Redfish Overview & Deep Dives

Jeff Autor
Redfish Forum Co-Chair
Distinguished Technologist,
Hewlett Packard Enterprise

Disclaimer

- The information in this presentation represents a snapshot of work in progress within the DMTF.
- This information is subject to change without notice. The standard specifications remain the normative reference for all information.
- For additional information, see the DMTF website.

A Hybrid IT Management Solution

Design Tenets

- Leverage common Internet / Web Services standards, other standards where appropriate
- Represent modern hardware designs (standalone to scale-out, current silicon, OCP)
- Does not require a PhD to design or use.
- Separation of protocol from data model, allowing them to be revised independently

Protocol Suite

- HTTPS / SSL: Primary data transport
- SSDP from uPnP: Service Discovery
- HTTP-based alert subscription
- Leverage OData v4

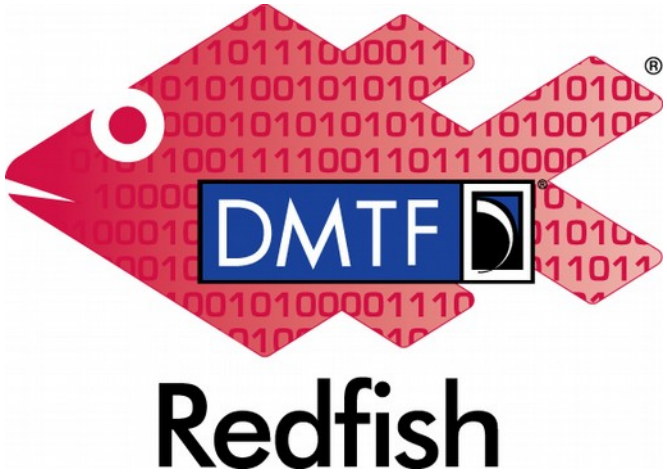
REST & JSON

- Modern, standards-based
- Widely used for web services, software defined and public APIs
- Easy for IT professionals and amateurs to utilize

Data Model

- Schema-based, starting with CSDL & JSON Schema
- Prepare to add schema language definitions as market changes
- An easy to use data model that a human can just read
- Create new modeling tenants to facilitate ease of design (inheritance by copy, polymorphism by union)

What is Redfish?



- **Industry Standard Software Defined Management for Converged, Hybrid IT**
  - HTTPS in JSON format based on OData v4
  - Schema-backed but human-readable
  - Equally usable by Apps, GUIs and Scripts
  - Extensible, Secure, Interoperable
- **Version 1 focused on Servers**
  - A secure, multi-node capable replacement for IPMI-over-LAN
  - Represent full server category: Rackmount, Blades, HPC, Racks, Future
  - Intended to meet OCP Remote Machine Management requirement
- **Expand scope over time to rest of IT infrastructure**
  - Additional features coming out approximately every 4 months
  - Working with SNIA to cover more advanced Storage (Swordfish)
  - Working with The Green Grid & ASHRAE to cover Facilities (Power/Cooling)
  - Work with the IETF to cover some level of Ethernet Switching

DMTF Redfish Forum

Co-Chairs: Jeff Autor (HPE), Mike Raineri (Dell)

Redfish Forum Leadership Companies

BROADCOM, CISCO, DELL, ERICSSON, Hewlett Packard Enterprise, intel, Lenovo, SUPERMICRO, VERTIV, vmware

Redfish Supporting Companies

American Megatrends Inc, ARM Inc, Artesyn Embedded Technologies, Cray Inc., Eaton, Fujitsu, Google, IBM, Insyde Software Corp, Mellanox Technologies, Microchip, NetApp, New H3C Tech., OSIsoft LLC, Quanta Computer, Solarflare Communications, Toshiba, Western Digital Corporation

Redfish Industry Alliance Partners & efforts

OCP (Open Compute Project) – Collaborating on profile definition
UEFI – Collaborating on Firmware Update and Host Interface work
SNIA – Collaborating on Storage modeling / alignment between SNIA SSM and Redfish
TGG – Pursuing relationship to work on Power/Cooling (existing DMTF Alliance Partner)
IETF – working on Switch modeling (no official alliance)
ASHRAE – American Society of Heating, Refrigerating and Air Conditioning Engineers
BBF – Broadband Forum
Gen-Z – Gen-Z Consortium
PICMG – Open Modular Computing for IIoT
NVMe – NVMe-MI

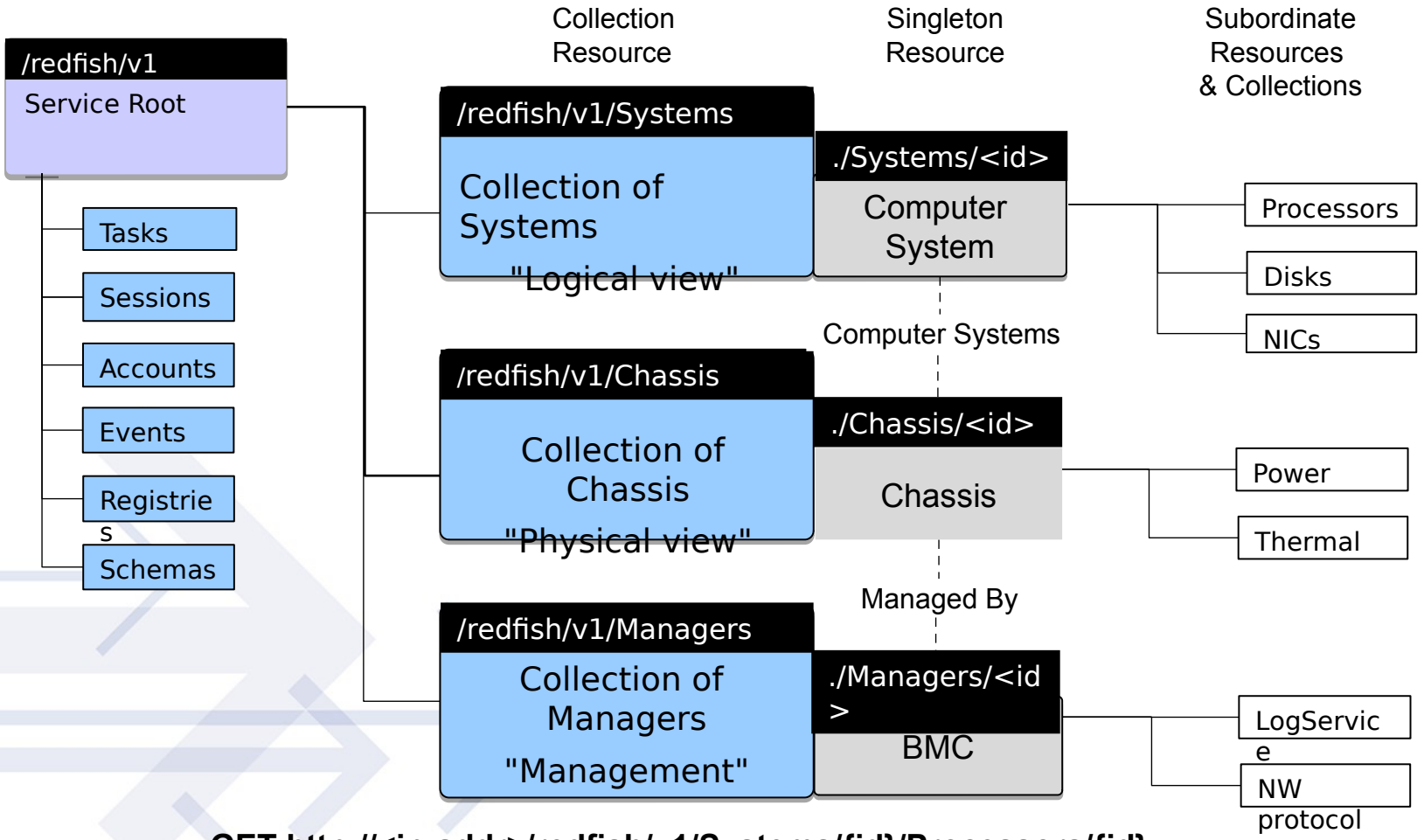
Timeline of Redfish® Specification

- The DMTF Redfish technology
  - Sep 2014: SPMF Formed in DMTF.
  - Aug 2015: Redfish Specification with base models (v1.0)
  - 2016.1: Models for BIOS, disk drives, memory, storage, volume (2016.1)
  - 2016.2: Models for endpoint, fabric, switch, PCIe device, zone, software/firmware inventory & update (2016.2)
  - 2016.3: Adv. communications devices (multi-function NICs), host interface (KCS replacement), privilege mapping (2016.3)
  - 2017.1: Composability (2017.1), WIP for Telemetry
  - 2017.2: Location, errata (2017.2), WIPs for Ethernet Switching, DCIM, OCP & Profiles
  - 2017.3: Profiles, Query parameters, errata (2017.3)
  - 2018.1: LDAP/AD, SSE, Assembly, minor enhancements & errata
  - 2018.2: **OpenAPI**, Telemetry, Jobs, Schedule, Compose II, Message II
  - 2018.3: Certificates, Sensor II (DCIM), FPGA
  - 2019.1: Spec Clean up; Additions to Certs, Password Management
  - 2019.2: Schema Description clean up; standard multipart Push Updates
- Alignment with other standard organizations
  - Aug 2016: SNIA releases first model for network storage services (Swordfish)
  - Working open YANG Redfish mapping algorithm for Ethernet Switch
  - DMTF created work registers with UEFI, TGG, OCP, ASHRAE, Broadband Forum, ETSI-NFV, NVMe, PICMG, GenZ, ODCC for work on applying Redfish

Redfish Artifacts – Excluding WIPs

- Specs
  - DSP0266 - Redfish Specification
  - DSP0272 - Redfish Interoperability Profiles Specification
  - DSP0270 - Redfish Host Interface Specification
  - DSP0272 - Schema Guide
- Bundles
  - DSP8013 - Redfish Interoperability Profiles Bundle
  - DSP8011 - Redfish Standard Registries
  - DSP8010 - Redfish Schema
- Presentations
  - Dozens of Presentations: Overviews, Schema, Data Model, How to Model,
- White Papers
  - Tech Note - Redfish Tech Note
  - DSP2052 - Redfish and OData White Paper
  - DSP2050 - Redfish Composability White Paper
  - DSP2047 - Redfish for Networking White Paper
  - DSP2046 - Redfish Resource and Schema Guide
  - DSP2045 - Redfish Frequently asked Questions (FAQ)
  - DSP2044 - Redfish White Pape
  - DSP2043 - Redfish Scalable Platforms Management API Mockups

Redfish Resource Map (simplified)

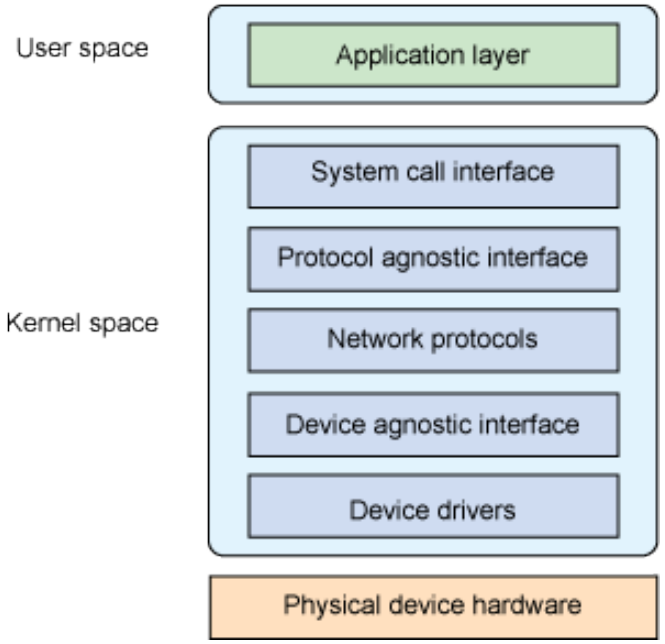


GET http://<ip-addr>/redfish/v1/Systems/{id}/Processors/{id}

Use the Redfish Resource Explorer (redfish.dmtf.org) to explore the resource map

Host Interface

- Replacement for IPMI KCS, etc.
- Exposes a NIC from Management Controller to OS
  - SMBIOS records provide information to allow kernel access
- Same access in-band as out-of-band
  - Kernel mode or user mode accessible
  - Encouraging OS vendors to begin consuming Redfish data.
  - This means you can get to the iLO homepage from the OS.
  - This means you can write your tools for the iLO homepage or Redfish and run them in the host OS.
  - Anything that accesses the out of band can be run on the host OS to access to local management subsystem.



Interoperability Profiles: Goals

- An “Interoperability Profile” provides a common ground for Service implementers, client software developers, and users
  - A profile would apply to a particular category or class of product (e.g. “Front-end web server”, “NAS”, “Enterprise-class database server”)
  - It specifies Redfish implementation requirements, but **is not** intended to mandate underlying hardware/software features of a product
  - Provides a target for implementers to meet customer requirements
  - Provide baseline expectations for client software developers utilizing Redfish
  - Enable customers to easily specify Redfish functionality / conformance in RFQs
- Create a machine-readable Profile definition
  - Document must be human-readable
  - Can be created by dev/ops personnel and non-CS professionals
- Enable authoring of Profiles by DMTF, partner organizations, and others
- Create open source tools to document and test conformance

Known Open Source around Redfish

DMTF

- Largely a suite validation software, conformance and client tools
- Open Compute Project (OCP)
- OpenBMC & OpenRMC
- Vertical integrations as part of the Redfish Forum's charter, the group works with various open source communities for assisting in the adoption of Redfish
  - Three projects are called out in the charter specifically
    - OpenStack
    - Ansible
    - Puppet
  - The Redfish Forum has been engaged with these groups for the past year, either by providing technical assistance, or contributing directly to their projects

