PLDM for Redfish Device Enablement Deep Dive v1.2

7 December 2018

Table of Contents

- Overview
- Discovery and Registration
  - Schemas and Redfish Resource PDRs
- Operations
- Advanced Topics: Binary Encoded JSON (BEJ)
  - Dictionary Preparation
  - Encoding Example
  - Decoding Example
- Advanced Topics: Tasks
- Advanced Topics: Eventing

Copyright 2018, DMTF

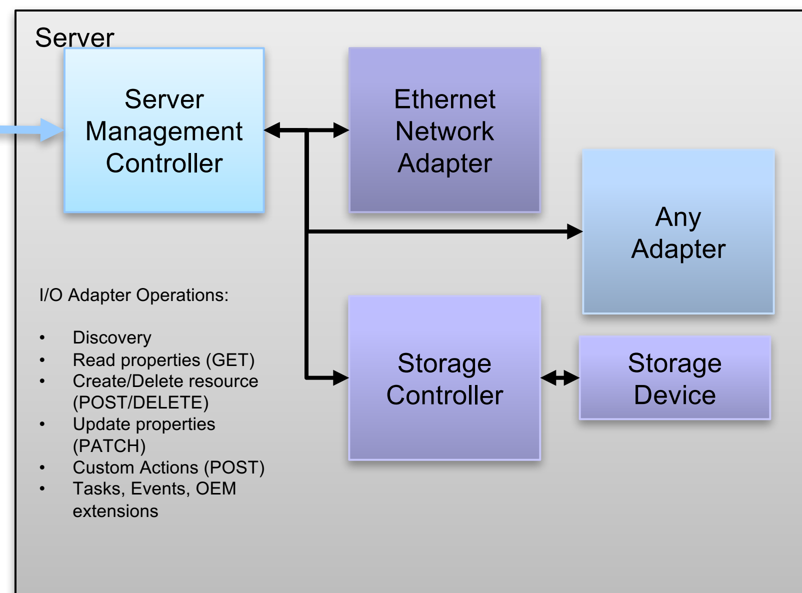
Redfish Resource Map (Simplified)

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

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

Copyright 2018, DMTF

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 proxy to encode/decode the data to/from the provider
- PLDM works over I2C & PCIe VDM. Additional mappings under consideration.

Copyright 2018, DMTF

Redfish Device Enablement (RDE)

- Standard Redfish
  - Transport for communication: HTTP/HTTPS
  - Payload for configuration: JSON
- The Challenge
  - HTTP servers require a lot of code and computation
  - JSON is a verbose protocol designed for human readability
    - Large message payloads, lots of space consumed in property names
    - Flexible layout: no defined order for properties
  - Devices have limited capabilities
    - Small memory footprint, limited processing capabilities
    - Can't handle raw JSON
  - Need a way to make the payloads manageable for devices

Copyright 2018, DMTF

RDE: The Solution

- Redfish Device Enablement
  - Make the cat smaller!
  - Transport for communication: PLDM
  - Payload for configuration: BEJ
- PLDM
  - Platform level data model, a low level messaging system that runs over a variety of system level buses
    - I2C/SMBus and PCIeVDM over MCTP
    - RBT over NC-SI
  - Supports topologies, eventing, discovery
- Binary Encoded JSON (BEJ)
  - Replace long identifier strings in JSON text with short sequence numbers
  - Dictionary enables lookup between sequence numbers and strings
  - Roughly a 10x compression!

Copyright 2018, DMTF

PMCI Hierarchy for Redfish Services

PLDM Layers
- SMBIOS
- Monitoring and Control
- BIOS Ctrl and Config
- FRU Data
- Firmware Update
- Redfish Device Enablement

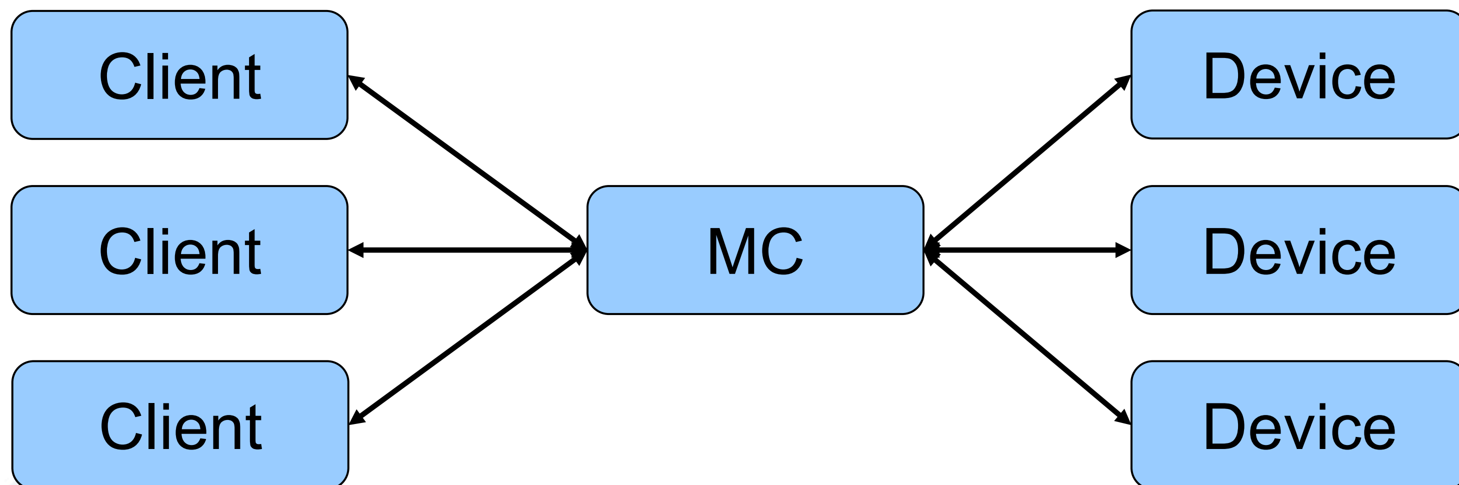
Upper Layers
- NVMe Mgmt I/F
- PLDM
- MCTP Ctrl
- Network Controller Sideband Interface (NC-SI)

