</table>

### 8.4.3.5.1 Parent and subordinate resources

Because there can be several instances of a particular Redfish schema in the resource tree, the requirements placed on those resources may vary depending on their usage. Since the profile is schema-centric, the SubordinateToResource function allows a profile to specify requirements based a resource instance's placement in the resource tree.
resource to which the requirements apply. This property contains an array of schema names, in the
top-down order that they appear in the path to the required resource.

Note that this functionality may also be accomplished using the `uris` function for the resource, which
is the preferred method.

### 8.4.3.5.2 Example

For the property `HostName` in the `EthernetInterface` schema, the example shows it as `Recommended`
property. But if an instance of `EthernetInterface` is linked from a `ComputerSystem` resource, through the
`EthernetInterfaceCollection` resource, the `Condition` is met, which changes the `HostName` property
requirement to `Mandatory`.

In the second part of the example, the `IPv6Addresses` array property is required to have at least one
item ( `MinCount` ) in the array. But if, as in the previous example, the instance is subordinate to a
`ComputerSystem` (and `EthernetInterfaceCollection`) resource, at least two items are required in the
array.

```json
{
   "EthernetInterface": {
      "PropertyRequirements": {
         "HostName": {
            "ReadRequirement": "Recommended",
            "WriteRequirement": "Recommended",
            "ConditionalRequirements": [
               {
                  "SubordinateToResource": [
                     "ComputerSystem",
                     "EthernetInterfaceCollection"
                  ],
                  "ReadRequirement": "Mandatory",
                  "Purpose": "Used to match this instance to other data sources."
               }
            ]
         }
      },
      "IPv6Addresses": {
         "ReadRequirement": "Mandatory",
         "MinCount": 1,
         "ConditionalRequirements": [
            {
               "SubordinateToResource": [
                  "ComputerSystem",
                  "EthernetInterfaceCollection"
               ],
               "MinCount": 2
            }
         ]
      }
   }
}```
8.4.3.5.3 Compare property

A typical need for a conditional requirement is a dependency on the value of another property within the resource. This type of dependency can be used when several different product variations share a common schema definition. In that case, Redfish schemas normally define a type-specifying property with enumerations, for a variety of product categories, that can be used to differentiate profile requirements by product category.

To accomplish this, there are three Profile properties related to this function:

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CompareProperty</td>
<td>string</td>
<td>The name or path to the property in this resource whose value is used to test this condition. If the value begins with a / character, the value shall represent an RFC6901-defined JSON Pointer, specifying an explicit path from the root level of the resource to a property within the resource. Otherwise, the property name will be evaluated at the current object level within the resource, and if it is not found, upper levels will be searched until the root level is reached. See the Compare property clause.</td>
</tr>
<tr>
<td>CompareType</td>
<td>string</td>
<td>The condition used to compare the value of the property named by CompareProperty to the value of CompareValues. If the comparison is true, this conditional requirement applies.</td>
</tr>
<tr>
<td>CompareValues</td>
<td>array</td>
<td>Values of the CompareProperty used to test this condition.</td>
</tr>
</tbody>
</table>

8.4.3.5.4 Examples

Simple dependencies can be expressed using the conditional requirement and a comparison. This example shows a CompareProperty condition applied to the Pepperoni property. If the PizzaType property is not equal to Cheese, then the Pepperoni property becomes both mandatory and must have a value of true.

```json
{
    "Pepperoni": {
        "ReadRequirement": "Recommended",
        "ConditionalRequirements": [
            {
                "Purpose": "Pepperoni is required on all pizza types except Cheese."
            }
        ]
    }
}
```
This example shows a `CompareProperty` condition applied to the `IndicatorLED` property, which has a base `Recommended` requirement, but becomes `Mandatory` if the `SystemType` property has a value of `Physical` or `Composed`.

```json
{
  "IndicatorLED": {
    "ReadRequirement": "Recommended",
    "ConditionalRequirements": [
      {
        "Purpose": "Physical and composed systems must have a writable LED",
        "CompareProperty": "SystemType",
        "CompareType": "AnyOf",
        "CompareValues": [
          "Physical",
          "Composed"
        ],
        "ReadRequirement": "Mandatory",
        "WriteRequirement": "Mandatory"
      }
    ]
  }
}
```

This example shows a `CompareProperty` condition applied to the `SerialNumber` property, which has a `Conditional` requirement, becoming `Mandatory` only in cases where the `/Location/PartLocation/LocationType` property (specified as a JSON Pointer per RFC6901) has a value that is not `Embedded`.