Education

Education: Presentations

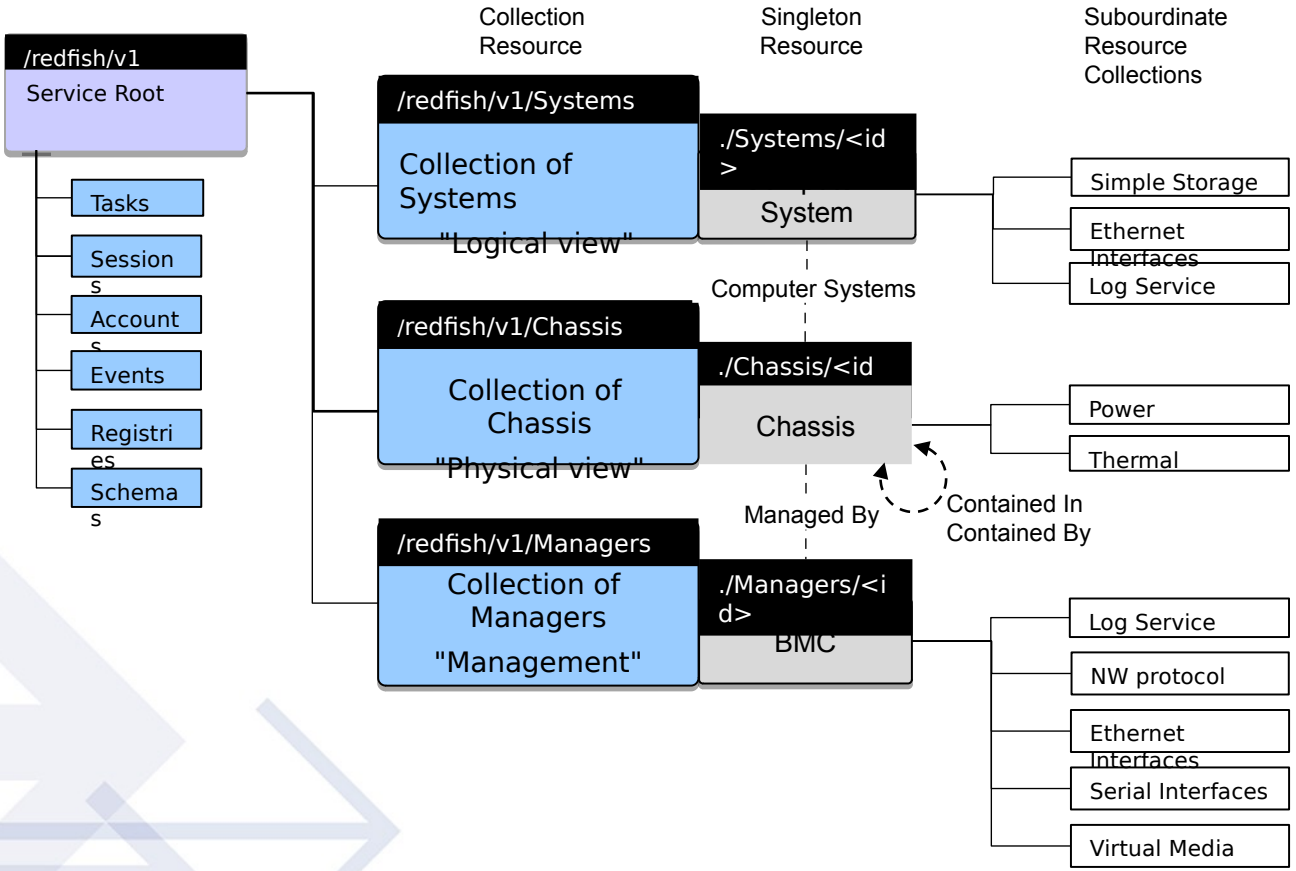
- Redfish School - Sessions
- Redfish School - Redfish CSDL Usage
- Redfish School - Introduction to CSDL
- Redfish School - Redfish Systems - Sept 2016
- Redfish School – Events 1
- Redfish School – Events 2
- Redfish School – Tasks
- Redfish School - Composability 2017
- Redfish School - Storage Modeling
- Redfish School - OData Client Support in Redfish
- Redfish School - Architecture 2016
- Redfish 2019.1 Release Presentation WIP
- End-to-End Interoperable Management: The Standards Requirement for Modern IT
- Redfish 2018 Release 3 Overview
- Redfish Forum Open Source Community Engagements 2018 (WIP)
- Redfish Tech Note - Simple and Secure Management for Converged, Hybrid IT
- Redfish 2018 Release 2 Overview
- Redfish Release History
- Redfish DCIM Work-in-Progress
- Redfish & RDE for Storage
- Redfish OpenAPI Support Presentation 2018Q3 (WIP)
- OCP Summit 2018 DMTF Standards for OCP Platforms Management
- Redfish Interoperability Profiles v1.0
- Redfish Introduction and Overview
- Redfish Technical Overview
- Managing Network Devices with Redfish & Yang - CNSM 2017 Keynote
- Redfish School - Advanced Communication Devices (ACD)
- UCC Redfish DCIM Modeling Presentation
- The Case for Redfish 2017-05 (WIP)
- Redfish Basic Server Interoperability Profile (WIP)
- Managing Network Infrastructure via Redfish v2 (WIP)
- YouTube Series: Redfish School
- D2 T1 S6 Managing Servers with Redfish (WIP)
- Webinar: Redfish Overview
- Webinar: Redfish Data Model Deep Dive

Unreleased Information – DMTF Confidential

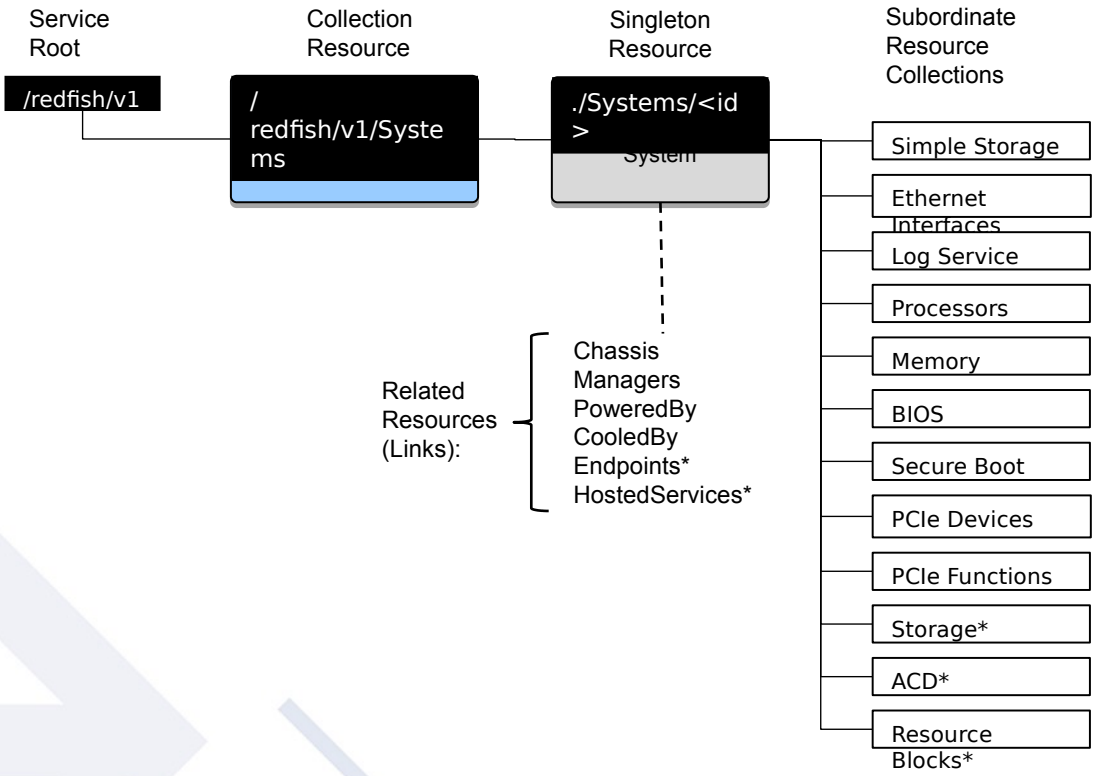
- Specifications in various states:
  - Schema Supplement
  - NVMe-MI Mapping Document
  - Ethernet Switching
- White Papers never done:
  - BIOS White Paper, Firmware Update, Boot Config, OEM Extensions, Fabrics, Sensors and Alarms, Jobs, Privilege Map, DCIM
- YouTubes never developed:
  - Proc & Memory, Users & Priv, BIOS, Telemetry, Host Interface, Profiles, FW Update, Boot model / option, PCI device and slot, OEM Extensions, OpenAPI, Fabrics, Sensor Model, Alarms, Jobs, Privilege Map, DCIM, Ethernet Switching
- Fundamental Architecture design points never released:
  - ***Aggregation***

Death by Data Model

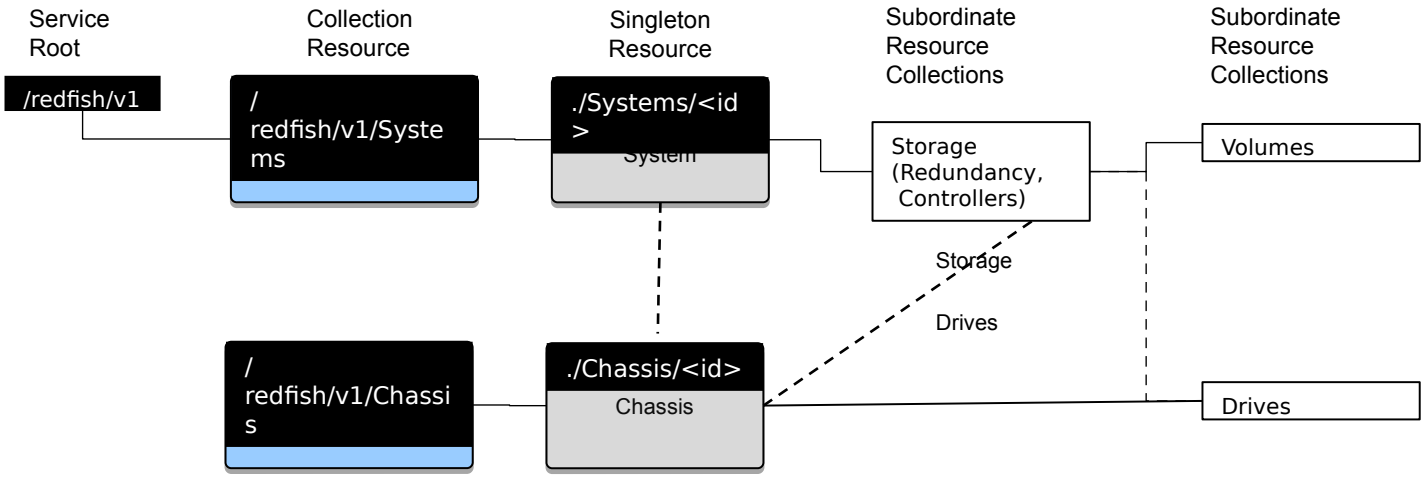
Initial Redfish Data Model



Current Server Redfish Data Model



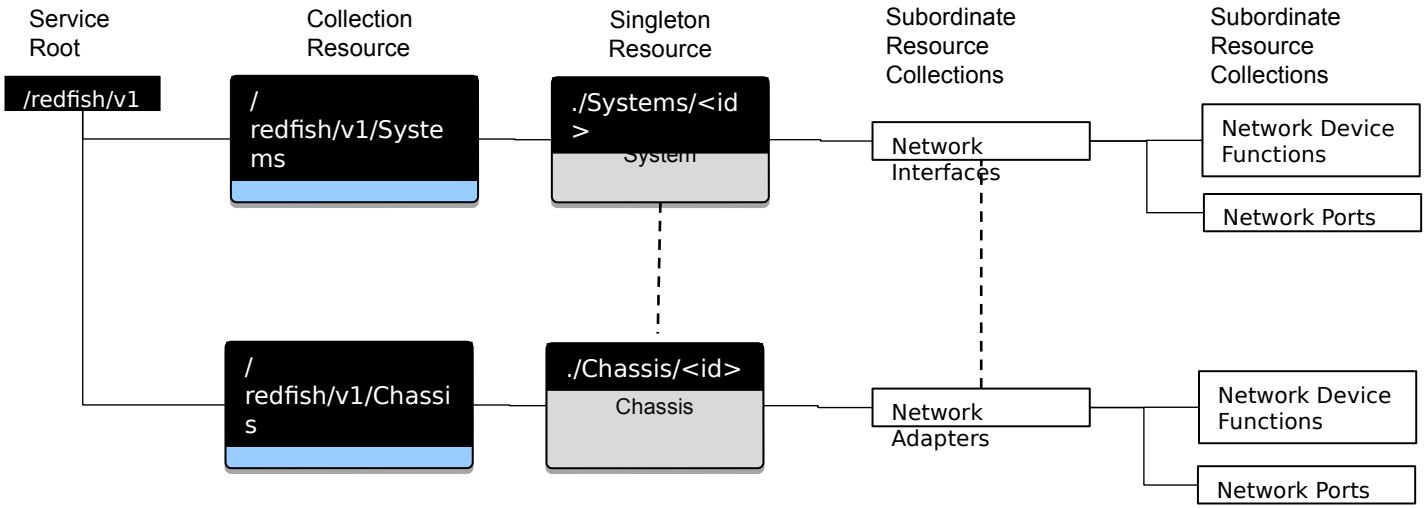
Server Storage in Redfish



This is a hardware representation. This is a different approach than Swordfish which is more akin to a service oriented model.

Storage has a Controllers object in it.
Drives are in an array
Volumes are in a collection.
Post Drives to a Volume to create a volumes

Server Adv. Comm. Devices (ACD) in Redfish

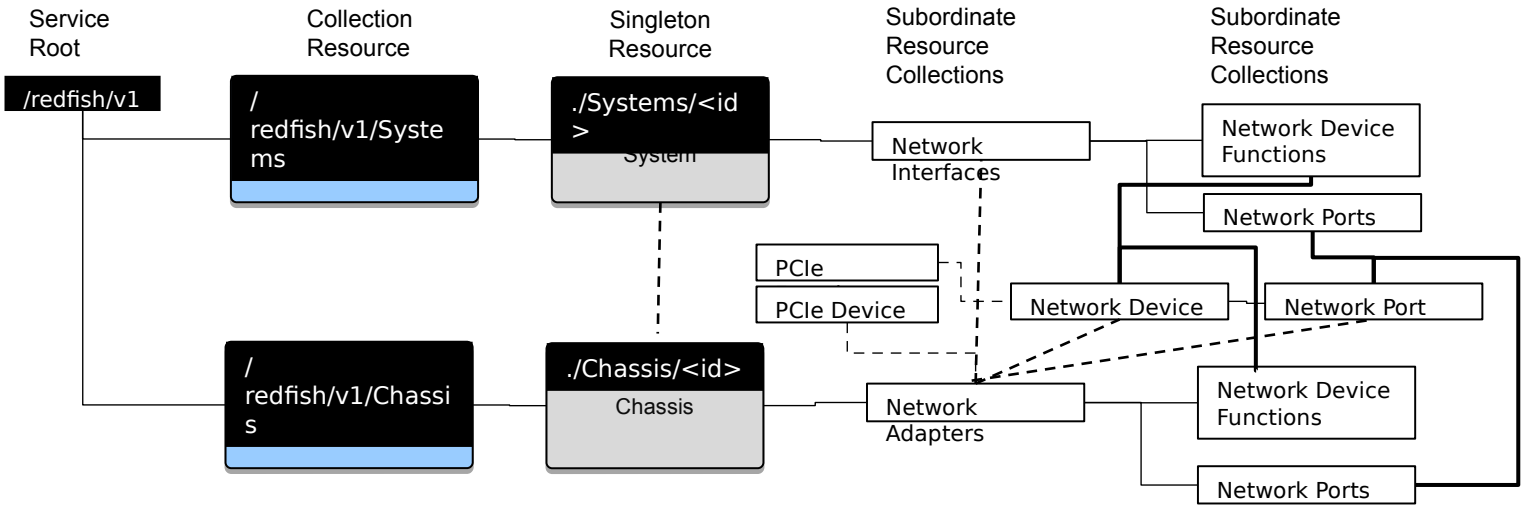


Network Interfaces is the logical view of the ACD. Network Adapter is the physical view and the contents may not be the same

- Such as in Compose where the OS sees only the part of the ACD it is allowed to know about).