Transport Layers
- Management Component Transport Protocol (MCTP)
  - MCTP/ SMBus
  - MCTP/ Serial
  - MCTP/ PCIe VDM
- RMII Based Transport (RBT)

Physical Layers
- SMBus
- Serial
- PCIe VDM
- RMII

Copyright 2018, DMTF

RDE Architecture



- Roles
  - Client: uses standard JSON over HTTP/S
  - Management Controller (MC): translates between client and device
  - Device: hosts data, uses BEJ over PLDM
- RDE splits Redfish support between the device and the MC
- Client unaware that device does not talk native Redfish
  - Ensures compatibility and supportability

Copyright 2018, DMTF

RDE Operations

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

Copyright 2018, DMTF

The RDE API – Discovery and Registration

Command	Usage
NegotiateRedfishParameters	Concurrency and feature support
NegotiateChannelParameters	Asynchrony support and chunk transfer size
GetSchemaDictionary	Dictionary retrieval
GetSchemaURI	Formal schema identification
GetSchemaInstanceETag	Get a digest of schema data

Copyright 2018, DMTF

The RDE API – Operations

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

Copyright 2018, DMTF

Next Up: Discovery and Registration

- Overview
- Discovery and Registration
  - Schemas and Redfish Resource PDRs
- Operations
- Advanced Topics: Binary Encoded JSON (BEJ)
  - Dictionary Preparation
  - Encoding Example
  - Decoding Example
- Advanced Topics: Tasks
- Advanced Topics: Eventing

Copyright 2018, DMTF

RDE Device Discovery

1. MC detects presence of device and that it supports PLDM (such as via MCTP Get Message Type Support command)
2. MC confirms that device supports both Monitoring and Control (type 2) and Redfish Device Enablement (type 6) via PLDM GetPLDMTypes command
3. MC confirms that device supports appropriate PLDM base, M&C, and Redfish commands via PLDM GetPLDMCommands command
4. MC uses RDE NegotiateRedfishParameters command to perform initial handshake with device, resolving:
   a) Concurrency: How many pending operations can the device manage at once?
   b) Supported Operations
   c) Device Redfish provider name
5. For each communication channel the MC wants to talk to the device on, it uses the RDE NegotiateChannelParameters command, resolving:
   a) Asynchrony support: Will the device issue asynchronous alerts for Operation completion or Redfish Events?
   b) Maximum transfer chunk size: how much data can be sent at once?

Copyright 2018, DMTF

RDE Device Registration

1. MC queries device for its Platform Data Records (PDRs)
   a. PLDM GetPDRRepositoryInfo command gets total number of PDRs
   b. PLDM GetPDR commands retrieve each Redfish Resource PDR
   c. Redfish Entity Association PDRs capture some logical links between resources
   d. Redfish Action PDRs detail custom actions supported by device
2. MC gets schema IDs and versions from the Redfish Resource PDRs
   a. Retrieving just the identities (names) of the schemas, not the full textual schema contents.
   b. Can retrieve binary dictionary to get translation info for schema via GetSchemaDictionary command
   c. Dictionary includes standard schema data and OEM extensions
   d. Dictionaries may be stripped down to list just those fields that the device supports
3. MC offers management data (by proxy) for the device
   a. MC acts as translating provider on behalf of device
   b. External client management utility ("client") won't know it isn't talking directly to the device
4. Client locates MC provider via standard Redfish techniques

Copyright 2018, DMTF

Next Up: Schemas and PDRs

- Overview
- Discovery and Registration
  - Schemas and Redfish Resource PDRs
- Operations
- Advanced Topics: Binary Encoded JSON (BEJ)
  - Dictionary Preparation
  - Encoding Example
  - Decoding Example
- Advanced Topics: Tasks
- Advanced Topics: Eventing

Copyright 2018, DMTF

Redfish Resource PDRs

- New PDR type to encapsulate the data for a resource
- Each PDR represents one or more instances of the data
  - Many-to-one relationship between PDRs and schemas
  - Many-to-one relationship between PDRs and resources
- Device lines up schemas to correspond to the management topology it wants to expose
- Topology determined jointly by device and MC
  - MC exposes outer (top levels) of the topology
  - Device builds component trees that hook into the structure built by the MC
  - Component trees are linked internally via the containerID field that identifies parent for each resource. Parent of SYSTEM means it hooks to the MC's structure
  - Device suggests where to place its component trees, but MC gets final say
- Formal URIs for schemas can be retrieved via GetSchemaURI command

Redfish Resource PDR

resourceID = 1

containingResourceID = SYSTEM

version = F1F0F0FF

schemaName = EthernetInterface

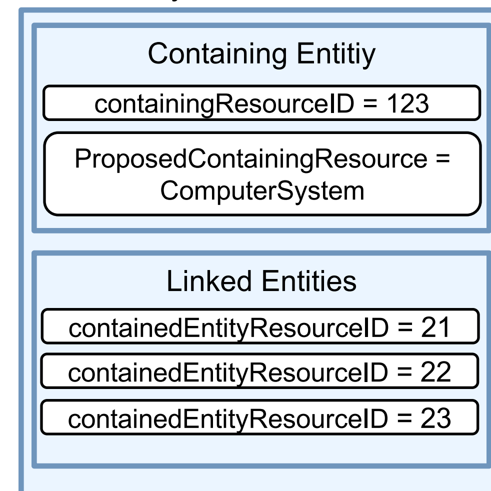
ProposedContainingResource = ComputerSystem

