Redfish DCIM Work in Progress

DMTF Redfish Forum – DCIM Task Force

Version 0.8 – August 2018

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 Distributed Management Task Force (DMTF) website.

DCIM Work in Progress v0.8

- Focused on architectural direction and common features of the model
  - General-purpose Sensor model
  - Ability to reflect “sensor readings” elsewhere in the model
  - Handling of alarm conditions where no additional data is available
- Rendered these into one example with schema and mockup
  - Rack-based Power Distribution Unit (RackPDU)
  - Schemas and mockups
- Other DCIM equipment schemas were not updated
  - Desire for feedback on the architectural features as shown in RackPDU
  - All schemas will be refactored in the next release to match the direction

SENSOR MODEL

Sensor definition

- A Redfish Sensor is a monitoring device which produces a single reading
  - May include other related, time-coherent readings
  - Reports physical context and other “purpose” identification properties
  - Metadata describing accuracy, sampling frequency, etc.
  - Simple thresholds which indicate a change of state of the monitored item

- Base sensor definition:
  - “ReadingType” – The sensor type (temperature, voltage, etc.)
  - “Reading” – Value of the sensor (no Units in name to allow for generic software use)
  - “ReadingUnits” – Applies to all values in sensor (thresholds, reading)
    - Units shall be explicitly defined for each ReadingType to avoid mis-identification.
    - Keep to one order-of-magnitude usage per Unit of Measure if possible
    - Large-scale difference in magnitude may warrant separate ReadingTypes (e.g. mV, kV)
  - “Thresholds” – Set of defined thresholds with a common structure
    - Severity, hysteresis, etc.

Example: Voltage Sensor

```
{
   "@odata.id": "/redfish/v1/Chassis/1/Sensors/VRM1",
   “@odata.type”: “Sensor.v0_8_0.Sensor”,
   “ReadingType”: “Voltage”,
   "Name": "VRM1 Voltage",
   "SensorNumber": 11,
   "Status": {
      "Health": "OK"
   },
   "Reading": 12,
   “ReadingUnits”: “V”,
   “Thresholds: {
      "UpperCritical": {
         “Reading”: 13
      }
   },
   "MinReadingRange": 0,
   "MaxReadingRange": 20,
   "PhysicalContext": "VoltageRegulator",
   "RelatedItem": [
       { "@odata.id": "/redfish/v1/Systems/1" } ]
},
```

- DCIM models utilize a common “Sensor” schema to model devices which provide a single reading (or a set of coherent readings) as its function.
- New “ReadingType” used to differentiate between types of sensors
- “Reading” is separate from “ReadingUnits” in the Sensor definition to allow common telemetry usage (an exception to the Redfish naming convention).
- Value of “ReadingUnits” applies to all Reading-related properties, and is defined by the ReadingType with only one unit allowed per type for interoperability.
- SensorNumber is a legacy IPMI concept (used in existing Power/Thermal schemas) which we may be able to abandon here…

Sensor Collections, Filtered Arrays and Excerpts

- *Sensor* Collection is a Resource Collection which holds all sensors housed in a Chassis or Facility (perhaps other containment)
- A Filtered Array allows convenient access to a subset of a collection
  - E.g. Temperatures[] contains all temperature sensors
  - This shows up in the WIP mockup for RackPDU “TriggeredAlarms”
- Redfish Excerpt concept allows “copies” of sensor data to exist where desired for common use cases without duplication of the entire Sensor resource contents

SCHEMA EXCERPTS

Schema Excerpt Goals

- Provide compact response for sensor payloads
  - Deliver sensor reading and *enough* context for normal usage
  - End users “just need to get temperatures…”
- Ensure full data can be discovered
  - Avoid requirement of query parameters or other “hidden” protocol features
- Minimize implementation overhead
  - Share code between the sensor resource and any “compact” version
- Create concept that could apply elsewhere in Redfish, not just Sensor

Excerpt Property Concept

- Allow a copy of selected data from a Collection member
  - But avoids addition of Links/Members or other schema-induced overhead
- Include only *essential* or *dynamic* properties for common usage
  - Reduce static payload size and processing of frequently polled resources
  - Definition of Excerpt properties set in the Collection member's schema
- Provide link to the full resource instance in a Collection
  - "DataSourceUri" property (value is URI for the Member of a Collection)
- Excerpt properties are not necessarily required
  - Properties may be Required and an Excerpt: "Reading", "ReadingUnits"
  - But optional properties may also be Excerpt: "PhysicalContext"