Again, Redfish is a hypermedia interface, so we're not showing the path to the Network Device Functions and Ports. In fact, we're not showing the instances yet, just the collections. So lets add them…

Server Adv. Comm. Devices (ACD) in Redfish (con't)

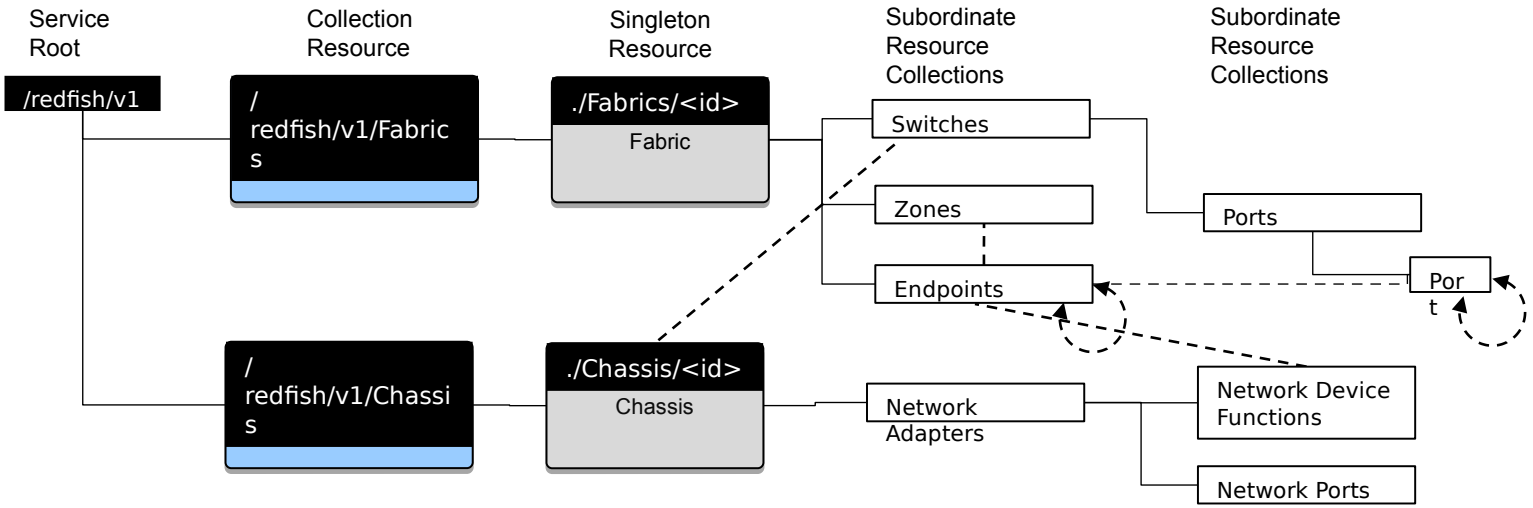


The lines are just hrefs basically. So from the chassis view you can make functions (PCIe Functions) on the Adapter and give the systems access (in the case of compose).

Collections are really just arrays of hrefs. So while this looks complicated, it allows for dynamic assignment of PCIe Functions to Ports and Systems and allows the creation of the Functions on the Network Adapters

Why are PCIe Devices & Functions different resources? Because of Fabrics…www.dmtf.org

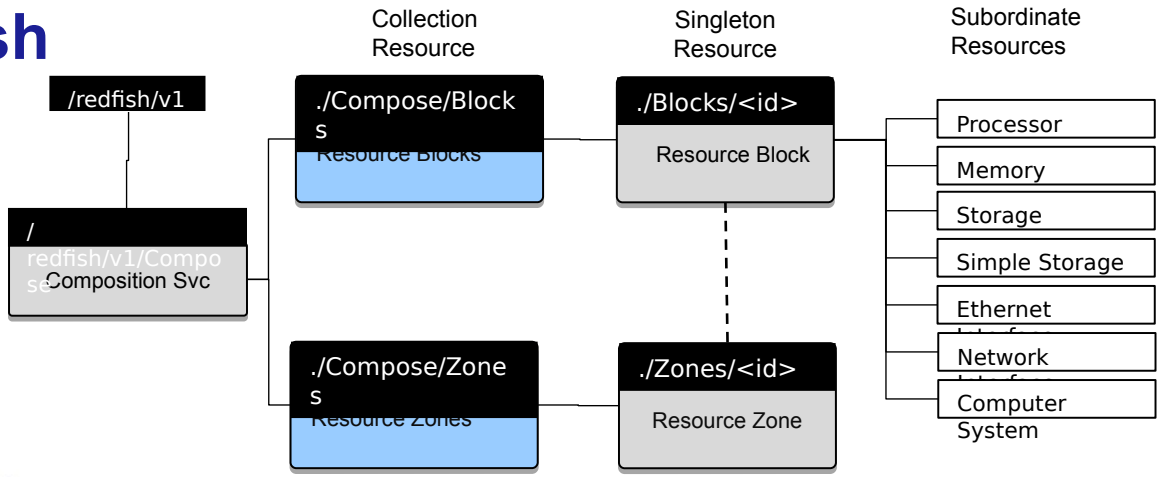
Fabrics in Redfish



- Zones specify allowable access between endpoints.
- Devices/Entities are associated with Endpoints. The endpoint model is the underlying transport.
- Devices/Entities could be in Chassis or Computer Systems (kept out of picture to keep it simple)
- Adding Initiator/Target information to endpoints to help facilitate fabric management.

Common Model	PCIe	SAS
Fabric	N/A	N/A
Zone	PCIe Zone	SAS Zone
Endpoint	PCIe Function	SAS EP
Switch	PCIe Switch	SAS Swtich
Port	PCIe Port	N/A
N/A	PCIe Device	N/A

Compose in Redfish



Blocks represent the resources.

Zones represent the groups of resources that can be “composed” together. ResourceBlocks can be in more than one ResourceZone.

Compose a system by POSTing to Computer System. It has a capabilities object to indicate what can be composed. Systems can also be pre-composed

2 1/2 Compose Methods – Specific and Constrained. There is also Extension to existing systems.

Resource Block Collection can be exposed off of ServiceRoot for RMs.

There are more of these, but let's stop here

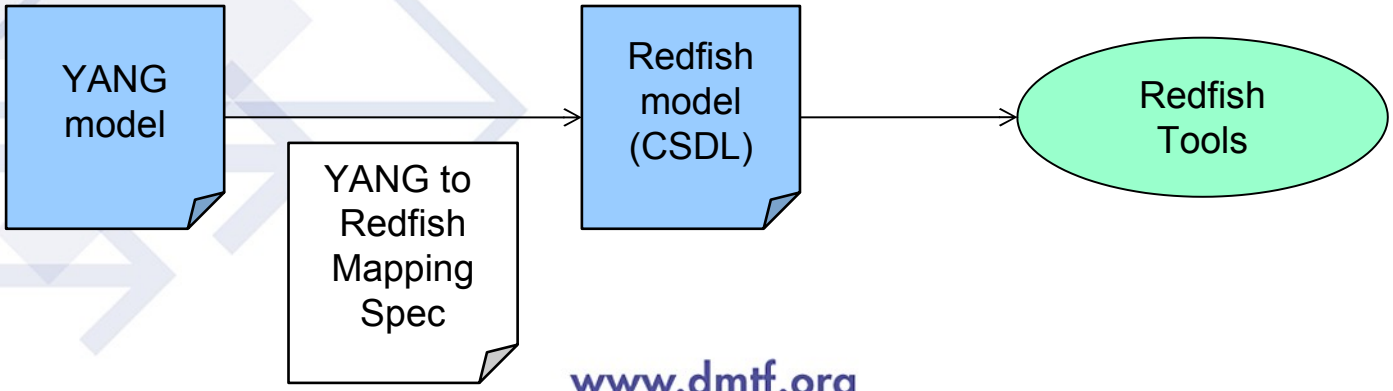
- Telemetry
- Certificates
- FW Update/SW Update
- Accounts, Roles, LDAP/AD
- Sessions
- Eventing (Push and SSE)
- Memory (including Persisten)
- Processors (FPGA/GPU)
- DCIM (Power/Cooling/ASHRAE)
- Simple Storage, Simple NICs
- BIOS
- Virtual Media, SSH, KVM
- Tasks, Jobs, Schedule

... And more

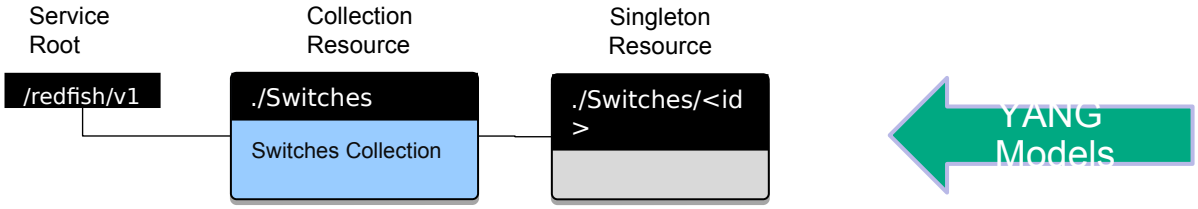
- OEM extensions in payloads
- Query Parameters
  - Expand, Filter, Select
  - Top, Skip, Only, Excerpt
- Note that Ethernet Switches are a very different Model
  - Next slides

Ethernet Switch Representation: YANG to Redfish

- Enable converged infrastructure management
  - One interface (one tool chain) to manage compute, storage and network
  - Switches have platform components in common with servers and storage
  - Network Functions Virtualization (NFV) will need common manageability for compute and networking
- DMTF wants to leverage the networking industry's expertise
  - YANG is the basis for general network industry manageability
  - Large body of existing YANG work
  - Model driven approach to network management



Ethernet Switches in Redfish

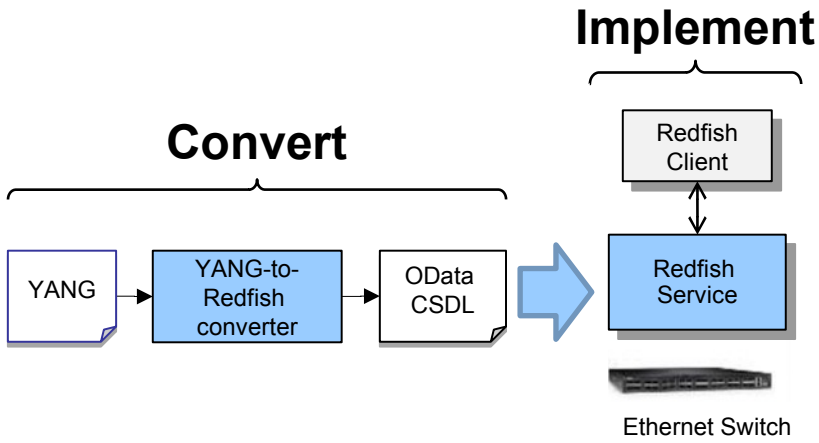


- Phase 1 - convert a small set of YANG models to Redfish models to manage an Ethernet Switch
  - Prove out the process
  - Publish and validate the programmatic conversion tool
- Phase 2 – larger list of YANG models

Phase 1: RFCs

- RFC7223 (Interfaces)
- RFC7224 (IANA Interface types)
- RFC7277 (IPv4 and IPv6)
- RFC7317 (system, system_state, platform, clock, ntp)

Phase 1: OpenConfig instead



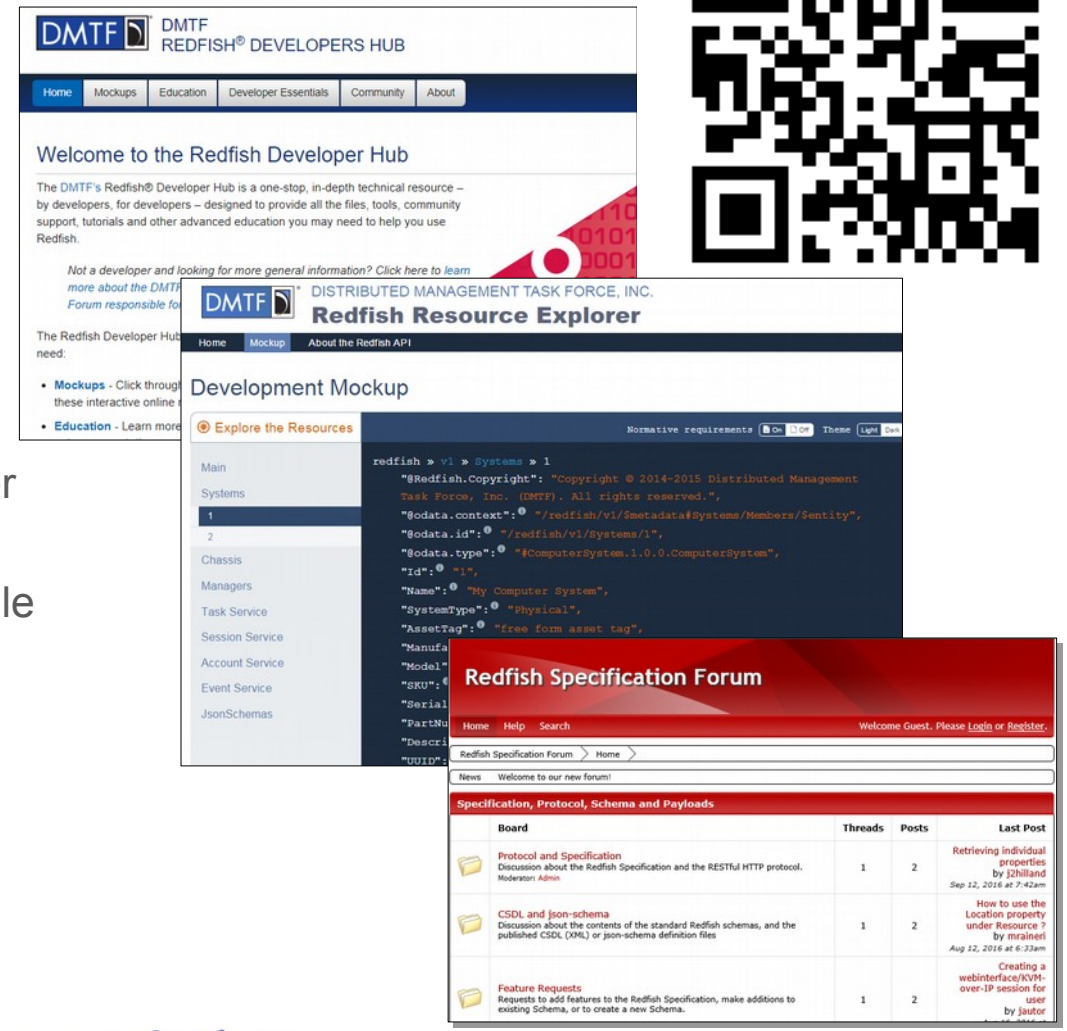
Two More Slides

In Summary

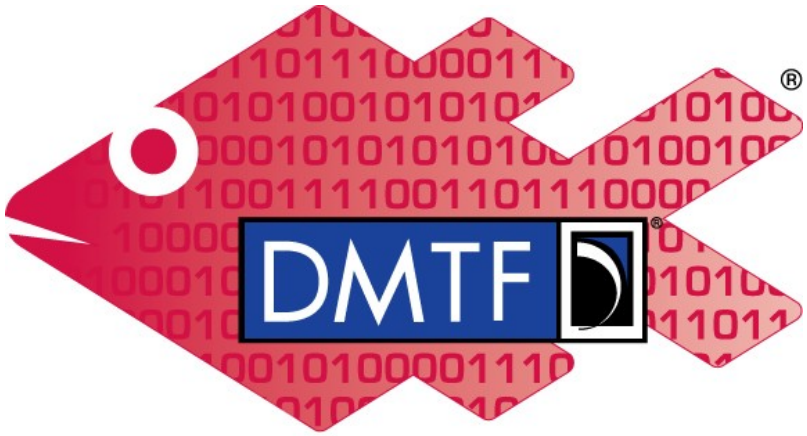
- Redfish, along with the DMTF alliance partners, is working to define interoperable software defined hybrid IT management for servers, storage, networking, power/cooling, fabrics and more
- And is solving problems from composition to resource managers, aggregation engines
- As well as plumbing the mechanisms inside the box to be self contained and self describing