schemaURI = redfish.dmtf.org/Schemas/v1/redfish-schema.v1_1_0

Copyright 2018, DMTF

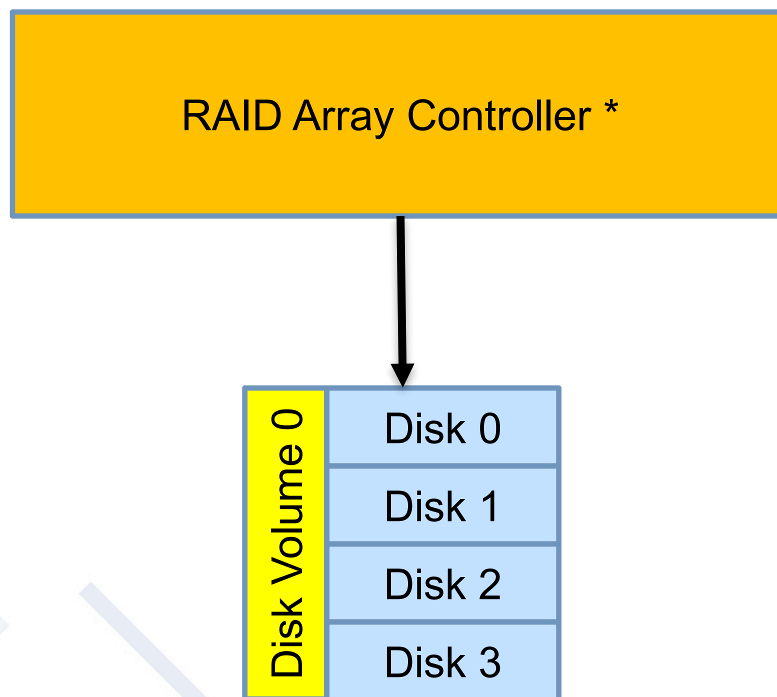
Redfish Entity Association PDRs

- Usually not needed since linking is implicit in Redfish Resource PDRs
- Links between resources internal to devices can just be supplied in response to reads via the Links section of the resource schema
- Primarily needed when a resource inside the device is a related resource for a resource outside the device
- Connection to MC's resource topology works same as for Redfish Resource PDR



Copyright 2018, DMTF

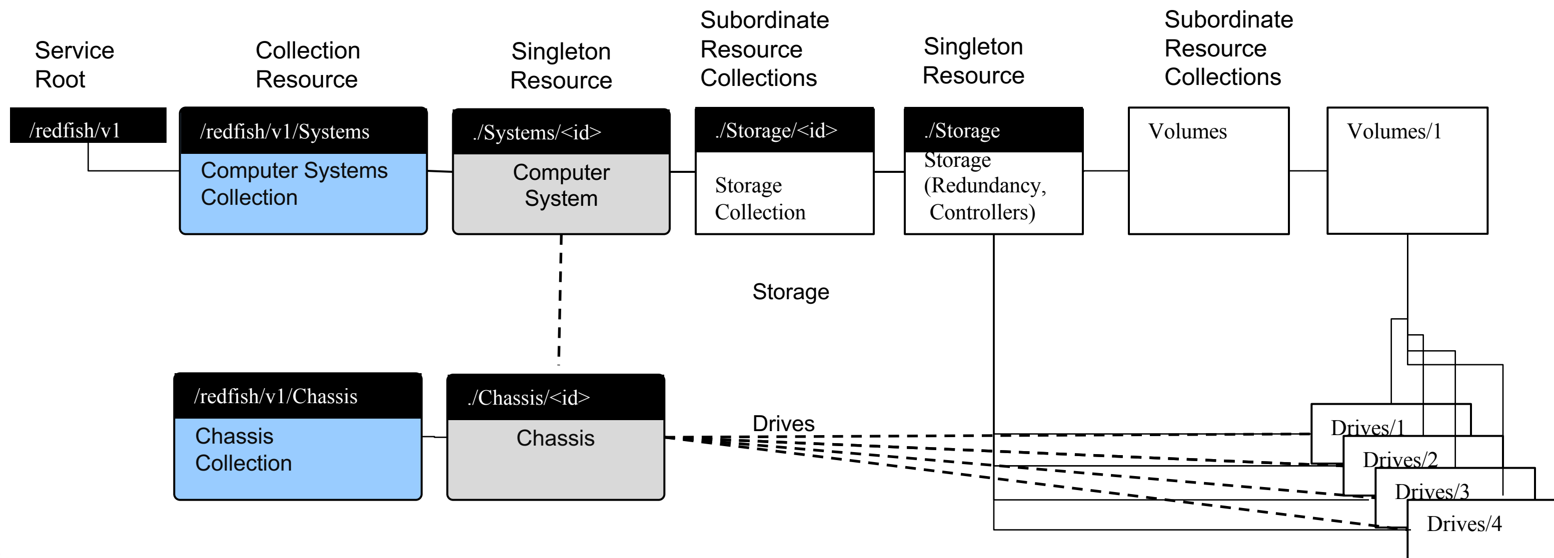
Disk Subsystem (Physical Layout)



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

Copyright 2018, DMTF

Server Storage in Redfish

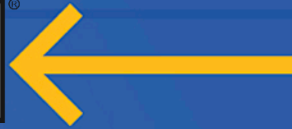


Note that the Volumes are in Collections off of the Storage resource, drives are in arrays off of the storage resource and optionally the Chassis.

Copyright 2018, DMTF

Disk Subsystem: Linking Redfish Resource PDRs

Copyright 2018, DMTF



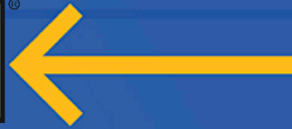
Linking without Redfish Entity Association PDRs

Copyright 2018, DMTF

Next Up: Operations

- Overview
- Discovery and Registration
  - Schemas and Redfish Resource PDRs
- Operations
- Advanced Topics: Binary Encoded JSON (BEJ)
  - Dictionary Preparation
  - Encoding Example
  - Decoding Example
- Advanced Topics: Tasks
- Advanced Topics: Eventing

Copyright 2018, DMTF



RDE Operations

0 - Inactive

1 - Need Input

2 - Triggered

3 - Task Running

4- Have Results

5 - Completed

6 - Abandoned

7 - Failed

Any of {1,2,3,4}

Any of {1,2,3,4}

RDEOperationComplete

RDEOperationInit, no or inlined payload, no params, no or inlined payload, no params