```json
{
  "SerialNumber": {
    
  }
}
```
8.4.3.6 Handling deprecated properties

As the Redfish data model evolves, there are many cases of properties being deprecated in favor of an improved property that replaces it. While it would be preferred for a profile to specify the new property, existing service implementations could not meet these requirements. Furthermore, it may take substantial time for broad adoption of the replacement property.

Rather than creating property-level requirements for both the original and the replacement property, the ReplacesProperty and ReplacedByProperty functions allow a profile to specify requirements where either the original or replacement property can fulfill those requirement, without requiring "legacy" support of deprecated properties.

For example, the widely-implemented IndicatorLED property, which suffered from interoperability issues due to the different LED states implemented by vendors, was replaced by the LocationIndicatorActive property throughout the Redfish data model. To encourage vendors to implement the replacement property, an entry for LocationIndicatorActive is added to the profile. But as existing implementations will only support the deprecated IndicatorLED, it is referenced using the ReplacesProperty function. That allows an implementation to meet the requirements by supporting either property, but not placing any requirement that it supports both the deprecated property and its replacement.

8.4.3.6.1 Examples

This example shows the LocationIndicatorActive property requirements, which can be met if the service implements the deprecated IndicatorLED property. But if the service implements the replacement property, it has no profile requirement to carry the deprecated, "legacy" property.
This example follows the previous one, but retains specific requirements on the deprecated `IndicatorLED` property. If the service implements only the deprecated `IndicatorLED` property, it will meet both of these property requirements, as long as it supports setting the `IndicatorLED` property to the "Off", "Lit", and "Blinking" values. If the service implements the `LocationIndicatorActive` property, the requirements on `IndicatorLED` are ignored.

```json
{
    "PropertyRequirements": {
        "LocationIndicatorActive": {
            "ReadRequirement": "Mandatory",
            "WriteRequirement": "Mandatory",
            "ReplacesProperty": "IndicatorLED"
        },
        "IndicatorLED": {
            "ReadRequirement": "Mandatory",
            "WriteRequirement": "Mandatory",
            "ReplacedByProperty": "LocationIndicatorActive",
            "MinSupportValues": [
                "Off",
                "Lit",
                "Blinking"
            ]
        }
    }
}
```

### 8.4.4 Action requirements

Because several critical functions of a Redfish service are implemented as `Actions`, the profile may place requirements for support of these Actions. The requirements can specify which parameters must be supported, and may specify allowable values for those parameters.

The following functions are available to specify requirements for an action within a resource requirement:
### 8.4.4.1 Parameters

The following functions are available to specify requirements for a parameter on a particular action:

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReadRequirement</td>
<td>string</td>
<td>The requirement to apply to this action.</td>
</tr>
<tr>
<td>Parameters</td>
<td>object</td>
<td>The requirements for any parameter available for this action.</td>
</tr>
<tr>
<td>Purpose</td>
<td>string</td>
<td>A description of the purpose of this requirement. This text can provide justification or reasoning behind the requirement for use in the profile documentation.</td>
</tr>
</tbody>
</table>

### 8.4.4.2 Example

This example shows the **Reset** action as required for this resource, along with the required parameter **ResetType**, which must support the values of **ForceOff** and **PowerCycle**.

```json
{
    "ActionRequirements": {
        "Reset": {
            "ReadRequirement": "Mandatory",
            "Purpose": "Ability to reset the unit is a core requirement of most users.",
            "Parameters": {
                "ResetType": {
                    "ParameterValues": [
                        "ForceOff",
                        "PowerCycle",
                        "On"
                    ],
                    "RecommendedValues": [
                        "GracefulShutdown",
                        "GracefulRestart",
                        "ForceRestart"
                    ]
                }
            }
        }
    }
}
```
8.5 Registry-level requirements

While not normally part of the JSON resources, the Redfish-defined message registries are important for interoperability, as they indicate what functionality has been implemented for events, and are also a useful method for setting expectations on the use of extended info error messages when interacting with a Redfish service.

The following functions are available to specify registry-level requirements:

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Repository</td>
<td>string</td>
<td>A URI providing the location of the repository which contains the JSON files to be included. The filenames of the JSON files contained in the repository are expected to follow the Redfish message registry filename conventions. If absent, the repository location shall be the Redfish registry repository (<a href="http://redfish.dmtf.org/registries">http://redfish.dmtf.org/registries</a>).</td>
</tr>
<tr>
<td>MinVersion</td>
<td>string</td>
<td>The minimum version required by this Redfish profile. The version shall be represented using a <code>&lt;major&gt;.&lt;minor&gt;.&lt;errata&gt;</code> format, including an optional errata version. If this property is absent, the minimum value shall be 1.0.0.</td>
</tr>
<tr>
<td>ReadRequirement</td>
<td>string</td>
<td>Resource-level requirement for this registry. See the Read requirement clause.</td>
</tr>
<tr>
<td>Purpose</td>
<td>string</td>
<td>A description of the purpose of this registry. This text can provide justification or reasoning behind the requirement for use in the profile documentation.</td>
</tr>
<tr>
<td>Messages</td>
<td>object</td>
<td>The messages in this registry that have support requirements for this Redfish profile. If this property is absent, all messages in this registry follow the registry-level ReadRequirement.</td>
</tr>
<tr>
<td>SupportedFeatures</td>
<td>object</td>
<td>The features in this registry that have support requirements for this Redfish profile. If this property is absent, all supported features in this registry follow the registry-level ReadRequirement.</td>
</tr>
</tbody>
</table>
8.5.1 Messages

Within the registry object are additional objects that are named to match the message name in the registry definition. This object then contains the message-level requirements.

The following options are available at the message-level:

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReadRequirement</td>
<td>string</td>
<td>Message-level requirement for this message. See the Read requirement clause.</td>
</tr>
</tbody>
</table>

8.5.2 Example

This example shows requirements for two message registries, including one OEM-defined registry. The Base registry is a DMTF standard registry. By default since no OwningEntity is listed, and therefore can be retrieved by default from DMTF's repository. The Base registry lists only four messages that are required.

In the case of the OEM-defined registry ContosoPizzaMessages, the Mandatory requirement set at the registry level specifies that all messages defined in that registry are required.