Redfish Developer Hub: redfish.dmtf.org

- Resources
  - Schema Index
  - Specifications
  - GitHub for Redfish Tools
  - Registries
  - Other Documentation
- Mockups
  - Simple Rack-mounted Server
  - Bladed System
  - Proposed OCP Redfish Profile
  - More being added
- Education/Community
  - Redfish User Forum
  - Whitepapers, Presentations
  - YouTube shorts & Webinars

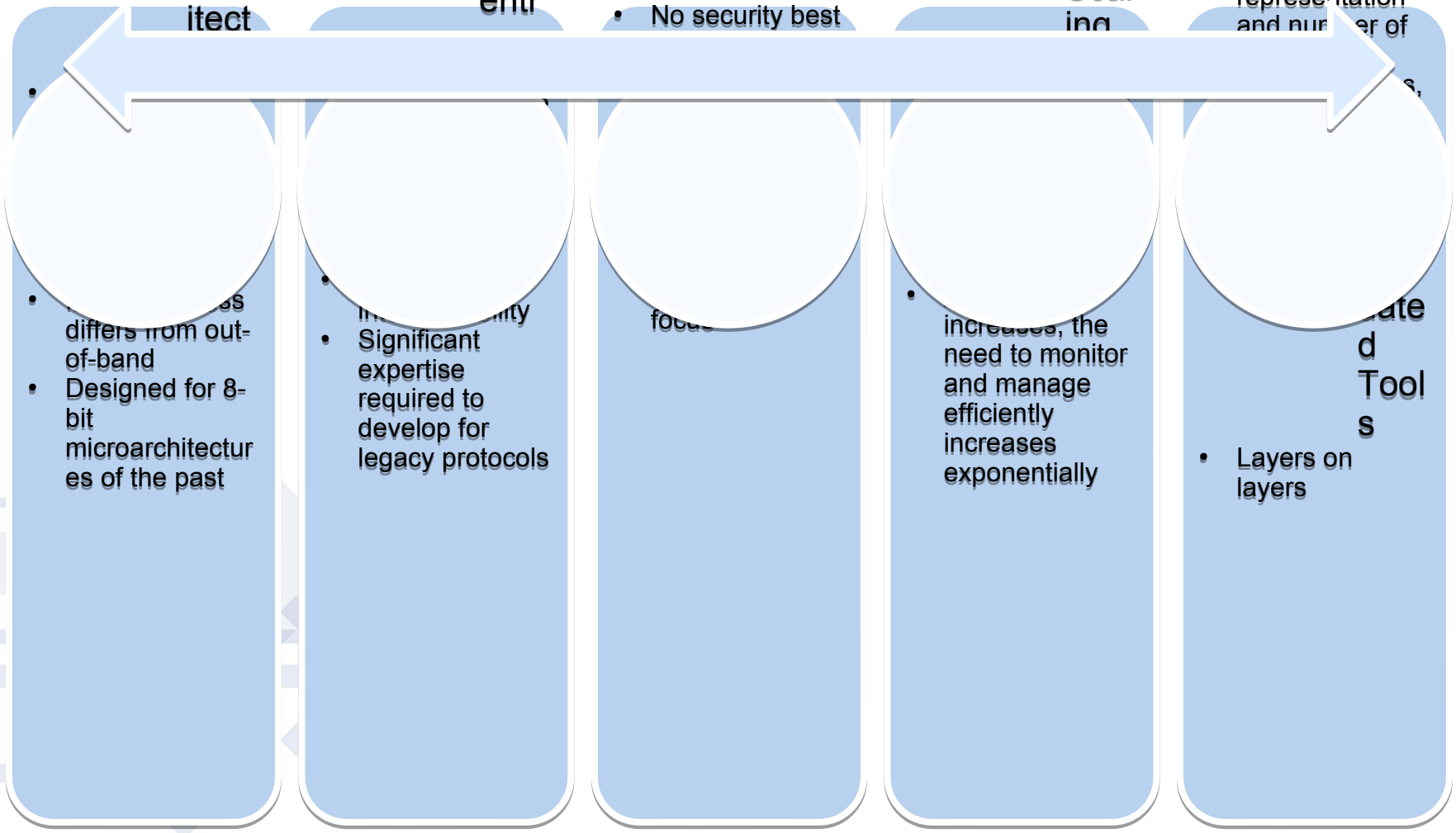


Thank you!



Backup Material

The Status Quo prior to Redfish

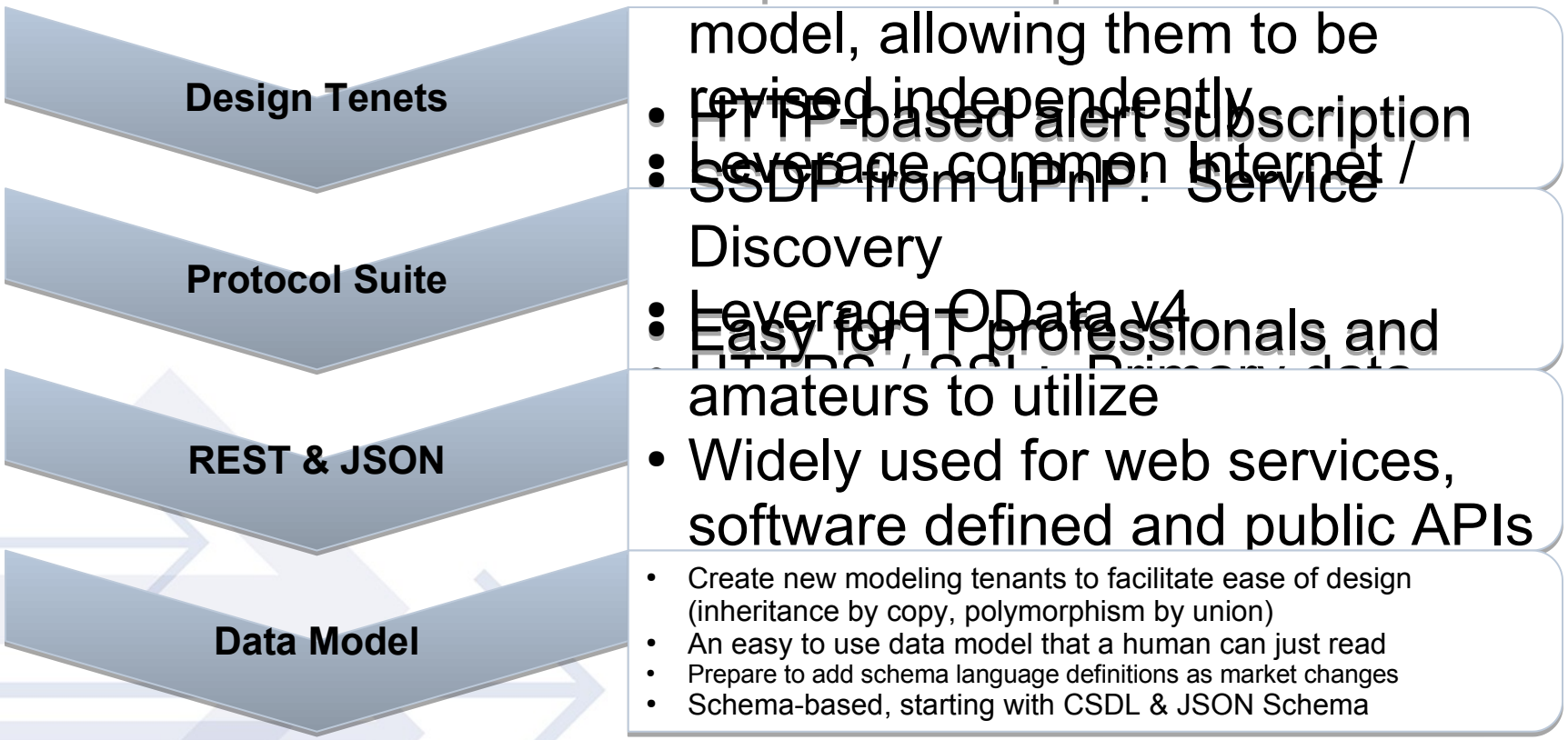


What we did to fix the Status Quo

How to Re-invent IT Management

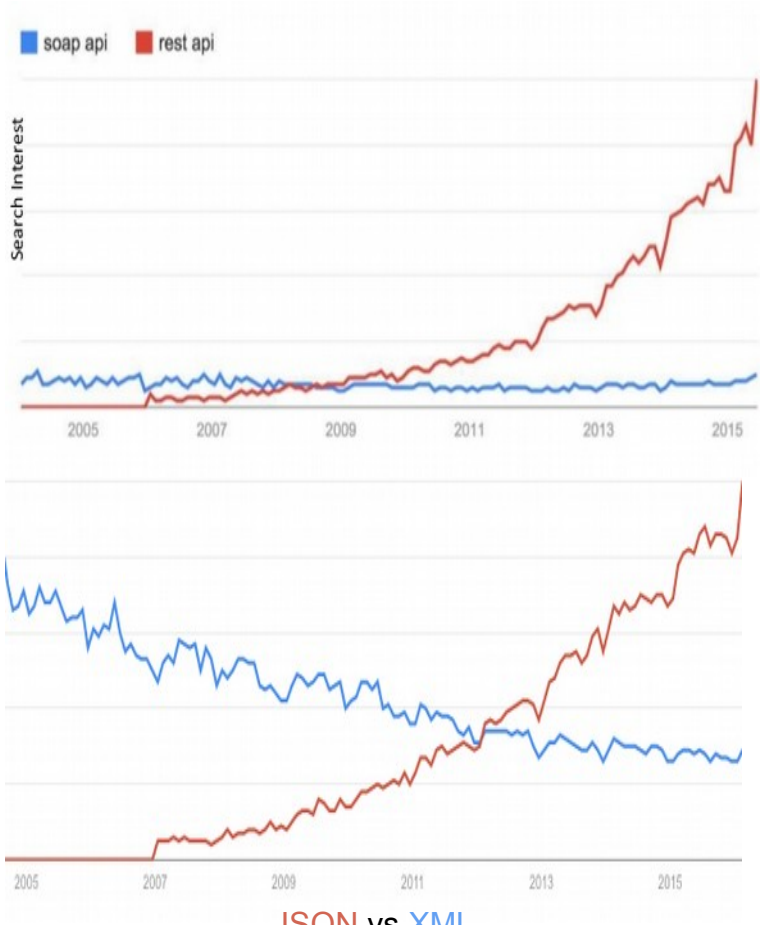
Interoperability is key

A Hybrid IT Management Solution



Why REST, HTTP and JSON?

- **REST:** The API architecture
  - Rapidly replacing SOAP
- **HTTPS:** The Web protocol
  - Well-understood by admins
  - Known security model
  - Known network configuration
- **JSON:** Modern data format
  - Human-readable
  - Simpler than XML
  - Modern language support
- The combination of language support and ubiquity of REST, HTTP and JSON means that IT management tasks can be performed using the same skill set and tool chain as all other IT and dev/ops tasks.



JSON vs XML

Redfish Tools Task Force

Open source tools to enable Redfish: http://www.dmtf.org/standards/opensource

	Tool	Description
Extend	CSDL Validator	Validates the CSDL conforms to Redfish requirements
	CSDL-to-JSON schema convertor	Generates json-schema files from CSDL
	YANG to Redfish	Converts a YANG model into a set of Redfish CSDL files, enabling Ethernet switching standard access via Redfish
Working Svc	Document Generator	Generates documentation from json-schema
	Mockup Server	Exposes a mockup as a static HTTP service (GETs only)
	Mockup Creator	Creates a mockup from a Redfish service
	Profile Simulator	Dynamic simulator of the proposed Redfish profile for OCP
	Interface Emulator	Emulate a Redfish interface statically or dynamically.
Test	Service Validator	Validates a Redfish service is conformant
	JSON Schema Response Validator	Validates any JSON resource against DMTF provided JSON schemas
	Reference Checker	Validates the reference URLs in CSDL files
	Use Case Checker	Collection of tools to validate common use cases for Redfish Services.
	Service Conformance Tool	Verifies conformance of a Redfish service to assertions in the Redfish Specification
Client	CLI (redfishtool)	A command line tool for interacting with a Redfish service (similar to ipmitool)
	Event Listener	A lightweight HTTPS server that can be executed to read and record events from a Redfish Service
	C Library (libRedfish)	C libraries for interacting with Redfish services
	Python Utility & Library	A Command line tool with UI and python libraries for interacting with Redfish services

How simple is Redfish?

Example Python code to retrieve serial number from a server:

```
rawData = urllib.urlopen('http://192.168.1.135/redfish/v1/Systems/1')
jsonData = json.loads(rawData)
print( jsonData['SerialNumber'] )
```

Output is:

```
1A87CA442K
```

Three lines of code: point to the resource, get the data, print the serial number.