RDEOperationInit, no or inlined payload, no params, result payload or params

RDEOperationInit, non inlined payload or params

MultipartSend or SupplyCustomRequestParameters; more input pending

RDEOperationComplete

RDEOperationInit, no or inlined payload, no params, Task spawned

MultipartSend, last chunk, no params or params sent

Inputs supplied, Task spawned

(‡)

MC fails to advance Operation

Error occurs while executing Operation

RDEOperationStatus, non inlined payload or response params

Inputs supplied, Operation finished, results available

(†)

MultipartReceive or RetrieveCustomResponseParameters; more results pending

Inputs supplied, Operation finished, no or inlined payload, no response params

MultipartReceive or RetrieveCustomResponseParameters, no more results pending

RDEOperationStatus, Operation finished, no or inlined payload, no response params

RDEOperationStatus, Operation finished, no or inlined payload, no response params

RDEOperationComplete

(†): RDEOperationStatus, Operation finished, non-inlined payload or response params

(‡): RDEOperationStatus, Task spawned

Copyright 2018, DMTF

Next Up: Binary Encoded JSON

- Overview
- Discovery and Registration
  - Schemas and Redfish Resource PDRs
- Operations
- Advanced Topics: Binary Encoded JSON (BEJ)
  - Dictionary Preparation
  - Encoding Example
  - Decoding Example
- Advanced Topics: Tasks
- Advanced Topics: Eventing

Copyright 2018, DMTF

Binary Encoded JSON (BEJ)

- Compact binary format for self-describing encoding of multiformat data
- Separates the string labels of the data from the data itself
  - Property names and enumeration strings are static, don't need to be part of the encoding as long as we can systematically restore them later
  - We'll put them into a separate "dictionary" and refer to them by "sequence numbers"
  - Requires that sequence numbers be well-defined. JSON not ordered, so we'll have to impose a canonical ordering on it
- Comparable to ASN.1, but simpler to deal with
  - Providing sequence number AND format for each tuple eliminates need for context-aware decoding
  - Adding counts for sets and arrays enables preallocation of memory for decoded contents without requiring an additional pass through the data
  - Less state required to decode
  - Directly tied to JSON, so we can skip some of the exotic ASN.1 formats and reuse high bits in the format byte as flags
  - Easier to implement: 2 days for BEJ encoder/decoder vs a month for ASN.1

Copyright 2018, DMTF

BEJ and Redfish Schemas

- BEJ encoding details for individual Redfish schemas can be captured in dictionaries
  - Everyone has to agree to use the same dictionaries or this doesn't work
  - Standardized procedure for generating dictionaries ensures that everyone works with the same dictionary
  - Code to create dictionaries will be created and made open source
  - Dictionaries can be made publicly available on DMTF publicly available website
  - Dictionaries can be defined once, when schemas are defined
    - By DMTF in the case of standard schemas
    - By OEMs in the case of OEM schemas
  - Dictionaries are static: once revisions are defined, no runtime changes
- Dictionaries do not cross Redfish schema boundaries
  - 1:1 mapping from dictionaries to Redfish schemas
  - Data for distinct schemas is stored in distinct dictionaries
- Dictionaries are consistent from version to version of Redfish schemas by design of sequence number assignment
- Dictionaries are never exposed to Redfish clients; they are internal between devices and MCs

Copyright 2018, DMTF

Building Dictionaries from Redfish Schemas for BEJ

- Example: Dummy Simple Schema (artificial, fits in slide)
- Prep work: done before or as part of compiling MC/device firmware

1. Canonize Redfish schema
   a. Expand reference properties to get all schema data into one place
   b. External schema links get placeholders (that ultimately will map to other PDRs)
   c. Put in alphabetical order (by rev) within each layer of the hierarchy
      a. fields/enum values added for initial version of the Redfish schema are sorted
      b. fields/enum values added for each subsequent version are sorted and *appended* to the list from the previous
      c. sequence numbers for fields/enum values removed from previous versions of the Redfish schema are never reused
      d. this ensures that sequence numbers never change across versions over the life of a Redfish schema
   d. Insert sequence number annotations
      a. in increasing order within each level of the hierarchy, starting at 0
2. Collect canonized schema data into dictionary for use with BEJ

Copyright 2018, DMTF

Basic Schema (translated to a compact format)

```
<Schema url="http://redfish.dmtf.org/.../redfish-schema.v1_1_0.json"
title="DummySimpleSchema" version="1.2.0">
   <Item name="DummySimple" type="set">
      <Link name="Id" ref="http://redfish.dmtf.org/.../definitions/Id"/>
      <Item name="SampleIntegerProperty" type="number" readonly="no" optional="no"/>
      <Item name="SupportEnabledProperty" type="boolean" readonly="yes" optional="yes"/>
      <Item name="ChildArrayProperty" type="array">
            <Link name="" ref="http://redfish.dmtf.org/.../definitions/ChildArray"/>
      </Item>
   </Item>
</Schema>

<Schema url="http://redfish.dmtf.org/schemas/v1/Resource.json#/definitions/Id"
title="IdSchema" version="1.0.0">
   <Item name="Id" type="string" readonly="yes" optional="no"/>
</Schema>

<Schema url="http://redfish.dmtf.org/.../redfish-schema.v1_1_0.json"
title="ChildArraySchema" version="1.0.0">
   <Item name="ChildArray" type="set">
      <Item name="LinkStatus" type="enum" enumtype="String" readonly="yes" optional="no">
         <Enumeration value="LinkUp"/>
         <Enumeration value="NoLink"/>
         <Enumeration value="LinkDown"/>
      </Item>
      <Item name="AnotherBoolean" type="boolean" readonly="yes" optional="yes"/>
</Schema>
```