Creating an Excerpt

- Add an object to contain the subset of properties from another resource
  - Contents will be the Excerpt properties from the target resource
- This object can be a single instance or a Filtered Array
  - “Temperature”:{excerpt} or “Temperatures”: [{excerpt}, {excerpt}, …]
- Filtered Array instances all share the same sub-type (key property)
  - Follows normal array pattern for Redfish
  - Example: “Temperatures”[ ] contains only temperature sensors, not humidity, fans, or other types of Members in the Sensors Collection
- Service returns only supported Excerpt properties
  - Properties marked in referenced schema as “Redfish.Excerpt”
  - Only those properties supported by implementation
  - If resource contains no marked properties, Service returns entire resource

Example: Voltage Sensor

```
{
   "@odata.id": "/redfish/v1/Chassis/1/Sensors/VRM1",
   “@odata.type”: “Sensor.v0_8_0.Sensor”,
   “ReadingType”: “Voltage”,
   "Name": "VRM1 Voltage",
   "SensorNumber": 11,
   "Status": {
      "Health": "OK"
   },
   "Reading": 12,
   “ReadingUnits”: “V”,
   “Thresholds: {
      "UpperCritical": {
         “Reading”: 13
      }
   },
   "MinReadingRange": 0,
   "MaxReadingRange": 20,
   "PhysicalContext": "VoltageRegulator",
   "RelatedItem": [
      { "@odata.id": "/redfish/v1/Systems/1" } ]
},
```

Highlighted data included in Excerpt

Excerpts tied to a single ReadingType

Name needed for context within arrays

Status is important data about the sensor

Reading (with Units) is the primary property

Thresholds and Ranges are static data and therefore not included in excerpts

Physical context used to identify excerpts

Example: Excerpt Copy of Voltage Sensor

```
“Voltages": [
  {
     “DataSourceUri": “<path>/Sensors/VRM1",
     "Name": “VRM1 Voltage",
     "Status": {
        "Health": "OK"
     },
     "Reading": 12,
     “ReadingUnits”: “V”,
     "PhysicalContext": “VoltageRegulator"
  },
  . . .
],
```

- Voltages[] Filtered Array would appear where useful in other (non-Sensor) resources
- DataSourceUri link points to Sensor collection
  - This property is constructed for the Excerpt, and does not appear in the referenced collection member
- Includes Excerpt properties from “Sensor”:
  - Reading
  - ReadingUnits
  - Name (from Resource)
  - Status (from Resource)
  - PhysicalContext – Need for multiple sensors (CPU vs Ambient)
- Excludes non-Excerpt properties:
  - Thresholds
  - Sensor capabilities
  - RelatedItems

Redfish Schema Annotations for Excerpt properties

- **Excerpt:** Property should appear in referenced copies
  - <Annotation Term="Redfish.Excerpt" String="Sensor.ReadingType\RPM"/>
  - Optional string value indicates a "key" property and enumeration value used to indicate that the property is only used for certain types
  - If no string value is provided, then the property is Excerpt for all uses
  - This conditional inclusion is used during conversion to other schema languages and for documentation generation, although it may be useful for code generation utilities.
- **ExcerptCopy:** Object references Excerpt properties from a resource
  - <Annotation Term="Redfish.ExcerptCopy" String="Sensor.ReadingType\RPM"/>
  - The optional string again references the key property and value, and it is this value that is used to match the conditional inclusion of Excerpt properties.
- **ExcerptCopyOnly:** Property only populated in copies, not original resource
  - <Annotation Term="Redfish.ExcerptCopyOnly"/>
  - Used for pointer property to the original resource, which would be a duplicate 'self' pointer in that original resource.

Excerpt Query Parameter

- When used, service returns only the “excerpt” properties in payload
  - Optional, but recommended, protocol feature
  - Example: `GET /redfish/v1/Chassis/Sensors/CPUTemp1?excerpt`
- Lacks “leading-$” naming to allow service to safely ignore as unknown
  - Specification requires service to reject unknown “$” parameters
  - In this case, no harm to returning entire payload
- Combines well with $expand, $filter on collections
  - `GET \redfish\v1\Chassis\1\Sensors?excerpt&$expand&`
    `$filter=“Members/ReadingType”%20eq%20“Temperature”`

Recommended usage

- Ad hoc request, simple clients: GET resources with Excerpts defined
- Discovery of sensors: GET Sensor Collection with $expand
- Sensor details or settings: GET Sensor Collection member
- Frequent polling individual sensor: GET member with “excerpt” query
  - “Get reading for temperature sensor #4”
  - But use TelemetryService features to avoid polling!
- Polling by sensor type (GUI): Sensor collection with $expand, $filter
  - Can combine with “excerpt” for better performance