***Example uses Redfish ComputerSystem resource**

Redfish v1.0 Feature Set

Retrieve "IPMI class" data
- Basic server identification and asset info
- Health state
- Temperature sensors and fans
- Power supply, power consumption and thresholds

Basic I/O infrastructure data
- Host NIC MAC address(es) for LOM devices
- Simple hard drive status / fault reporting

Discovery
- Service endpoint (network-based discovery)
- System topology (rack/chassis/server/node)

Security
- Session-based leverages HTTPS

Perform Common Actions
- Reboot / power cycle server
- Change boot order / device
- Set power thresholds

Access and Notification
- Serial console access via SSH
- Alert / event notification method(s)
- Event Log access method(s)

BMC infrastructure
- View / configure BMC network settings
- Manage local BMC user accounts

Working on more…

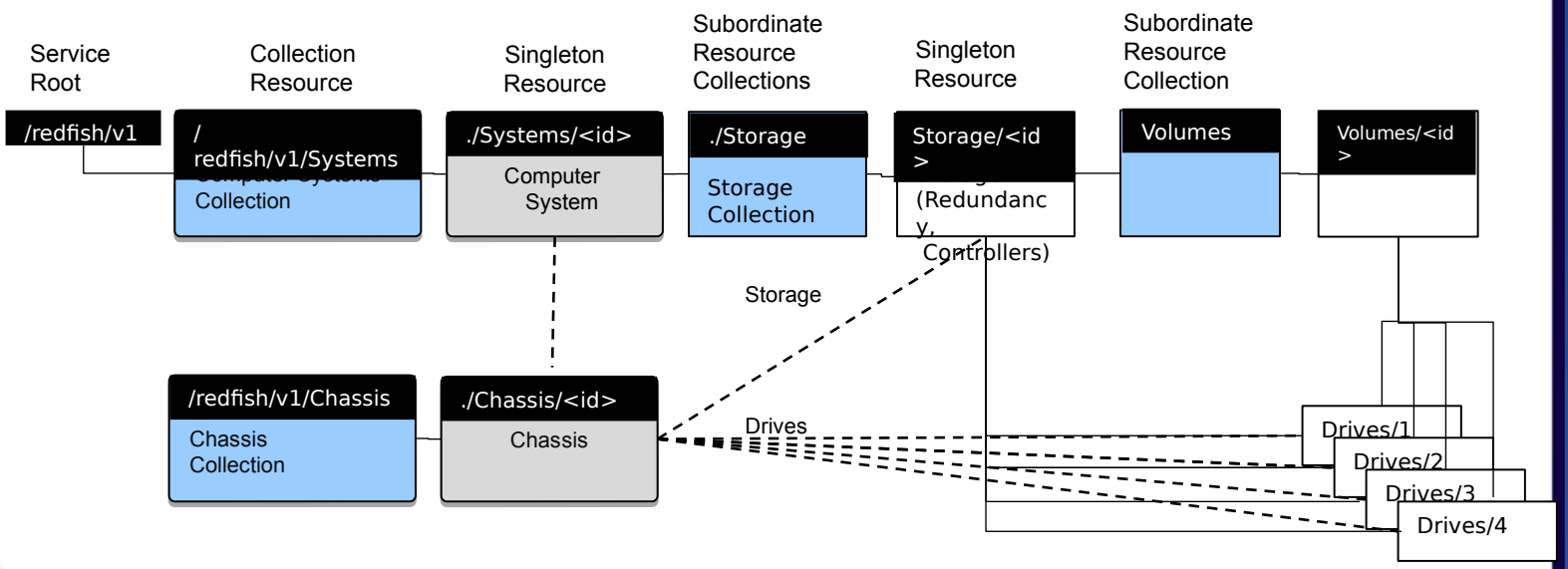
Redfish Storage Model

a.k.a. Local Storage, Storage “Lite”

Storage Resource Overview

- Storage: A representation of a storage sub-system
  - Contains sets of Volumes, Drives, and Storage Controllers
  - Storage Controller information is an array of objects in the Storage resource
    - Describes the protocols supported by the controller, the speed of the controller interface, and manufacturer information about the controller
- Drive: The physical media for the data
  - Manufacturer information about the drive (part number, serial number, etc)
  - Capability information about the drive (size, protocol, encryption, etc)
  - Contains control aspects (secure erase and LED setting)
- Volume: The logical construct used by the OS/hypervisor
  - Contains status about a volume (what drives contribute to the volume, size information, identifier information, etc)
  - Allows a client to control the volume (initialization, encryption settings, etc)

Server Storage in Redfish



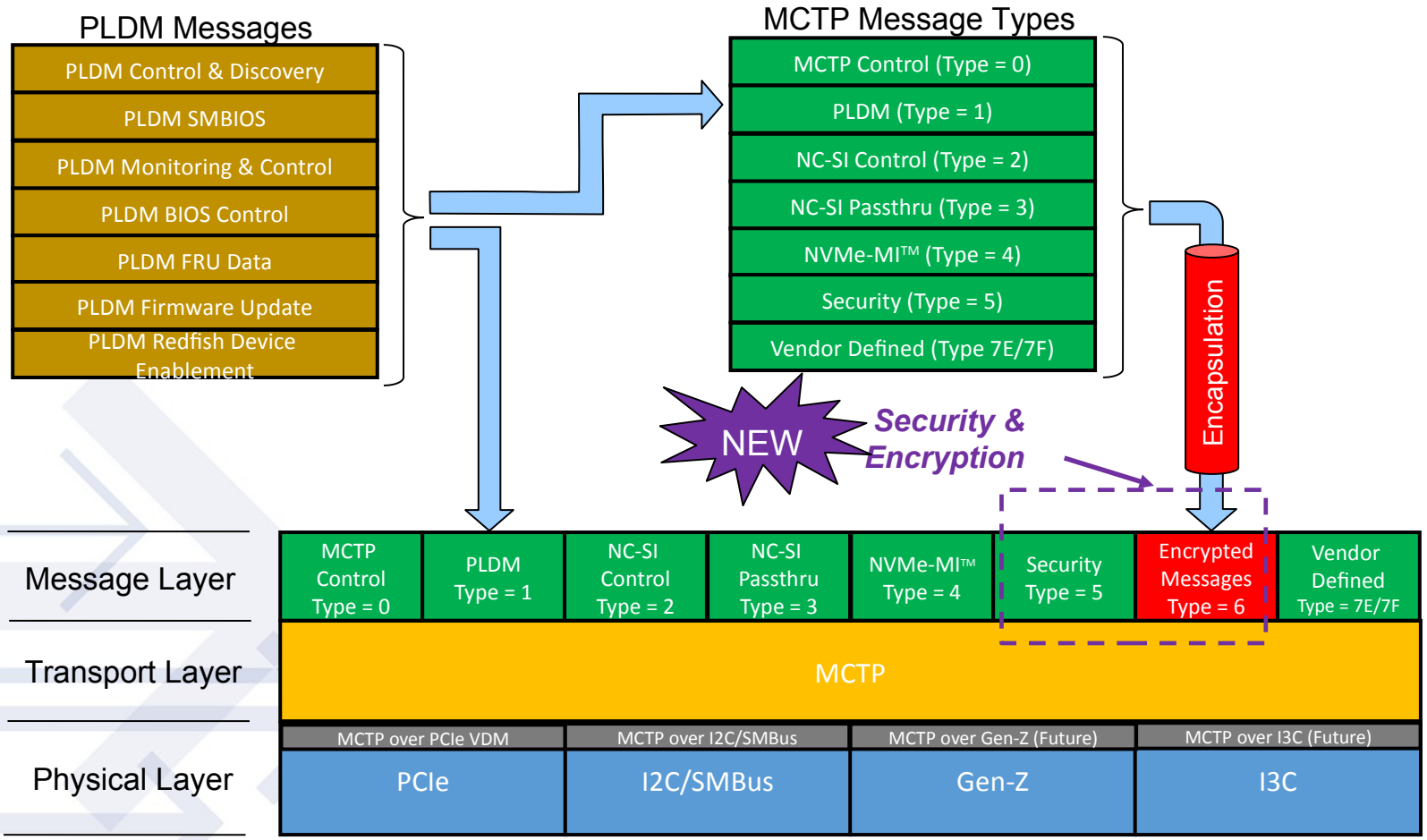
Note that the Volumes are in Collections off of the Storage resource, drives are in arrays off of the storage resource and optionally the Chassis.

PMCI

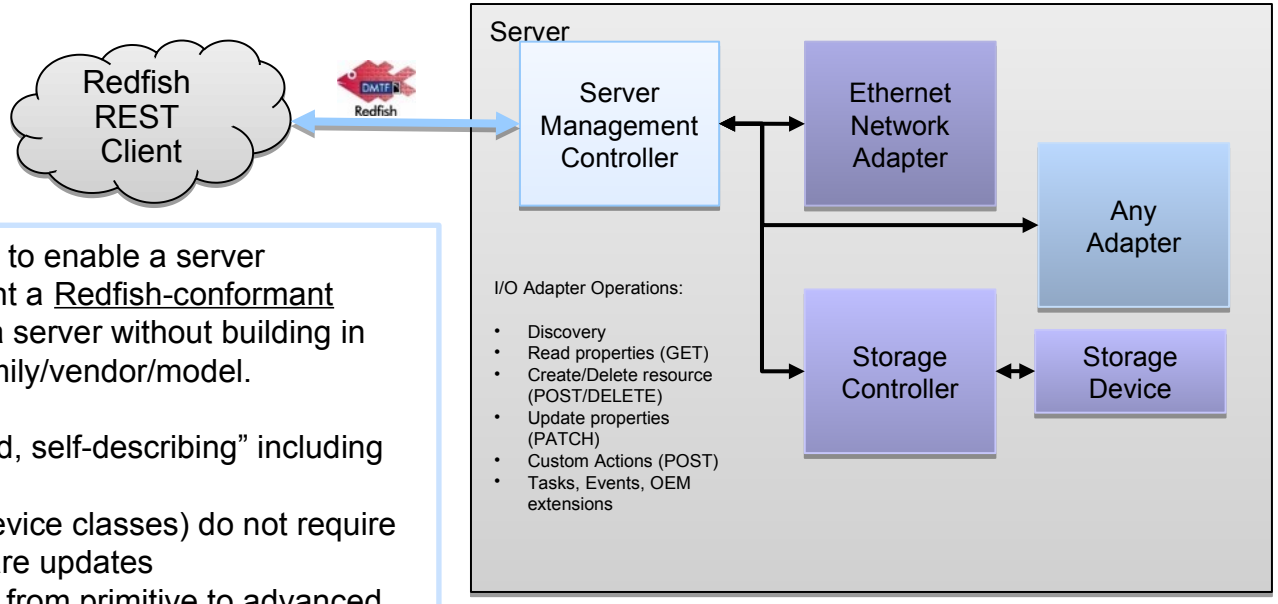
PMCI WG

- Inside the Box suite of specs
  - MCTP – Transport layer spec that maps to I2C, PCIe, Serial, Gen-Z, etc.
    - Control – Sensor Data
    - NVMe-MI – Used by NVMe for management information
    - Security – new protocol for attestation and authentication
    - Encryption – being developed after security to encrypt management traffic
  - PLDM – transported on top of MCTP
    - Monitoring & Control – Sensor Data
    - SMBIOS & FRU – so MC can expose this data
    - Firmware Update – how Redfish FW Update gets handled
    - RDE – Redfish Device Enablement: A provider architecture for Redfish
  - NC-SI
    - Started out using RMII. Now can go over MCTP. It's a NIC thing

MCTP Security Proposal – Diagram View



Redfish Device Enablement: PLDM Redfish Providers



PMCI WG developing a standard to enable a server Management Controller to present a Redfish-conformant management of I/O Adapters in a server without building in code specific to each adapter family/vendor/model.

- Support adapter "self-contained, self-describing" including value-add (OEM) properties
- New managed devices (and device classes) do not require Management Controller firmware updates
- Support a range of capabilities from primitive to advanced devices (lightweight/low bandwidth options)
- Leveraging PLDM, a provider architecture is being specified that can binary encode the data in a small enough format for devices to understand and support.
- MC acts as a broker to encode/decode the data to/from the provider
- PLDM works over I2C & PCIe VDM. Additional mappings under consideration.

RDE

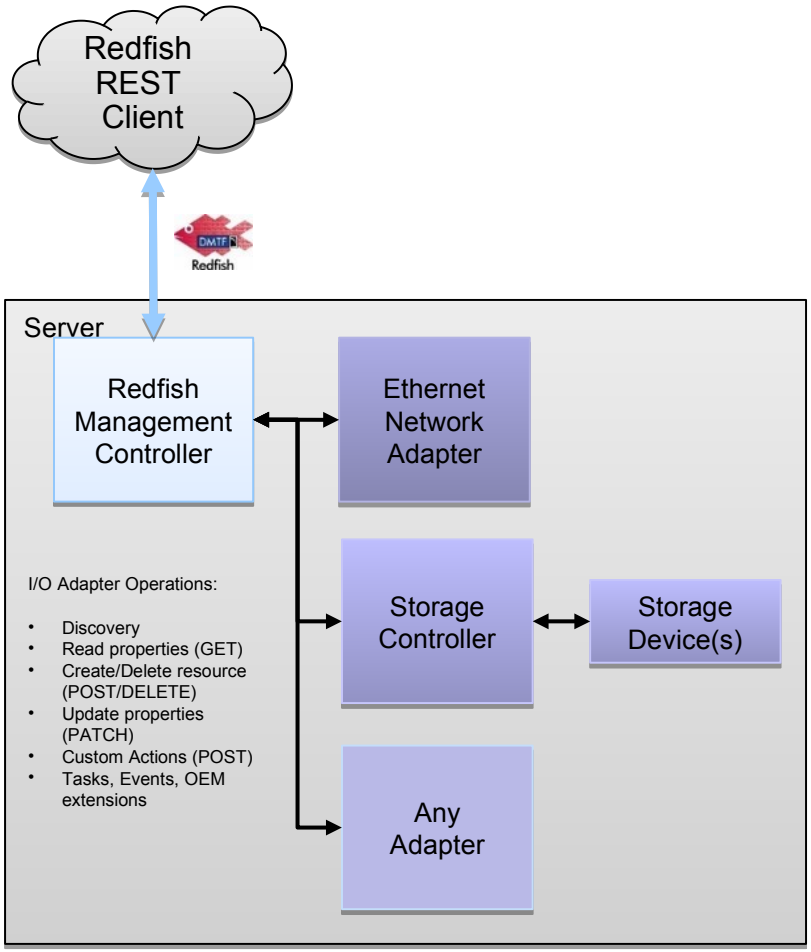
Or

"How you can fill all that storage stuff out without creating a lock step firmware dependency between the management controller firmware and the storage firmware"

Redfish Device Enablement: PLDM Redfish Providers

PMCI WG developing a standard to enable a server Management Controller to present a Redfish-conformant management of I/O Adapters without building in code specific to each adapter family/vendor/model.

- Support adapter "self-contained, self-describing" including value-add (OEM) properties
- New managed devices (and device classes) do not require Management Controller firmware updates
- Support a range of capabilities from primitive to advanced devices (lightweight/low bandwidth options)
- Leveraging PLDM, a provider architecture is being specified that can binary encode the data in a small enough format for devices to understand and support.
- MC acts as a broker to encode/decode the data to/from the provider
- PLDM works over I2C & PCIe VDM. Additional mappings under consideration.