```json
{
   "Registries": {
      "Base": {
         "MinVersion": "1.1.0",
         "Messages": {
            "Success": {},
            "GeneralError": {},
            "Created": {},
            "PropertyDuplicate": {}
         }
      },
      "ContosoPizzaMessages": {
         "OwningEntity": "Other",
         "OwningEntityName": "Contoso",
         "Repository": "http://contoso.com/registries",
         "ReadRequirement": "Mandatory"
      }
   }
}
```
8.5.3 Supported features

Within the registry object are additional objects that are named to match the SupportedFeatures name in the registry definition. This object then contains the features-level requirements.

The following options are available at the feature-level:

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReadRequirement</td>
<td>string</td>
<td>Feature-level requirement for this supported feature. See the Read requirement clause.</td>
</tr>
</tbody>
</table>

8.5.4 Example

This example shows requirements for a feature registry. The SwordfishFeatures registry is a DMTF-partner defined standard registry, specified as such where the OwningEntity is SNIA. SNIA re-publishes their registries through DMTF and therefore this registry can be retrieved from DMTF's repository. In this example, the SwordfishFeatures registry lists two features that are supported by the service.

```json
{
    "Registries": {
        "SwordfishFeatures": {
            "MinVersion": "1.0.2",
            "SupportedFeatures": {
                "SNIA.Swordfish.Discovery": {},
                "SNIA.Swordfish.EventNotification": {}
            }
        }
    }
}
```
## 9 ANNEX A (informative) Change log

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.6.0</td>
<td>2022-12-07</td>
<td>Added support for resource use cases.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added RequiredResourceProfile to the <strong>Schema-level functions</strong> section to allow inclusion of individual schema-level requirements from other profiles.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added ReplacesProperty and ReplacedByProperty to the <strong>Property-level functions</strong> section to provide better handling of requirements on deprecated or obsolete properties.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added guidance for handle deprecating properties.</td>
</tr>
<tr>
<td>1.5.0</td>
<td>2021-12-02</td>
<td>Added support for JSON Pointer usage in CompareProperty.</td>
</tr>
<tr>
<td>1.4.1</td>
<td>2021-10-06</td>
<td>Corrected descriptions for CreateResource, DeleteResource, and UpdateResource to match terminology and usage in the sample profiles.</td>
</tr>
<tr>
<td>1.4.0</td>
<td>2021-09-15</td>
<td>Made many changes for style consistency, grammar, and general clarity. Except for the following additions, no normative changes were made. Any clarifications that inadvertently altered the normative behavior are considered errata, and will be corrected in future revisions to the specification.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added License and ContributedBy.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added missing table entries for Protocol, Resources, and Registries.</td>
</tr>
<tr>
<td>1.3.0</td>
<td>2019-12-06</td>
<td>Added SupportedFeatures to registry-level information, and added an overview and example of SupportedFeatures.</td>
</tr>
<tr>
<td>1.2.0</td>
<td>2019-07-03</td>
<td>Added IfPopulated enumeration value to ReadRequirement to indicate to conformance test tools that a required property is not required if the underlying hardware isn't populated, or that a required resource may not be populated under test conditions.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added Supported enumeration value to ReadRequirement and WriteRequirement to allow for required properties that are supported by some, but perhaps not all, instances of a resource.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added missing pattern term for version properties.</td>
</tr>
<tr>
<td>1.1.0</td>
<td>2019-02-26</td>
<td>Added support for new protocol features from Redfish Specification v1.6. Added ability to make requirements based on URI patterns as specified in Redfish schema files. Updated normative references to current versions. Clarified that Repository value may indicate a profile or schema file location, as appropriate. Formatting improvements.</td>
</tr>
<tr>
<td>Version</td>
<td>Date</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>1.0.1</td>
<td>2018-05-15</td>
<td>Errata release. Corrected definition of <code>Comparison</code> for conditional requirements to match the schema usage (and consistent with other usage). Added missing <code>Values</code> property for conditional requirements and added new <code>CompareType</code> property to replace the inconsistent usage of <code>Comparison</code>. Added example for a conditional requirement that uses the <code>Values</code> array.</td>
</tr>
<tr>
<td>1.0.0</td>
<td>2018-01-02</td>
<td>Initial release.</td>
</tr>
</tbody>
</table>