ALARMS

Alarm definition

- Many types of equipment encounter conditions needing immediate attention
  - But in many cases these are difficult to represent as properties (data), as the condition may not be directly measured or observed
  - Or the condition could only be represented in “normal” / “attention required” states
  - In the past, these conditions were rendered as SNMP Traps
  - Examples: “Circuit overload”, “Moisture detected”, etc.
- Rather than create large numbers of properties that have static “normal” values except in rare cases, Redfish defines a collection of “Alarms”
  - Leverages Redfish LogEntry resource format – inherit by copy…
  - But unlike a Log, the entries are static based on Alarm definitions (not removed from collection)
    - One “entry” (collection member) per supported Alarm
    - This allows discovery of available alarms
    - AlarmStatus allows configuration, enable/disable of specific alarms and actions
- TriggeredAlarms[] (or “ActiveAlarms”?) array
  - Excerpt of Alarm collection for active alarms
  - Provides simple annunciator panel output

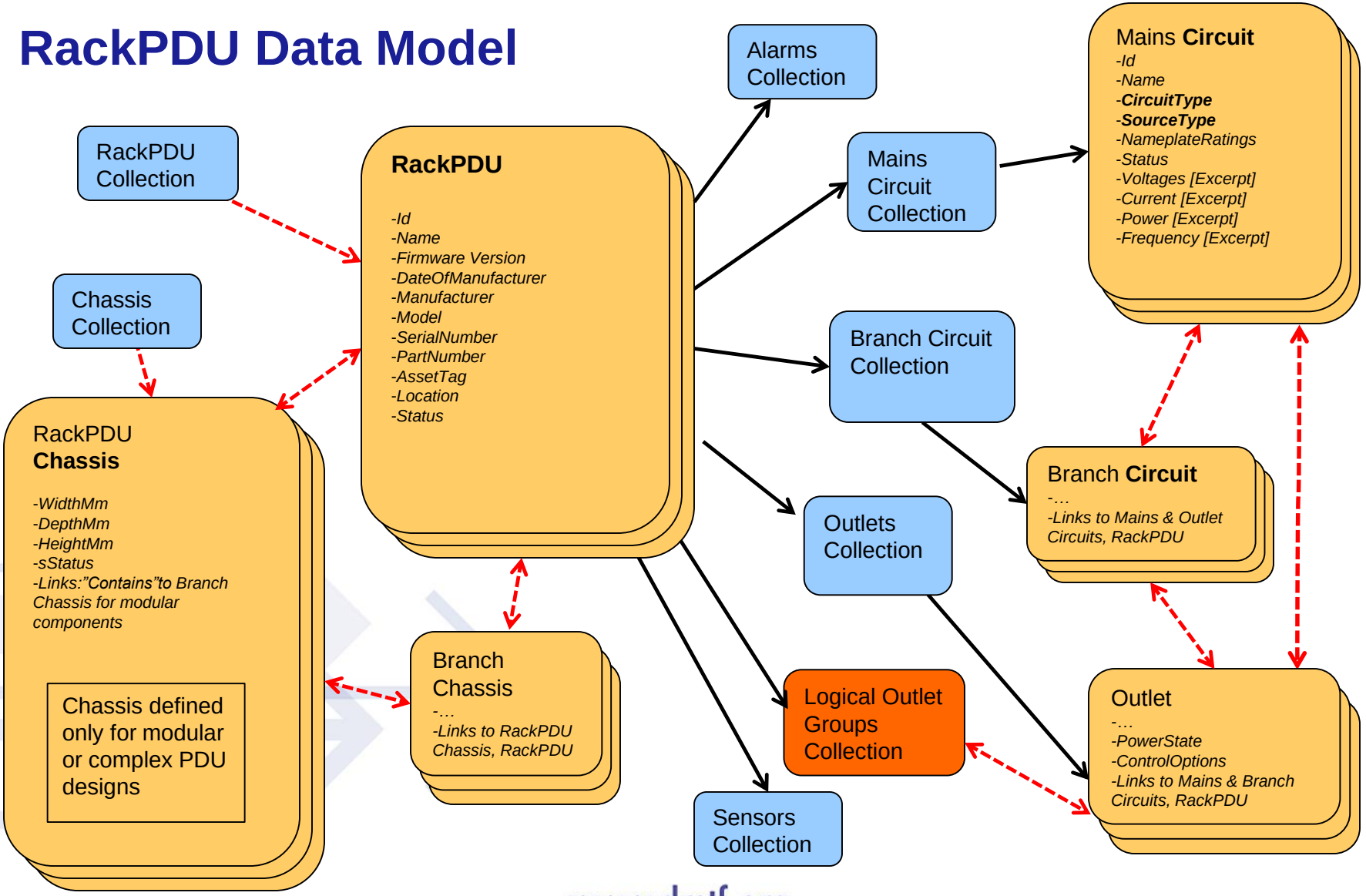
Alarm Example

```
"@odata.context": "/redfish/v1/$metadata#Alarm.Alarm",
"@odata.id": "/redfish/v1/DCIMPower/default/RackPDU/1/Alarms/Overload,
"@odata.type": "#Alarm.v0_8_0.Alarm",
"Id": "Overload",
"Name": "PDU Unit Overload",
"AlarmState": "Triggered",
“Acknowledged": false,
"Severity": "Critical",
"TriggerTime": "2018-08-07T14:44:00Z",
"AutomaticReArm": true,
"Message": "Rack PDU Overload Condition",
"MessageId": "DCIM.0.1.0.Overload",
"MessageArgs": [
    "58703"
],
"Links": {
    "RelatedSensor": {
        "@odata.id": “<Sensor URI>/ACMainPower"
    },
    "Oem": {}
},
```