RDE Discovery & Registration

Discovery is PLDM based

- Devices are discovered using PLDM (which uses MCTP) and determines that it supports RDE (PLDM Type 6)
- MC uses RDE to negotiate parameters with the device.
  - Concurrency, Operations Supported, Provider Name
- MC uses RDE to negotiate channel parameters on each channel
  - Asynchrony, Max Chunk Size

Registration leverages Platform Data Record

- MC queries device for PDRs and any Associations and Actions
  - PDR = Platform Data Record. Equates to one or more Redfish Resources
- And gets Schema Identities and versions for the PDR
  - Can get a Resource ETag
- And Retrieves any Dictionary for the PDR
  - Dictionaries may be truncated to only what device supports

RDE BEJ – Translating JSON to binary using dictionaries

- RDE's BEJ (Binary Encoded JSON) is a binary representation of the JSON payload using the algorithm specified
- Dictionaries are the key
- So the dictionary is used to turn each of these into a compact form.
  - RDE uses the dictionary to take the JSON Body and turn it into binary
  - Includes how to nest objects, handle annotations and other Redfish nuances
  - Roughly a 10:1 compression in early analysis
- DMTF will publish Dictionaries for all DMTF Schema on the website
  - Program to generate dictionaries will be made open source

RDE Operations

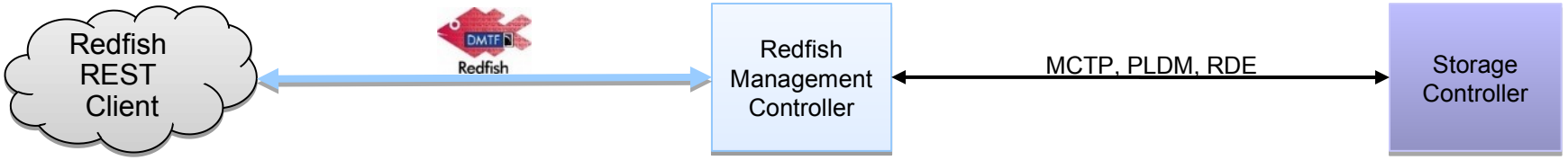
- RDE has slightly different terms than HTTP in the spec

HTTP Operation	RDE Operation
GET	Read
PUT	Replace
PATCH	Update
POST	Action or Create
DELETE	Delete
HEAD	Read Headers

- RDE supports multiple outstanding operations, tasks and events

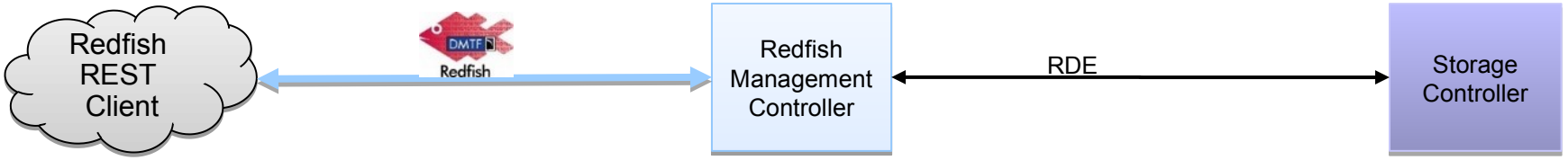
Command	Usage
RDEOperationInit	Begin an Operation
SupplyCustomRequestParameters	Provide additional parameters for Operations
RetrieveCustomResponseParameters	Get additional response data
RDEOperationStatus	Check up on an active Operation
RDEOperationComplete	Finalize an Operation
RDEOperationKill	Cancel an Operation
RDEOperationEnumerate	See what Operations are active
MultipartSend, MultipartReceive	Bulk data transfer

So how does all this fit together?



- Before a Client ever Contacts the Redfish Service:
  - The Management Controller uses MCTP to enumerate the devices
    - This is dependent on the Medium (I2C, PCIe, etc.)
  - MC then uses PLDM for next phase of discovery
  - PLDM support for Type 6 (RDE) is discovered as being supported!
  - RDE Discovery takes place over PLDM
  - RDE Device Registration
    - Get all the Resources, Actions, and Entity Associations
    - MC Retrieves the Dictionaries
- The MC is now ready to handle operations to and from the device

So how does all this fit together?



- Request comes in for client
- MC determines that the URI equates to an RDE provider.
- MC translates HTTP headers that need to be passed down
- The MC translates the JSON body into BEJ using the dictionaries.
- MC initiates the operation to the device.
- Device processes the request
- MC takes the response, reassembles it and translates from BEJ to JSON using the dictionaries.
- MC formulates HTTP Response and sends to Client

Additional information about RDE

- RDE also handles Tasks,
  specifies how Events are handled,
  and has information on handling OEM sections.

- There are state machine examples and tables.
- Binary Format for dictionaries is specified

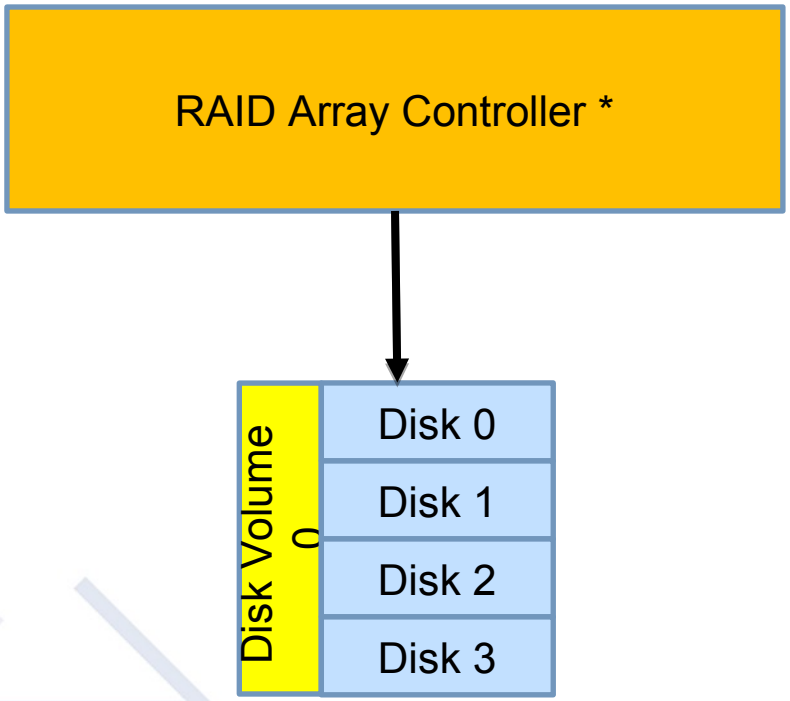
- There are examples of
  - what a Redfish tree would look like,
  - how the PDR would look like for that tree,
  - Examples of what a dictionary would look like
  - Example of BEJ encode/decode.

RDE Details

Or

"How you can fill all that storage stuff out without creating a lock step firmware dependency between the management controller firmware and the storage firmware"

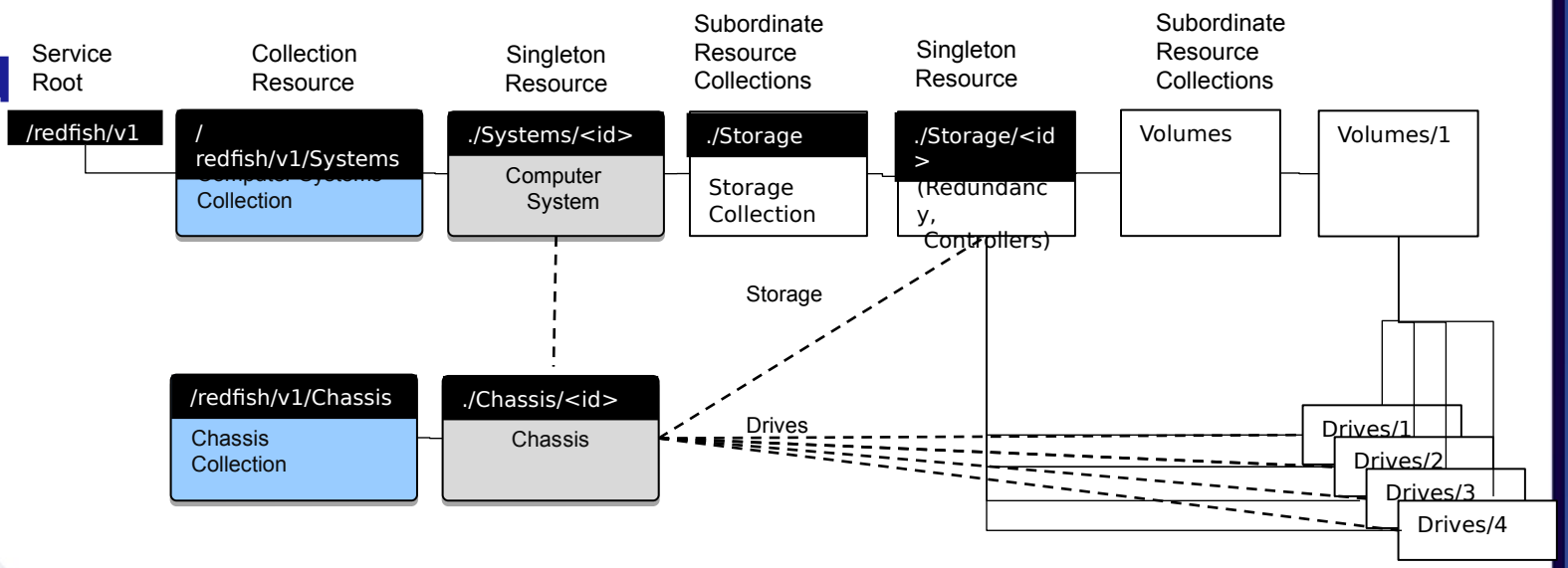
Disk Subsystem (Physical Layout)



* For this example, we're assuming that the raid array controller firmware exposes management for everything seen here

Server Storage in Redfish

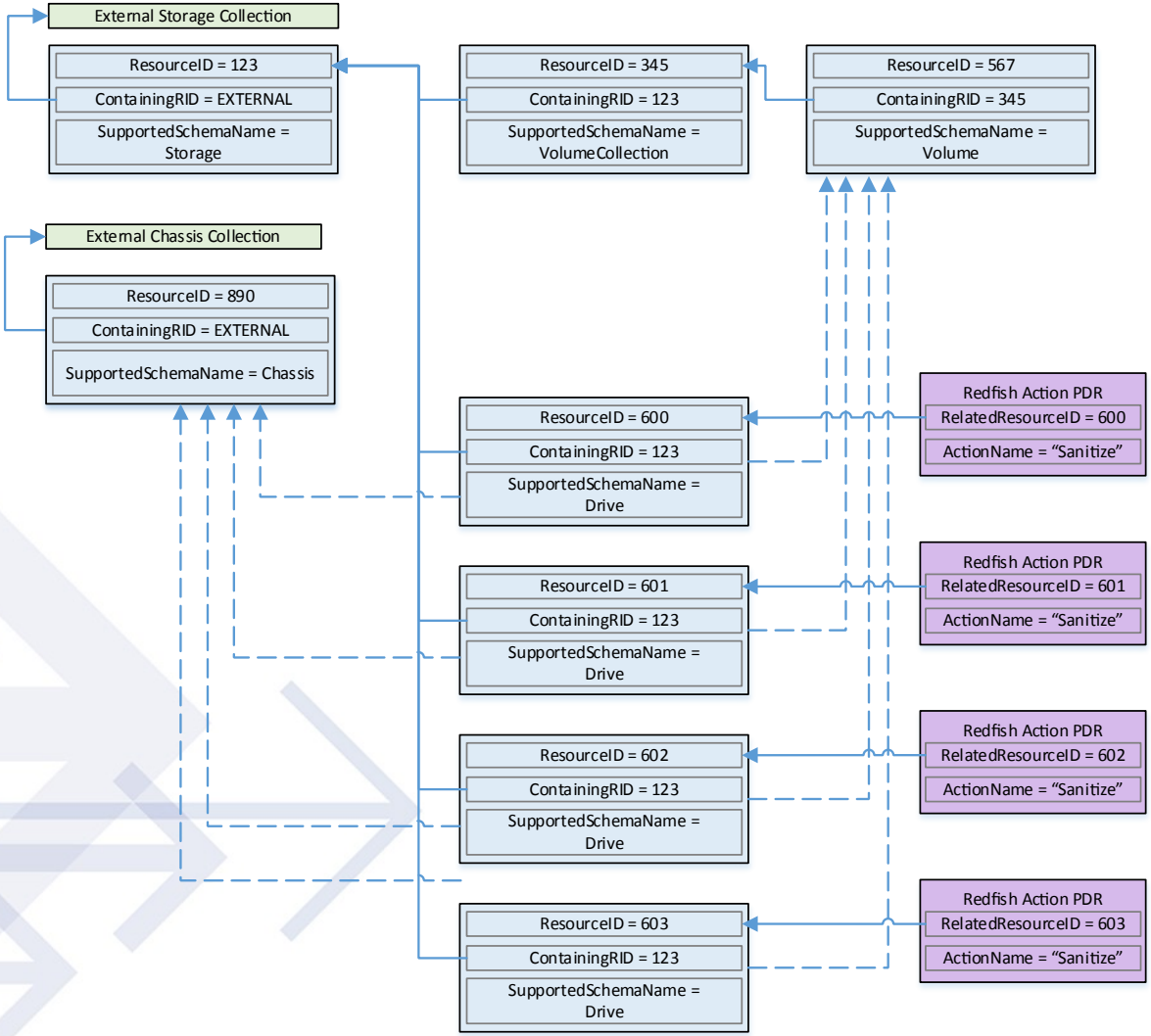
Gets Modeled Like This:



Note that the Volumes are in Collections off of the Storage resource, drives are in arrays off of the storage resource and optionally the Chassis.

Redfish Entity Association PDRs

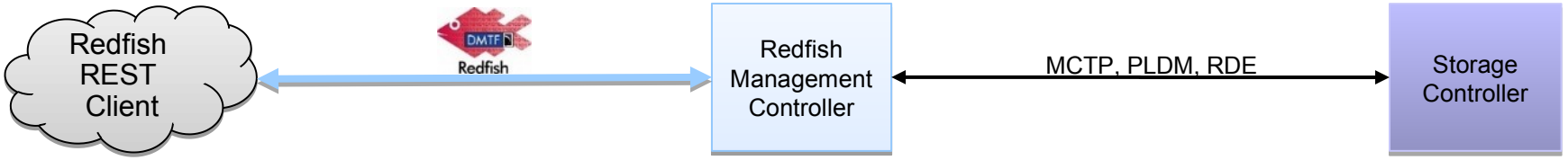
PDRs would look like this:



RDE BEJ – Translating JSON to binary using dictionaries

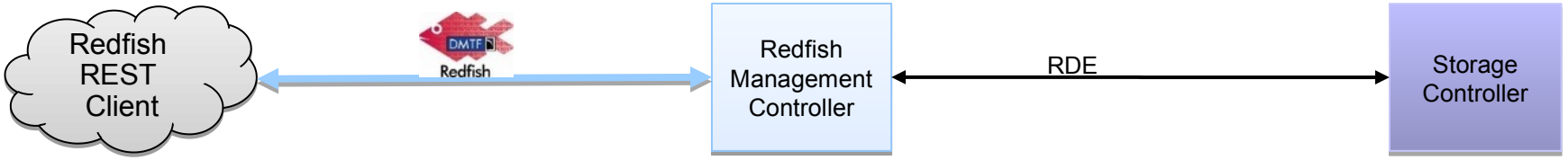
- RDE's BEJ (Binary Encoded JSON) is a binary representation of the JSON payload using the algorithm specified.
- Dictionaries are the key
  - Truncated representation of the Schema
    - One dictionary per schema
    - 100% backward compatible! Can use a newer dictionary on older resources.
  - Used to turn property names and objects into numbers
  - Used to turn enums into numbers
  - Numbers are already easy to represent in binary.
    - Real can be tricky but there are few of them.
  - String values stay strings
- JSON objects are made up of
  - Properties, objects, numbers, enums, strings, null
- So the dictionary is used to turn each of these into a compact form.
  - RDE uses an algorithm that uses the dictionary to take the JSON Body and turn it into binary.
  - SFLV Tuples – Sequence Number, Format, Length, Value.
    - Sequence number equates to property and is relative to level.
  - Includes how to nest objects, handle annotations and other Redfish nuances
  - Roughly a 10:1 compression
- DMTF will publish Dictionaries for all DMTF Schema on the website
  - Program to generate dictionaries will be made open source

So how does all this fit together?