Copyright 2018, DMTF

Schema With Reference Links Expanded

```
<Schema url=“.../redfish-schema.v1_1_0.json” title=“DummySimpleSchema” version=“1.2.0”>
   <Item name=“DummySimple” type=“set”>
      <Item name=“Id” type=“string” readonly=“yes” optional=“no”/>
      <Item name=“SampleIntegerProperty” type=“number” readonly=“no” optional=“no”/>
      <Item name=“SupportEnabledProperty” type=“boolean” readonly=“yes” optional=“yes”/>
      <Item name=“ChildArrayProperty” type=“array” arraytype=“set”>
         <Item name=“array element 0”>
            <Item name="LinkStatus" type="enum" enumtype="String" readonly="yes" optional="no">
               <Enumeration value="LinkUp"/>
               <Enumeration value="NoLink"/>
               <Enumeration value="LinkDown"/>
            </Item>
            <Item name=“AnotherBoolean” type=“boolean” readonly=“yes” optional=“yes”/>
         </Item>
         <Item name=“array element 1”>
            <Item name="LinkStatus" type="enum" enumtype="String" readonly="yes" optional="no">
               <Enumeration value="LinkUp"/>
               <Enumeration value="NoLink"/>
               <Enumeration value="LinkDown"/>
            </Item>
            <Item name=“AnotherBoolean” type=“boolean” readonly=“yes” optional=“yes”/>
         </Item>
         <Item name=“array element 2”>
            ...
         </Item>
      </Item>
   </Item>
</Schema>
```


Copyright 2018, DMTF

Alphabetized Schema (red = lines moved in sorting)

```
<Schema url="…/redfish-schema.v1_1_0.json" title="DummySimpleSchema" version="1.2.0">
   <Item name="DummySimple" type="set">
      <Item name="ChildArrayProperty" type="array" arraytype="set">
         <Item name="array element 0">
            <Item name="AnotherBoolean" type="boolean" readonly="yes" optional="yes"/>
            <Item name="LinkStatus" type="enum" enumtype="String" readonly="yes" optional="no">
               <Enumeration value="LinkDown"/>
               <Enumeration value="LinkUp"/>
               <Enumeration value="NoLink"/>
            </Item>
          </Item>
          <Item name="array element 1">
            <Item name="AnotherBoolean" type="boolean" readonly="yes" optional="yes"/>
            <Item name="LinkStatus" type="enum" enumtype="String" readonly="yes" optional="no">
               <Enumeration value="LinkDown"/>
               <Enumeration value="LinkUp"/>
               <Enumeration value="NoLink"/>
             </Item>
          </Item>
          <Item name="array element 2">
             ...
          </Item>
      </Item>
      <Item name="Id" type="string" readonly="yes" optional="no"/>
      <Item name="SampleIntegerProperty" type="number" readonly="no" optional="no"/>
      <Item name="SupportEnabledProperty" type="boolean" readonly="yes" optional="yes"/>
   </Item>
</Schema>
```


Copyright 2018, DMTF

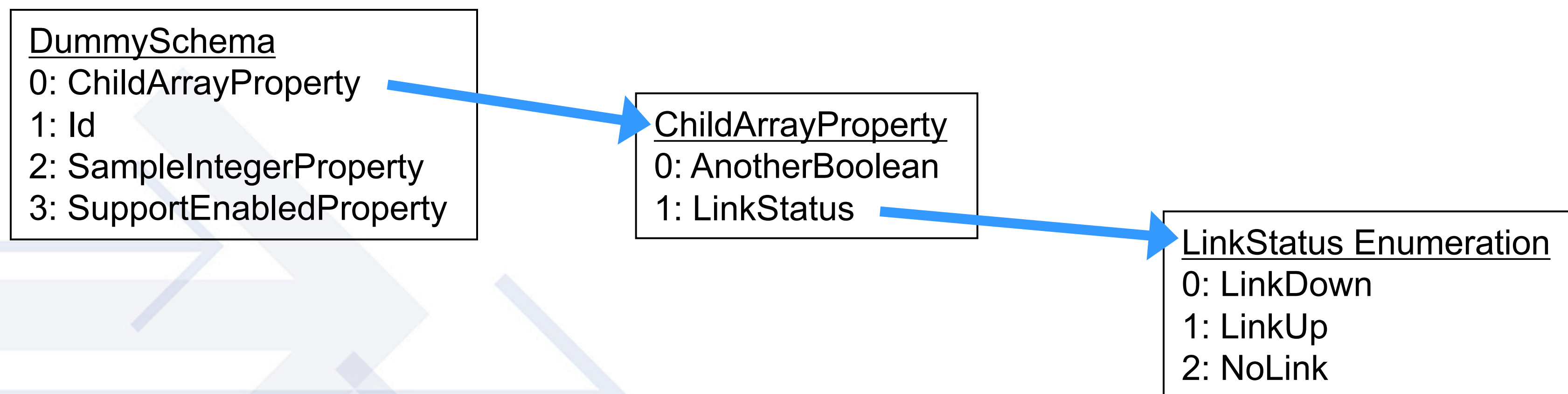
Sequence Number Annotations (red = added seq nos)

```
<Schema url="…/redfish-schema.v1_1_0.json" title="DummySimpleSchema" version="1.2.0">
   <Item name="DummySimple" type="set" seq="0">
      <Item name="ChildArrayProperty" type="array" arraytype="set" seq="0">
         <Item name="array element 0" seq="0">
            <Item name="AnotherBoolean" type="boolean" readonly="yes" optional="yes" seq="0"/>
            <Item name="LinkStatus" type="enum" enumtype="String" readonly="yes" optional="no" seq="1">
               <Enumeration value="LinkDown" seq="0"/>
               <Enumeration value="LinkUp" seq="1"/>
               <Enumeration value="NoLink" seq="2"/>
            </Item>
          </Item>
          <Item name="array element 1" seq="1">
            <Item name="AnotherBoolean" type="boolean" readonly="yes" optional="yes" seq="0"/>
            <Item name="LinkStatus" type="enum" enumtype="String" readonly="yes" optional="no" seq="1">
               <Enumeration value="LinkDown" seq="0"/>
               <Enumeration value="LinkUp" seq="1"/>
               <Enumeration value="NoLink" seq="2"/>
            </Item>
          </Item>
          <Item name="array element 2" seq="2">
             ...
          </Item>
      </Item>
      <Item name="Id" type="string" readonly="yes" optional="no" seq="1"/>
      <Item name="SampleIntegerProperty" type="number" readonly="no" optional="no" seq="2"/>
      <Item name="SupportEnabledProperty" type="boolean" readonly="yes" optional="yes" seq="3"/>
   </Item>
</Schema>
```


