Document Number: DSP2043

Date: 2015-03-03

Version: 0.95.0a

Scalable Platforms Management API Mockup Readme

Information for Work-in-Progress version:

IMPORTANT: This document is not a standard. It does not necessarily reflect the views of the DMTF or all of its members. Because this document is a Work in Progress, it may still change, perhaps profoundly. This document is available for public review and comment until the stated expiration date.

It expires on: 2015-08-24

Provide any comments through the DMTF Feedback Portal: http://www.dmtf.org/standards/feedback

Document Type: Readme

Document Status: Work in Progress

Document Language: en-US

CONTENTS

Foreword

The following files are part of the Redfish Scalable Platforms Management API ("Redfish") development effort

- Specification.md - this files is the main Redfish Scalable Platforms Management API Specification
- Whitepaper.md - this is intended to be a non-normative document helping those new to Redfish understand how to interact with the Redfish service and understand common functions and tasks.
- Readme2043.md - this document.
- Readme8010.md - Schema Readme

These other components are part of the Redfish Scalable Platforms Management API development effort

- Mockup (DSP2043) - this is a mockup that can be used as sample of output from GETs from A Redfish service. Informative in nature, it was used to develop the schema. A person can set up an NGINX or similar server and configure it to output JSON format and then use this directory for demonstration purposes.
- Schema - this contains the Redfish Schema definitions. These files are normative in nature and are normatively reference by the Redfish Specification. There are two Schema formats - CSDL (OData Common Schema Definition Language format which is in XML) and JSON Schema. These Schema defintions should be functionally equivalent, thus specifying the schema in two different languages.

1. Setting up and accessing the Mockup

You can view the markup either directly or through a webserver.

1.1. Directly

If you are viewing the files directly either through GitHub or a browser, you can get the html files to load in your browser. If you are viewing it in your browser, then make sure you have a JSON viewer loaded as it will help make the mockup more user friendly.

1.2. Via Webserver

Install NodeJS first; this will act as the webserver. Double click `run.bat` inside the Mockup folder to start a local server accessible at http://localhost:9080/redfish/v1. It will look almost like a real service from the GET perspective (the headers will not be the same).

Keep in mind that this is a mockup, not a prototype. But you can do GETs on it and see JSON so it's as real on the READ side as any prototype (except for ETags and some of the headers). If you get a plug-in for Chrome or Opera that does REST (there are some free ones out there) or a JSON decoder for pages, you will improve the experience. The plug-ins let expand and collapse structures making it very easy to interact with.

2. Concepts

Every URI represents a resource. This could be a service, a collection, an element or some other construct. But in RESTful terms, these URIs point to resources and clients interact with Resources. So when you see the term resource, you can think of it as what you get back when you access a URI.

The resource format is defined by a Schema. Each resource has a specific format that is specified in the Redfish Schema that the client can use to determine the semantics about the resource (though we try to make things as intuitive as possible). The Schema is defined in OData's Schema format.

All properties in the resource are intended to be used as JavaScript variables. This should accelerate adoption and allow JavaScript web pages and enabled apps to use the data directly. URIs are persistent across reboots but clients are expected to start at /redfish/v1 and do discovery of the URIs from there. This is known as a "hypermedia API" approach. Don't fixate on the URIs as URIs can be different between implementations. Current state objects can be separate from desired state objects. The section below works better if you are actually doing the GETs

2.1. Starting:

All clients start at the base /redfish/v1 object. Many items are broken down in arrays of references for scalable environments. Links to other resources are in the "links" section. You can see links to Systems, managers, the physical Chassis as well as services like Eventing, Tasks, Schema (meta data)). (Note – discovery of service endpoints will be done using UPNP's SSDP but that's not in the mockup).

2.2. Versioning:

Redfish has two kinds of versioning - the version of the protocol and the version of the resource schema. The version of the protocol is in the URI - that's why you should start at /redfish/v1. It means you are accessing version one of the protocol. Version 1 is the only one available now, but we needed to accommodate potential future versions.

Each resource has a resource type definition. Resource types are defined in versioned namespaces. Each resource instance has the type represented using the OData type annotation "@odata.type". The value of the type annotation is the URI of the resourcce type, including the versioned namespace. So when you see "@odata.type" : "#ServiceRoot.1.0.0.ServiceRoot", you are dealing with a resource that adheres to the ServiceRoot type definition, defined in 1.0.0 version of the ServiceRoot schema. The corresponding schema file would be located at /schema/v1/ServiceRoot in the Redfish schema repository. So the full URI for the type would be "/schem/v1/ServiceRoot#ServiceRoot.1.0.0.ServiceRoot. The schema file may contain other types used by in the resource type definition (for example, structured types and enums), which would have the same resource path but the fragment would describe a different type definition, typically within the same namespace.

2.3. References

When you see the links section, it will have a set of references to other resources in it. URIs are either absolute or relative. Absolute ones won't have the IP address but will start with /redfish/v1. If you have a plug in like the Chrome Advanced REST client, you can click on this to fill in the URI for your next GET.