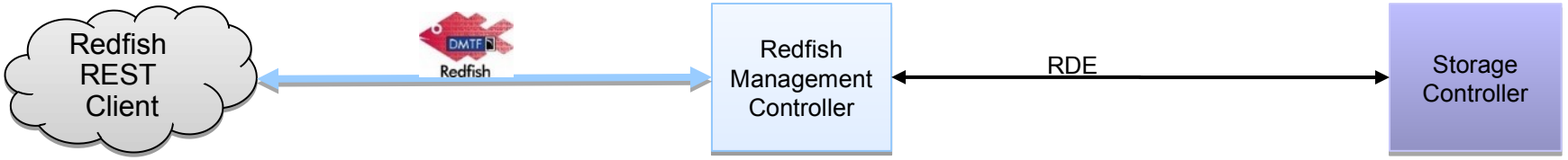
- Before a Client ever Contacts the Redfish Service:
  - The Management Controller uses MCTP to enumerate the devices
    - DSP0236 and mapping specs cover this in detail.
    - Phases: Bus enumeration, Bus address assignment, MCTP capabilities, EID assignment, routing info
  - This is dependent on the Medium (I2C, PCIe, etc.)
    - See the MCTP Mapping Specs for details
  - MC then uses PLDM for next phase of discovery
    - DSP0240 covers this in detail
    - Uses SetTID, GetTID, GetPLDMVersion, GetPLDMTypes, GetPLDMCommands
  - PLDM support for Type 6 (RDE) is discovered as being supported!
    - Other PLDM Types like FW Update and Monitoring & Control will be discovered and can be used by the MC and exported through Redfish *BUT* these are all specialized providers in the MC

So how does all this fit together?



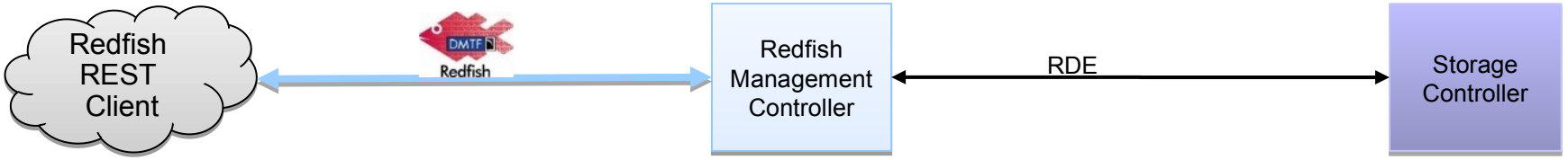
- Before a Client ever Contacts the Redfish Service:
  - RDE Discovery takes place over PLDM
    - NegotiateRedfishParameters to get the Provider name, concurrency support and features supported
    - NegotiateMediumParameters on each medium it plans to communicate on
  - RDE Device Registration
    - GetPDRRepositoryInfo & GetPDR to get PDRs for Resources, Actions, and Entity Associations
      - This is used to build the topology of URIs and links.
      - The ContainingRID is used to build the hierarchy.
    - MC Retrieves the Dictionaries
      - Dictionaries for the data (a.k.a. the major schema), Event, Annotation, Extended Info
- The MC is now ready to handle operations to and from the device

So how does all this fit together?



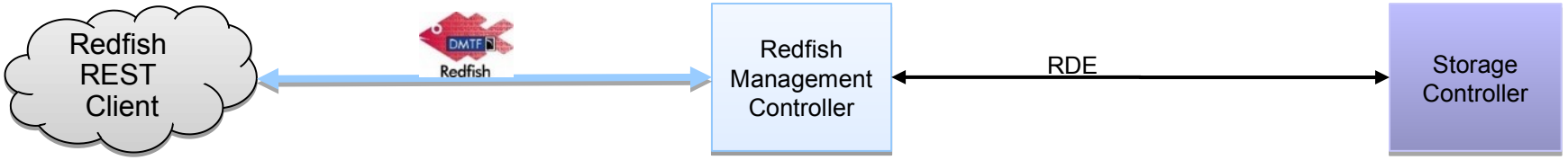
- Request comes in for client (we will use a GET)
  - MC determines that the URI equates to an RDE provider.
  - MC translates the URI into the EID and Resource ID needed to address it.
- MC translates HTTP headers that need to be passed down
  - Not all of them are sent but things like If-Match have to be
  - Query parameters are handled too.
  - Headers not specified by Redfish or RDE are also accommodated.
- The MC translates the JSON body into BEJ using the dictionaries.
  - Each element is turned into an SFLV tuple.
  - Algorithm is in the spec. Surprisingly small amount of code.
  - Accommodate deferred binding (substitution values).
    - Things like Links, Systems, $metadata, etc.

So how does all this fit together?



- MC initiates the operation to the Device.
  - Operation then follows a state diagram.
  - RDEOperationInit
    - This message has the ResourceID, OperationID, OperationType (Read), and any OperationFlags (such as customer request parameters).
    - Device responds
  - SupplyCustomRequestParameters would be used next (if needed)
    - Device responds if all went well. Example, Etag matched so proceed.
  - Since this is a GET, MultipartReceive would be used if bigger than one message.
    - Success response includes the data until done.
  - RetrieveCustomResponseParameters is executed if indicated.
    - Success response includes the data until done.
  - Operation is complete when RDEOperationComplete command.
    - Success indicates OperationID can be used for other operations

So how does all this fit together?



- MC then takes the response, reassembles it and translates from BEJ to JSON using the dictionaries.
- MC formulates HTTP Response and sends to Client

How to Model like Redfish

Extending Redfish

- Two ways to extend the Redfish schema, both are valid
- Add to existing schema
  - New properties can be added
  - New embedded objects can be added
  - Values added to existing enumerations defined for a property
- Create new schema types
  - Can be a new subordinate resource
  - Or even a top-level collection
  - Check first for “inheritance by copy” – can you leverage an existing schema with just a “<thing>Type” property?
    - Example: Rack Manager vs. BMC – both are Managers with a “ManagerType”

Make it easy for the Consumer(s)

- Avoid mistakes from previous interface designs
  - Favor ease-of-user on the consumer (client) side
    - Even if it makes service implementation harder to develop
    - But this is a “balance” – ensure both sides can be implemented
  - Think like the consumer of the interface and how they use it
    - Previously, only a small developer community needed to understand the complex relationships. Now, there are thousands of programmers using these interfaces.
    - If you think “We can document that…”, think again.
  - Flatten the data model
    - Fewer, but richer (larger) resources to minimize I/O traffic and avoid over-use of references
- Follow use cases
  - Consumer will use a programming language or browser to access the interface
  - Think like the programmer using the variables/dictionary or a person reading the JSON via a browser-based REST client
  - Define the various “actors” and how they will use the interface, as there is more than one type of consumer of the interface.

Human Readable JSON

- Start with a Mockup!
  - Copy the Redfish Mockup, modify it for your extensions
  - Use a web browser and a local web server to look at the data
  - Get data in the payload correct before developing schema to save development time
- Assume consumers will never read the Spec or the Schema
  - … nor should they be expected to
  - Property names should be intuitive in context
  - Use complete words like “Power” instead of “Pwr” unless name is too long
  - Don’t repeat
    - In the “Power” object, a property named “PowerCap” is redundant. It should just be “Cap”
    - The consumer is going to be using this in a variable or dictionary anyway, so they will already be referring to Power.Cap or Power/Cap. Power.PowerCap is not good design.

Don’t expose the architecture

- As engineers, we tend to expose every little piece in the architecture. There is usually data or settings associated with every little piece.
  - Customers don’t care how you built it. They only care about access to the information. And they want it as clearly and concisely as possible.
  - If the architectural details don’t affect the use cases, mask for simplicity
  - Think like a customer, not an implementer
    - *“Don’t show the consumer your dirty laundry.”*
- Example 1:
  - You have a system with several management controllers inside that all work in concert with the main management controller.
  - Don’t expose them – make their properties part of the main management controller
  - The customer only cares about the functionality and not each little micro-controller
- Example 2:
  - You have a bunch of fans that are all connected to different I2C controllers
  - Just put them all in the chassis Fans array. Don’t create a bunch of additional “little chassis” in your implementation unless that affects performance or reliability.

Scalability

- Minimize the number of I/Os
  - SSL overhead means I/O cost is in latency. HTTP doesn't use socket connection model.
  - Each transaction is expensive but bandwidth (payload size) is cheap in comparison.
    - So create fewer objects with a larger set of properties
  - Use objects to collect & organize similar data for readability and shorten property names
- Optimize for the common use case (80/20 rule)
  - ComputerSystem has an embedded object that cover Processor Summary. It also has links for more details on Processors. Most consumers just need the summary.
  - By optimizing for the common use case, you can avoid additional I/Os.
- Develop the model assuming OData option "expand" is supported
  - But the Processor collection is a root link off of Computer System
  - With Expand Level=1, links to all of the processors are returned in the System object
  - With Expand Level=2, the processors and all their properties are returned in the System object
  - Thus, instead of 3 or more I/Os, supporting Expand returns it all in a single IO
  - More later under "Subordinate resources vs peer/referenced relationships"

Scalability (continued)

- Model like you don't have a query language
  - When grouping collections, determine what goes in each group
  - OData does have a query language, but it is not used in Redfish
    - Cumbersome for small footprint (management) services
  - It is easier for both the service and the client to return a whole collection
    - Requires less processing on the service side.
    - The client side is going to use the methods in the programming language to access the dictionary/array/construct to find the items it cares about anyway.
  - Put things logically into collections, subordinate or peer objects based on use case and consumer need

Creating New Resources (Schemas)

- All resources are defined using a single “Resource” base class
  - Versioning within a schema is accomplished with traditional inheritance
- Redfish uses “*Inheritance by Copy*” to leverage properties
  - Copy useful property definitions from existing schema(s)
    - Add enumerations or additional properties as needed
  - Minimal use of true references to other schemas (e.g. “Status”)
    - Used for properties where model consistency is both useful and required
  - Keep existing semantics when leveraging properties to avoid customer confusion
    - Copy and rename slightly instead (e.g. Change “SpeedInRPM” to “SpeedInBPS”)
  - Keep leveraged properties in sync with original by hand
    - Avoid burdening customers with complex inheritance defined in multiple schema files
- Use “*Polymorphism by Union*” to re-use schemas for new “flavors”
  - For resources with multiple “types” (think CPU vs GPU), include properties for both
  - Add a “<thing>Type” property with enumeration so consumer can tell the difference
    - Example: “ProcessorType” with enumerations that include CPU and GPU

Adding New Properties

- Make sure the property is useful to the consumer
  - Don't expose unnecessary details or architecture
  - Ask "What will a user (or software) do with this property?"
- Naming
  - Property names should be clear and obvious in context of the schema
  - Use consistent terminology within the schema (and Redfish in general)
  - Balance between human readability and length
- Follow Redfish rules for CamelCase naming style
  - Acronyms larger than three letters become lower case
    - But use common sense, consider ignoring the rule for well known, common nomenclature
- Most properties are optional
- Many properties are null-able
  - Implementation will return property only if this instance supports it
  - If the instance doesn't support the property, don't return it
  - If the value is temporarily unavailable, return property with "null" value

Property Data Types

- Numbers
  - Number types include units as part of the property name
    - Schema ‘units’ annotation is for programmatic consumption (not humans)
    - Use reasonably-sized units to allow for growth, but report typical values as integers
    - Do not create separate “units” properties to describe numbers
      - Unless there are multiple values than cannot be converted (e.g. FanSpeed in RPM vs. PWM value)
  - Don’t represent bitfields as integers
    - Forces users to read other specifications to decode
    - Software has to convert to strings for display, “1” / “0” vs. “Enabled” / “Disabled”
- Strings - Use enumerations instead whenever possible
- Enumerations
  - Increases interoperability by specifying values instead of free-form text strings
  - Allows discovery (<property>@Redfish.AllowableValues) of valid settings
  - Additional enumerations can be added in future schema versions
- Use schema annotation to reduce user errors
  - Min/max ranges, pattern properties, units, etc.

Defining an embedded object vs. multiple properties

- If several properties go together, determine if they should be individual properties, or grouped into an embedded object
  - Embedded objects do create some additional burden on the consumer
  - Use them sparingly, but they have multiple applications
- Keep them as individual properties unless they hit any of the following:
  - If they start with the same name, put them in an object
  - If they are part of “polymorphism”: The resource has a “xxxType” property with enumerations and the properties do not apply to all of the types
  - If there are more than a few related properties
  - If the group of related properties is expected to expand over time
  - If a customer would expect them to “go together”
  - If you expect another implementation to leverage them (“inheritance by copy”)

Subordinate resources vs peer/referenced relationships

- Two kinds of links in Redfish resources
  - Both contain “@odata.id”, the OData equivalent of “href”
- Subordinate resources
  - Represent more details about the ‘parent’ resource or contain collections
    - Keep the data model as flat as practical (minimize I/Os)
  - Example: “Processors” contains additional details for ComputerSystem
  - Link appears in root of the resource for simplicity and to optimize for OData Expand
- Related resources
  - Show a relationship (peer or reference) to the ‘parent’ resource
  - Example: “ManagedBy” shows the Manager for a ComputerSystem
  - These are all contained in a “Links” embedded object to differentiate
  - Avoids first-level usage of OData Expand (where it would not be useful)
- Optimize definition assuming OData Expand will be implemented by service
  - Expand levels, when requested by client, allows subordinate resources to be included in the returned ‘parent’ payload, reducing the number of I/Os
  - Collection resources (or a subset of the resource) can be auto-expanded by service

Using Arrays and Collections

- Use arrays when:
  - Members will not come and go (empty array elements are “bad”)
  - Membership is fixed and finite
  - Ordinal value “natural” in context
  - Few interdependencies (reference) - links to members requires a JSON Pointer
  - Quantity too small to warrant a collection
  - Retrieving all members is a primary use case (like fans)
  - Consumer can access all members without harm (think permissions)
- Use collections when:
  - Members can be created or deleted or Provider will make them come and go.
  - There is no natural ordinal relationship
  - There will be interdependencies, peer references or external references
  - The member will have a lot of properties
  - A client will need to access members individually (think permissions)

Future Proof

- Assume Redfish will span management for software defined infrastructure
  - Craft models that favor the direction the industry is headed
  - Specification and schema should last a decade or more
  - Think about disaggregation, fabrics, composability
  - Do not preclude future technical growth
    - Improper modelling of common functionality makes it impossible to leverage in the future. Such modeling should be avoided.
  - Be technology agnostic in names and structure whenever possible
    - Allow leverage for the next “big thing”, which is probably similar to the current “big thing”. (e.g. “DiskSize” not “ScsiDiskSize”).
- Think about growth in Redfish as it expands in breadth and depth
  - Accommodate expansion of scope to switches, storage, power infrastructure and other data center attributes
    - Think about them even if you’re not defining them
- Enable OEM extensibility
  - Ensure that models accommodate the common OEM extension sections in the base property set as well as the link section.