Copyright 2018, DMTF

String/Sequence Number Lookup Tables

- Extract the strings and their corresponding sequence numbers into tabular form
- Each row has the sequence number, the string, and a pointer to a dependent table (null if none)



Copyright 2018, DMTF

Building Schema Dictionaries from Lookup Tables

- Just need data types, output format beyond the data already in the tables

Row	Seq	Format	Name	ChildPtr
0	0	set	DummySchema	1
1	0	array	ChildArrayProperty	5
2	1	string	Id	-
3	2	boolean	SampleEnabledProperty	-
4	3	Integer	SampleIntegerProperty	-
5	0	boolean	AnotherBoolean	-
6	1	enum|string	LinkStatus	7
7	0	string	LinkDown	-
8	1	string	LinkUp	-
9	2	string	NoLink	-

Copyright 2018, DMTF

Performing a Simple Read

1. Client sends read command to MC
   a. Command looks something like: GET /redfish/v1/systems/1/DummySchema
   b. Part in blue is how the MC decided to expose the device – everything up to the device root
   c. Part in red is controlled by the management topology exported from the device (as the MC decides to offer it)
2. MC identifies the Redfish Resource PDR on the device associated with the device-level portion of the target URI
3. MC uses the PLDM RDEOperationInit command to begin the Operation, specifying the PDR it identified previously and characterizing the Operation as a read
4. Device performs BEJ encoding of resource data
5. Device responds to the RDEOperationInit command signaling success. If the BEJ payload is small enough, the device includes it with the response message; otherwise it gives a transfer handle for the MC to use with MultipartReceive
6. MC uses the RDEOperationComplete command to finalize the Operation. Device frees resources associated with it
7. MC decodes response (using dictionary), to JSON, returns it to the client

Copyright 2018, DMTF

Next Up: Encoding Example

- Overview
- Discovery and Registration
  - Schemas and Redfish Resource PDRs
- Operations
- Advanced Topics: Binary Encoded JSON (BEJ)
  - Dictionary Preparation
  - Encoding Example
  - Decoding Example
- Advanced Topics: Tasks
- Advanced Topics: Eventing

Copyright 2018, DMTF

Encoding Data: Overview

1. Replace strings with sequence numbers
2. Dump optional data not present
3. Convert to {SFLV} tuples
    a. Insert counts for sets and arrays
    b. Use placeholders (for now) for lengths
4. Encode formats, counts, leaf values
5. Fill in lengths working from innermost to outermost

Copyright 2018, DMTF

Encoding Example: Raw Data

```
<Item name="DummySimple" type="set">
   <Item name="ChildArrayProperty" type="array">
      <Item name="array element 0">
         <Item name="AnotherBoolean" type="boolean" value="true"/>
         <Item name="LinkStatus" type="enum" enumtype="String">
            <Enumeration value="NoLink">
         </Item>
      </Item>
      <Item name="array element 1">
         <Item name="AnotherBoolean" optional="yes" NOT PRESENT/>
         <Item name="LinkStatus" type="enum" enumtype="String">
            <Enumeration value="LinkDown"/>
          </Item>
       </Item>
   </Item>
   <Item name="Id" type="string" value="Dummy ID"/>
   <Item name="SampleIntegerProperty" type="number" value="12"/>
   <Item name="SampleEnabledProperty" type="boolean" optional="yes" NOT PRESENT/>
</Item>
```


Copyright 2018, DMTF

Encoding Example: Replacing Strings with Seq Nos

(Low-order bit of sequence number is a dictionary selector tag; here, always 0)

```
<Item type="set" seq="0">
   <Item type="array" seq="0">
      <Item seq="0">
         <Item type="boolean" value="true" seq="0"/>
         <Item type="enum" enumtype="String" seq="2">
            <Enumeration value="(value 2)"/>
         </Item>
      </Item>
      <Item seq="2">
         <Item optional="yes" NOT PRESENT seq="0"/>
         <Item type="enum" enumtype="String" seq="2">
            <Enumeration value="(value 0)"/>
          </Item>
       </Item>
   </Item>
   <Item type="string" value="Dummy ID" seq="2"/>
   <Item type="boolean" optional="yes" NOT PRESENT seq="4"/>
   <Item type="number" value="12" seq="6"/>
</Item>
```


Copyright 2018, DMTF

Encoding Example: Dumping Non-present Data

```
<Item type="set" seq="0">
   <Item type="array" seq="0">
      <Item seq="0">
         <Item type="boolean" value="true" seq="0"/>
         <Item type="enum" enumtype="String" seq="2">
            <Enumeration value="(value 2)"/>
         </Item>
      </Item>
      <Item seq="2">
          <Item type="enum" enumtype="String" seq="2">
            <Enumeration value="(value 0)"/>
          </Item>
       </Item>
   </Item>
   <Item type="string" value="Dummy ID" seq="2"/>
   <Item type="number" value="12" seq="6"/>
</Item>
```


Copyright 2018, DMTF

Encoding Example: {SFLV} Conversions; Insert Counts for Sets and Arrays

```
{0x01 0x00, set, [length placeholder], {
    {count = 3}
    {0x01 0x00, array, [length placeholder], {
        {count = 2}
        {0x01 0x00, set, [length placeholder], {
            {count = 2}
            {0x01 0x00, boolean, [length placeholder], true}
            {0x01 0x02, enum, [length placeholder], 2}
        }
        {0x01 0x02, set, [length placeholder], {
            {count = 1}
            {0x01 0x02, enum, [length placeholder], 0}
        }
    }
    {0x01 0x02, string, [length placeholder], "Dummy ID"}
    {0x01 0x06, integer, [length placeholder], 12}
}
```