2.4. Main Objects

The "main" objects are Systems, Managers and Chassis. These are all collections (see next heading). We will dig into these resources in a minute, but it's good to know a bit about them. Systems can have one or more managers (since some managers are redundant), and are in one chassis. Managers are in a Chassis and can manage more than one system. And Chassis can house more than one System and/or Managers. Chassis can also have Chassis in them. It is the chassis that houses sensors, fans and the like. Systems have the CPU/Memory complex and devices and managers handle various management services but also have their own devices (like NICs). This is just an overview, but looking at the base object you can see this right away.

2.5. Collections

There are groups of similar resources returned as collections. Examples of these in the Mockup include Systems, Managers, Chassis, LogEntries, Sessions, EventSubscriptions and more.

So pick either Systems, Managers or Chassis and go down into it. You will see this is a Collection. Collections may be paginated; collections with a property named "@odata.nextLink" are incomplete, and the client can use the URL-value of the property to retrieve the next portion of the collection from the service.

Collection responses have a "value" property that contains a list of members, or links to members. Links to members are represented as JSON objects with a single "@odata.id" property containing the URI of the related resource.

2.6. Current Configurations vs Settings

There are basically two kinds of objects in Redfish - Current Configurations and Settings. Most objects represent the current state of any given resource. Occasionally, you'll see a property called "Settings" in the links section for a resource. This link tells you where to do PUTs and PATCHes. It represents the future state of the resource. Some resources can handle changes to them right away, others may require a restart/reboot of the system or service. "Settings" is used to let the client know what type of resource this is and where to make the changes. If you see a Settings link, that's where to make the changes that will be picked up at the next reboot. Examples of resources that need Settings are devices like NICs and Storage as well as BIOS.

2.7. Common Properties

As you go through the model, you keep seeing some of the same properties. You'll find Name and Modified in every resource. These are required. You'll also

see Status as an embedded object and it has the same definition across all usages. All of these are actually in a common part of the Schema and used by other Schema by reference.

2.8. Actions

Not everything can be done easily using REST, so Redfish leverages OData Actions for procedural operations. Things like "push button" on a System (which would reset the system or turn it off, depending on its setting) can't easily be represented in the System because the service has no idea what the state of the button is.

So instead of having hidden properties that you could PUT/PATCH to, or complex state machines, we created Actions. Actions are done with POSTs to the Resource (see the spec for specifics). You can tell what actions are supported in any resource by looking for the "Actions" property.

2.9. CorrelatableIDs

Go back to one of the Chassis and take a look at the fans by following the link to "ThermalMetrics" and you will see how Redfish shows redundancy. You will notice each fan has a property called “CorrelatableID”. Now look at the fan redundancy set. It shows the two fans in its set using the same values in the CorrelatableID properties. Thus the client can figure out which belongs to which.

You can't really have hrefs that point inside of objects, though JSON Pointer can do this. But not all clients can understand JSON pointer. So how do you do fan redundancy without hrefs? The way we did it is with CorrelatableIDs - an opaque string that is the same for like items. So the Fans all have IDs and the RedundancySet has an ID and the fans can point to the RedundancySet's ID and vice versa.

The manager isn't expected to directly provide all of the data in Redfish, given the advent of PMCI. Thus providers have no knowledge of URIs to other providers. So how do you correlate a PCI slot with a NIC? The Redfish way for providers that know nothing about each other is to come up with a CorrelatableID that they can all figure out on their own. Take UEFI device path - every PCI device knows its BDF and can figure out its UEFI device path. Thus it makes an excellent CorrelatableID for the Client to be able to figure out the PCI settings for the NIC (in this case).

Thus everything that has the same Correlatable ID in objects that all refer to different views or aspects of the same device or thing.

2.10. ETags

ETags are used by browsers to optimize IO (caching). They do an If-Match and if it matches, they don’t bother dragging the data. We use them to determine if an object has changed. Thus every ETag will change if a PUT is done or if a tool (like BIOS configuration at the console) takes place. Thus any race condition on PUTs between Redfish and non-Redfish clients (as well as other Redfish clients) are always noticed.

The problem is that this is a mock-up and not a real web service so you can't see the ETags work.

2.11. Services

Let's go back to root and take a look at some common services. Tasks, Sessions, EventService and AccountService are all common services. You can read more about them in the spec, but they should be fairly obvious. Tasks contains a list of jobs that may have been started, usually as the result of Actions.

2.12. Where does a client get schema definition?

Payloads contain a URL that the client can use to retrieve the schema for the service. That service schema file generally references other externally hosted schema files for common schema definitions. Types within a JSON payload are identified by a dereferenceable type URI.

3. Enough for now

Hopefully the information will have been enough for now to get you up to speed and you can start perusing the mockup and reading the spec as well as the schema to get an idea of how all of this works.