Using and defining Actions

- In REST, Actions are accomplished through a series of operations
  - The “S” in REST stands for state (as in state machine)
  - But for complex operations, this becomes burdensome for the user
- Define Actions:
  - When atomicity is needed across multiple objects
    - We don’t have (or want) a commit/rollback mechanism. A single operation is much easier
  - When simplicity or efficiency is needed
    - If a log has 10,000 entries, deleting records individually is neither simple nor efficient
    - Don’t force consumer polling to track the machine state for long-duration state transitions
    - Example: “Reset” modeled using PATCH would require tracking the machine state (such as in a graceful shutdown), and also adding “current” and “requested” state
  - When the provider needs to control the operation
    - Exposing the gory details could prevent interoperability between implementations.
- But not for simple use cases:
  - Don’t use an action as a substitute for a POST to a collection. POSTs to create are allowed to have side affects to other resources in Redfish.
  - Don’t use an action when a PATCH would do, like setting an indicator LED.

Using Annotations in the model

- Redfish uses common properties for unification
  - Name, Id, Members, @odata.id, @odata.type, status, etc.
  - Remember “inheritance by copy”?
    - It is common to reuse properties by copying them or referencing them.
- Annotations provide a method for including functionality without a copy
  - Example is @Redfish.Settings which provides for settings and a way for clients to determine if settings take place dynamically
  - Non-Redfish annotations are precluded outside of OEM section
    - Only @Redfish and @odata annotations are allowed
- Use Annotations when:
  - Functionality can be added to multiple resource types that would otherwise complicate the schema
- Don’t use them for:
  - Mixing metadata in with payload data
  - Concepts that are not likely to expand to other areas of the model

Schema Creation

- Redfish supports two schema formats
  - Common Schema Definition Language (CSDL) as defined by OData v4
  - JSON-schema (www.json-schema.org) draft 4
- Both formats are normative and equivalent
  - Contents of the CSDL and JSON-schema version of a schema are the same (any inconsistency is an error)
  - Schema language semantics or capabilities may differ between the two formats, but functionality used by Redfish is equivalent
  - Schema writers are encouraged to publish in both formats
    - Both formats can be used for payload validation
    - JSON-schema is generally considered easier for humans to read
    - CSDL format is required for OData conformance and interoperability
- Tools to aid in schema creation and validation
  - SPMF is working on open source tools to create, transform and validate both CSDL and JSON-schema formats for Redfish schemas

Extending the Schema language (CSDL / JSON Schema)

- CSDL and JSON schema do not have the same “qualifiers”
  - These are called annotations in CSDL and keywords in JSON Schema
  - Thus Redfish extended CSDL & JSON Schema to create a common set.
    - Added to JSON Schema: readonly, requiredOnCreate, longDescription, copyright, enumDescriptions
    - Added to CSDL: AllowableValues, Required, RequiredOnCreate, Settings, AutoExpand, IPv6Format, PropertyPattern, Copyright, Minimum, Maximum
- Add schema language extensions judiciously
  - Existing mechanisms can define much of what is in MOF
  - Ideas that the SPMF has had for extensions:
    - ETag, CorrelatableID, Default, Dependency, ModelCorrespondence, Profile
- Other Extensions are disallowed by the Redfish Specification
  - So work with SPMF to get qualifiers defined

Pass thru data model

- In order to accommodate light-weight environments, one of the design goals was a “Pass Thru” data model.
  - Fixing up payloads with links puts a big burden on the web server implementation
- The “Providers” should be self contained
  - Should not reference items outside of their domain
  - This applies to properties, links, annotations and actions
- Except for @odata annotations for the specific resource, dependencies on the web server to fix-up payload are strongly discouraged
  - Don’t require the web server to fix-up links either in the root or in the links section, populate annotations or actions

Redfish
Ethernet Switch Management

Network Model – status of manageability

- Complex and disparate toolsets, protocols and systems
- Resource intensive and time consuming
- Proprietary vendor implementations
- Poor portability of skillsets across compute, storage and networking
- Lack of interoperability with rest of infrastructure

Proposal: Redfish models based on YANG models

- YANG is a model driven approach to network management
- Basis for general network industry manageability
  - IETF – YANG is the standard for network management modeling
  - IEEE – Adopted YANG as modeling language
  - Other consortiums and bodies have also adopted YANG for network models (e.g. OpenConfig, OpenDaylight, etc.)
- Large body of existing work
  - Extensive coverage from multiple SDOs
  - Many vendor proprietary YANG models
  - Many man-years of work by industry experts across all networking feature sets
- DMTF wants to leverage the networking industry's expertise

Why use Redfish for Managing Network?

- Completes the converged infrastructure management API story
  - Switches have platform components common to servers and storage
  - Rapid expansion of open Network Operating System (NOS) solutions
  - NFV will need common manageability for compute and networking
- Orchestrator systems can use a common interface for inventory and control
- Allows partnerships with networking standard orgs
  - Specify a prescriptive baseline of YANG models for network switch
  - Reduce overlap and clarify manageability domains

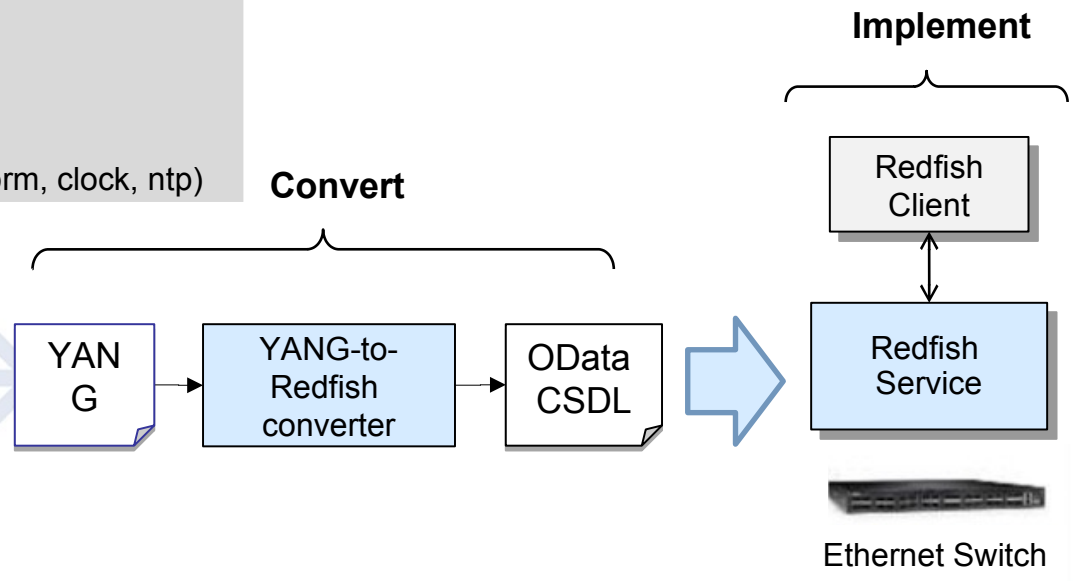
Network Switch Model

Convert from YANG models

- Phase 1 - convert a small set of YANG models to Redfish models
  - Proves the process, and validates the converter
  - dmtf.org/sites/default/files/standards/documents/DSP-IS0004_0.9a.zip
- Phase 2 – additional YANG models

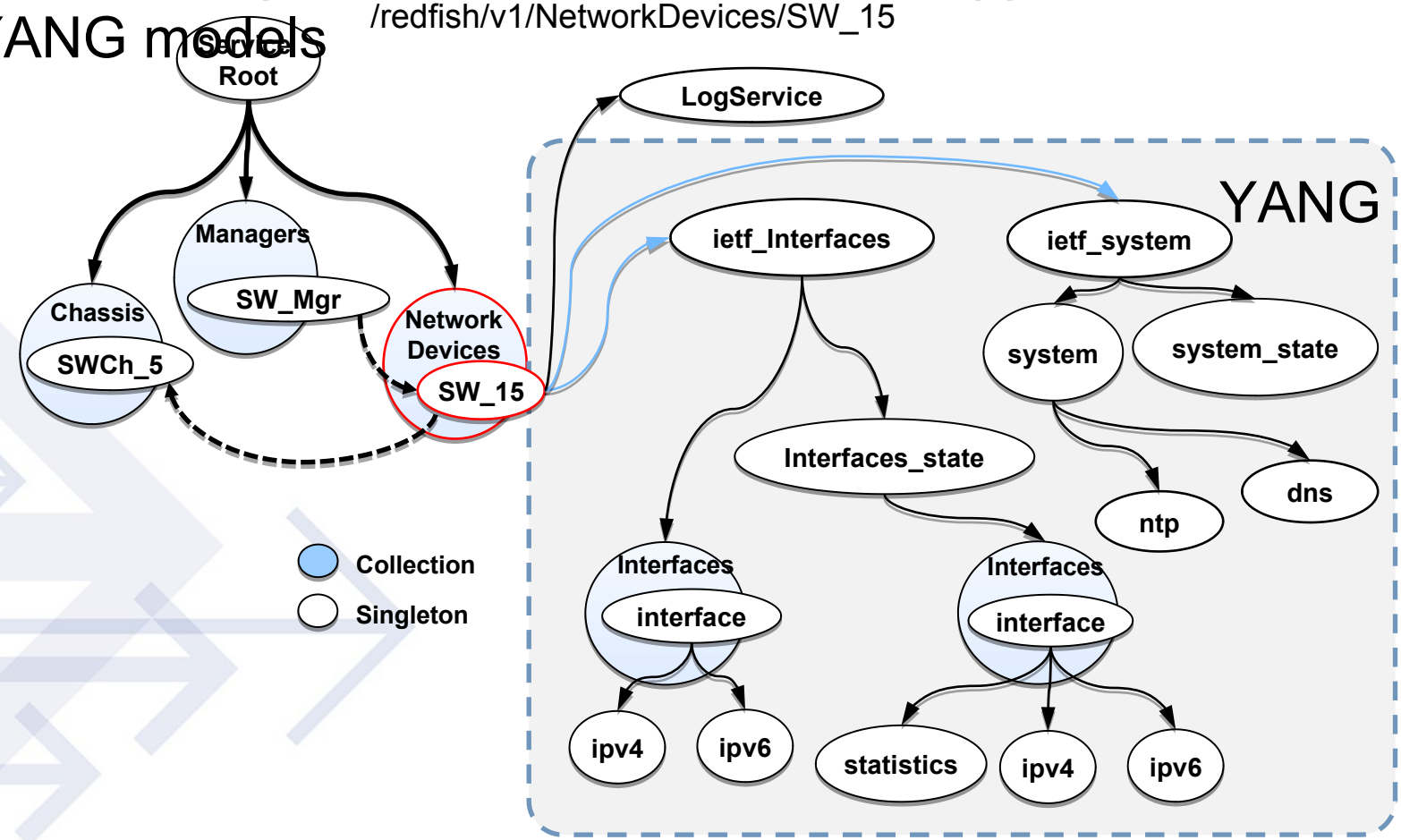
Ethernet Switch (Phase 1)

- RFC7223 (Interfaces)
- RFC7224 (IANA Interface types)
- RFC7277 (IPv4 and IPv6)
- RFC7317 (system, system_state, platform, clock, ntp)



The NetworkDevice Resource

The attachment point for Redfish models mapped from the YANG models



Mapping YANG-to-CSDL

YANG data model (RFC7223)

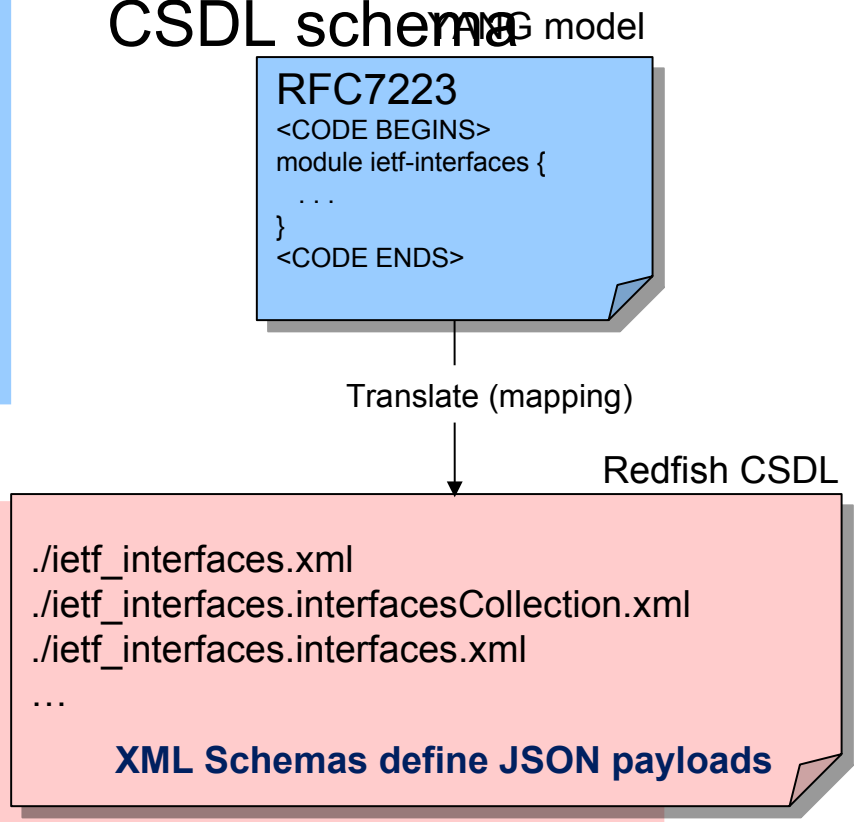
```
+--rw interfaces
| +--rw interface* [name]
| +--rw name string
| +--rw description? string
| +--rw type identityref
| +--rw enabled? boolean
| +--rw link-up-down-trap-enable? enumeration
+--ro interfaces-state
+--ro interface* [name]
+--ro name string
+--ro type identityref
+--ro admin-status enumeration
```

Redfish resource (GET response)

```
{
    "Id": "ethernet1",
    "Name": "ethernet1",
    "Description": "Ethernet interface on slot 1",
    "type": "iana_if_type:ethernetCsmacd",
    "enabled": "true",
    "link_up_down_trap_enable": "true"

    "@odata.context": "…",
    "@odata.type": "#interface_v1_0_0.interfaces",
    "@odata.id": "/redfish/v1/NetworkDevices/Switch1/ietf_interfaces/interfaces/ethernet1"
}
```

Translate YANG models to Redfish CSDL schema



Redfish Tool chains

1. Tools to enable Redfish modeling
2. Tools to enable Redfish clients
   - Ability for early client development
   - DMTF extending charter to allow contribution to external repositories