Copyright 2018, DMTF

Example Encoding: Encoding of Formats, Counts, Leaf Values

```
{0x01 0x00, 0x00, [length placeholder], {
   0x01 0x03
   {0x01 0x00, 0x01, [length placeholder], {
      0x01 0x02
      {0x01 0x00, 0x00, [length placeholder], {
         0x01 0x02
         {0x01 0x00, 0x07, [length placeholder], 0xFF}
         {0x01 0x02, 0x04, [length placeholder], 0x01 0x02}
      }
      {0x01 0x02, 0x00, [length placeholder], {
         0x01 0x01
         {0x01 0x02, 0x04, [length placeholder], 0x01 0x00}
      }
   }
   {0x01 0x02, 0x05, [length placeholder],
       0x44 0x75 0x6D 0x6D 0x79 0x20 0x49 0x44 0x00}
   {0x01 0x06, 0x03, [length placeholder], 0x0C}
}
```


Copyright 2018, DMTF

Example Encoding: Filling in Lengths at Leaves

```
{0x01 0x00, 0x00, [length placeholder], {
   0x01 0x03
   {0x01 0x00, 0x01, [length placeholder], {
      0x01 0x02
      {0x01 0x00, 0x00, [length placeholder], {
         0x01 0x02
         {0x01 0x00, 0x07, 0x01 0x01, 0xFF}
         {0x01 0x02, 0x04, 0x01 0x02, 0x01 0x02}
      }
      {0x01 0x02, 0x00, [length placeholder], {
         0x01 0x01
         {0x01 0x02, 0x04, 0x01 0x02, 0x01 0x00}
      }
   }
   {0x01 0x02, 0x05, 0x01 0x09,
      0x44 0x75 0x6D 0x6D 0x79 0x20 0x49 0x44 0x00}
   {0x01 0x06, 0x03, 0x01 0x01, 0x0C}
}
```


Copyright 2018, DMTF

Example Encoding: Working Outward to Fill in Lengths

```
{0x01 0x00, 0x00, [length placeholder], {
   0x01 0x03
   {0x01 0x00, 0x01, [length placeholder], {
      0x01 0x02
      {0x01 0x00, 0x00, 0x01 0x0F, {
         0x01 0x02
         {0x01 0x00, 0x07, 0x01 0x01, 0xFF}
         {0x01 0x02, 0x04, 0x01 0x02, 0x01 0x02}
      }
      {0x01 0x02, 0x00, 0x01 0x09, {
         0x01 0x01
         {0x01 0x02, 0x04, 0x01 0x02, 0x01 0x00}
      }
   }
   {0x01 0x02, 0x05, 0x01 0x09,
      0x44 0x75 0x6D 0x6D 0x79 0x20 0x49 0x44 0x00}
   {0x01 0x06, 0x03, 0x01 0x01, 0x0C}
}
```


Copyright 2018, DMTF

Example Encoding: Outermost Lengths

```
{0x01 0x00, 0x00, 0x01 0x3F, {
    0x01 0x03
    {0x01 0x00, 0x01, 0x01 0x24, {
        0x01 0x02
        {0x01 0x00, 0x00, 0x01 0x0F, {
            0x01 0x02
            {0x01 0x00, 0x07, 0x01 0x01, 0xFF}
            {0x01 0x02, 0x04, 0x01 0x02, 0x01 0x02}
        }
        {0x01 0x02, 0x00, 0x01 0x09 {
            0x01 0x01
            {0x01 0x02, 0x04, 0x01 0x02, 0x01 0x00}
        }
    }
    {0x01 0x02, 0x05, 0x01 0x09,
         0x44 0x75 0x6D 0x6D 0x79 0x20 0x49 0x44 0x00}
    {0x01 0x06, 0x03, 0x01 0x01, 0x0C}
}
```


Copyright 2018, DMTF

Final Encoded Data: 68 bytes

```
0x01 0x00 0x00 0x01  0x3F 0x01 0x03 0x01
0x00 0x01 0x01 0x24  0x01 0x02 0x01 0x00
0x00 0x01 0x0F 0x01  0x02 0x01 0x00 0x07
0x01 0x01 0xFF 0x01  0x02 0x04 0x01 0x02
0x01 0x02 0x01 0x02  0x00 0x01 0x09 0x01
0x01 0x01 0x02 0x04  0x01 0x02 0x01 0x00
0x01 0x02 0x05 0x01  0x09 0x44 0x75 0x6D
0x6D 0x79 0x20 0x49  0x44 0x00 0x01 0x06
0x03 0x01 0x01 0x0C
```


Copyright 2018, DMTF

Next Up: Decoding Example

- Overview
- Discovery and Registration
  - Schemas and Redfish Resource PDRs
- Operations
- Advanced Topics: Binary Encoded JSON (BEJ)
  - Dictionary Preparation
  - Encoding Example
  - Decoding Example
- Advanced Topics: Tasks
- Advanced Topics: Eventing

Copyright 2018, DMTF

Decoding the Data: Overview

1. Map onto {SFLV} tuples, using counts and lengths to identify where the chunks of data lie
2. Throw away counts and lengths
3. Decode format info
4. Decode value info
5. Use sequence numbers to restore strings and enum values
6. Write out as JSON

Copyright 2018, DMTF

Decoding Example: Raw Data

```
0x01 0x00 0x00 0x01  0x3F 0x01 0x03 0x01
0x00 0x01 0x01 0x24  0x01 0x02 0x01 0x00
0x00 0x01 0x0F 0x01  0x02 0x01 0x00 0x07
0x01 0x01 0xFF 0x01  0x02 0x04 0x01 0x02
0x01 0x02 0x01 0x02  0x00 0x01 0x09 0x01
0x01 0x01 0x02 0x04  0x01 0x02 0x01 0x00
0x01 0x02 0x05 0x01  0x09 0x44 0x75 0x6D
0x6D 0x79 0x20 0x49  0x44 0x00 0x01 0x06
0x03 0x01 0x01 0x0C
```


Copyright 2018, DMTF

Decoding Example: Map to {SFLV} Tuples with Counts for Sets and Arrays (F=0x01 or 0x02)

```
{0x01 0x00, 0x00, 0x01 0x3F, {
    0x01 0x03
    {0x01 0x00, 0x01, 0x01 0x24, {
        0x01 0x02
        {0x01 0x00, 0x00, 0x01 0x0F, {
            0x01 0x02
            {0x01 0x00, 0x07, 0x01 0x01, 0xFF}
            {0x01 0x02, 0x04, 0x01 0x02, 0x01 0x02}
        }
        {0x01 0x02, 0x00, 0x01 0x09 {
            0x01 0x01
            {0x01 0x02, 0x04, 0x01 0x02, 0x01 0x00}
        }
    }
    {0x01 0x02, 0x05, 0x01 0x09,
        0x44 0x75 0x6D 0x6D 0x79 0x20 0x49 0x44 0x00}
    {0x01 0x06, 0x03, 0x01 0x01, 0x0C}
}
```


Copyright 2018, DMTF

Decoding Example: Throw Away Counts and Lengths

```
{0x01 0x00, 0x00, {
    {0x01 0x00, 0x01, {
        {0x01 0x00, 0x00, {
            {0x01 0x00, 0x07, 0xFF}
            {0x01 0x02, 0x04, 0x01 0x02}
        }
        {0x01 0x02, 0x00, {
            {0x01 0x02, 0x04, 0x01 0x00}
        }
    }
    {0x01 0x02, 0x05,
        0x44 0x75 0x6D 0x6D 0x79 0x20 0x49 0x44 0x00}
    {0x01 0x06, 0x03, 0x0C}
}
```


Copyright 2018, DMTF

Decoding Example: Decode Format Info

```
{0x01 0x00, set, {
    {0x01 0x00, array, {
        {0x01 0x00, set, {
            {0x01 0x00, boolean, 0xFF}
            {0x01 0x02, enum, 0x01 0x02}
        }
        {0x01 0x02, set, {
            {0x01 0x02, enum, 0x01 0x00}
        }
    }
    {0x01 0x02, string, 0x44 0x75 0x6D 0x6D 0x79
                        0x20 0x49 0x44 0x00}
    {0x01 0x06, integer, 0x0C}
}
```


Copyright 2018, DMTF

Decoding Example: Decode Value Info

```
{0x01 0x00, set, {
    {0x01 0x00, array, {
        {0x01 0x00, set, {
            {0x01 0x00, boolean, true}
            {0x01 0x02, enum, <value 2>}
        }
        {0x01 0x02, set, {
            {0x01 0x02, enum, <value 0>}
        }
    }
    {0x01 0x02, string, "Dummy ID"}
    {0x01 0x06, integer, 12}
}
```


Copyright 2018, DMTF

Decoding Example: Use Sequence Numbers to Restore Strings and Enum Values

```
{"DummySimple", set, {
    {"ChildArrayProperty", array, {
        {<Array Element 0>, set, {
            {"AnotherBoolean", boolean, true}
            {"LinkStatus", enum, "NoLink"}
        }
        {<Array Element 1>, set, {
            {"LinkStatus", enum, "LinkDown"}
        }
    }
    {"Id", string, "Dummy ID"}
    {"SampleIntegerProperty", integer, 12}
}
```


Copyright 2018, DMTF

Decoding Example: Write Out JSON


```
{
    "DummySimple" : {
       "ChildArrayProperty" : [
          {
             "AnotherBoolean" : true,
             "LinkStatus" : "NoLink"
          },
          {
             "LinkStatus" : "LinkDown"
          }
       ],
       "Id" : "Dummy ID",
       "SampleIntegerProperty" : 12
    }
}
```


Copyright 2018, DMTF

Next Up: Tasks

- Overview
- Discovery and Registration
  - Schemas and Redfish Resource PDRs
- Operations
- Advanced Topics: Binary Encoded JSON (BEJ)
  - Dictionary Preparation
  - Encoding Example
  - Decoding Example
- Advanced Topics: Tasks
- Advanced Topics: Eventing

Copyright 2018, DMTF

Tasks

- When an Operation cannot complete quickly, the device spawns a Task to run it asynchronously
- Running async status reflected in initial return to the MC
- MC creates Task and TaskMonitor objects in the appropriate collections
  - ResourceID names the resource the Task is operating on
  - OperationID names the specific Operation
  - Combination of these can form part of the external URI exposed to the client for Task management
  - Client read of the TaskMonitor causes the MC to issue the RDEOperationStatus command to get current info from the device
  - Client delete of the Task collection member or TaskMonitor causes MC to issue RDEOperationKill to abort the Operation
  - MC removes both Task collection member and the TaskMonitor when client successfully reads TaskMonitor

Copyright 2018, DMTF

Next Up: Eventing

- Overview
- Discovery and Registration
  - Schemas and Redfish Resource PDRs
- Operations
- Advanced Topics: Binary Encoded JSON (BEJ)
  - Dictionary Preparation
  - Encoding Example
  - Decoding Example
- Advanced Topics: Tasks
- Advanced Topics: Eventing

Copyright 2018, DMTF

Eventing

- Two forms of Events in RDE
  - Task completion Events: asynchronous completion of an long-running Task
  - Redfish Events: various occurrences, based on the message registry
  - New PLDM eventClass defined for each type
- Two models for getting Events from device to MC
  - If device, channel, and MC all support asynchronous notifications
    - Device pushes Event with PLDM M&C PlatformEventMessage command
    - MC response acknowledges Event and allows device to free Event resources
  - Otherwise (such as SMBus)
    - MC polls in a loop via PLDM M&C PollForPlatformEventMessage command
    - Each loop iteration acknowledges the previous Event

Copyright 2018, DMTF

Thank you!

Copyright 2018, DMTF