- Uses Redfish Message concepts
- Allows for automatic re-arm
  - May need more options
  - Re-arm at reset, time elapsed, etc.
- AlarmState = “Disabled”, “Armed”, “Triggered”
- Use PATCH to acknowledge & clear
- Links to sensors, related items

RACK PDU MODEL

RackPDU Data Model



RackPDU additions since previous draft

- Single instance (not array) of some sensor types
  - Energy and Power readings for whole PDU
- New subordinate collections
  - Mains: Input Circuits
  - Branches: Output Circuits
  - Outlets: All outlets regardless of their branch assignment
  - Logical Outlet Groups: Defined sets for control
- Created sub-objects to hold instances of sensors
  - Max of one sensor per leg/type, so this avoids arrays
  - See: \public-infrastructure\DCIMPower\default\RackPDU\1\Mains\AC1
- “Faceplate” ratings object removed
  - Many are max values on Mains so belong in the Circuit resource
  - Simple ‘faceplate’ values may still be useful here, feedback requested

Incorporating Feedback from industry

- Many obvious additions, these have been incorporated
  - Some of these were in early (unreleased) mockups as well
- Added “Peak Value” concept to Sensor
  - Allow use on any sensor
  - Value is an excerpt property if supported, timestamps and action are not
  - Action to clear peak value and reset interval
  - Also must allow for fixed intervals defined by service
- Added ‘delay’ for power on/off
  - Action parameter allows for override of these (if supported)
- Need to ensure ‘unmanaged’, ‘monitored’, ‘controlled’ outlet support
  - PDUs can contain a mix of these
  - Probably a simple count of unmanaged outlets (don’t create Outlet collection entries for zero data…)

Circuit schema

- Single schema type to describe input and output circuits
  - Outlet schema may become a copy of Circuit with outlet-specific properties
- Ratings
  - Current is straightforward – value in Amps
  - Voltage – should this be a “nominal voltage” value (Volts)
  - … or does Voltage need to be expressed as an enum for interop?
    - See Voltage Ranges from Power Supply (copy those or new set of enums)
- AC vs DC circuits / outlets
  - One voltage sensor type for both AC and DC
    - Added a VoltageType property to describe [AC, DC, either?]
  - Would like feedback on the choice between:
    - Single “Voltage” reading type with AC/DC sub-type property
    - Two reading types: “ACVoltage” and “DCVoltage”

Sensor additions

- Adding “Energy” sensor type
  - Previously a property under Power sensor type
  - Will need to describe ’32-bit rollover’ behavior
- Adding “ResetStatistics” action for Sensor
  - Energy and other interval sensors or running counts need to be reset
- Handling “Peak” readings – from industry feedback
  - This is a common use case, so don’t want to force into a Telemetry model
  - Add “PeakReading” to Sensor, along with timestamp/interval
    - Keeps data close to values, shares all the reset of the Sensor object
    - “SensorResetTime” – time interval was last reset (better as “elapsed time”?)
    - “PeakReadingTime” – time when PeakValue occurred
    - All get reset with “ResetStatistics” action
    - Could be a fixed interval (defined by implementation) or user reset

Outlet usage in Circuit Schema

- PowerState / Breaker State
  - Use Booleans for since we don't need 'shutdown' states?
  - Or enums for one or the other?
- Actions for PowerState and Breaker Reset
  - On/Off/Reset(reboot)/Override
- Receptacle Type (three properties):
  - IEC defines the plug styles, single char "A", "B", etc.
    - These should be enums to provide the useful translation "B is North American grounded outlet" "Type1-15", "Type5-15", "Type15-15"
  - NEMA defines sub-types within IEC type, 1-18
    - Combines with amp rating to get the full NEMA name (e.g. "2-15R")
  - Locking Receptacle - completes description of the receptacle type
  - Common names for datacenter receptacles may be desired
    - Under investigation – where does "C-30" map?

NEMA outlet types

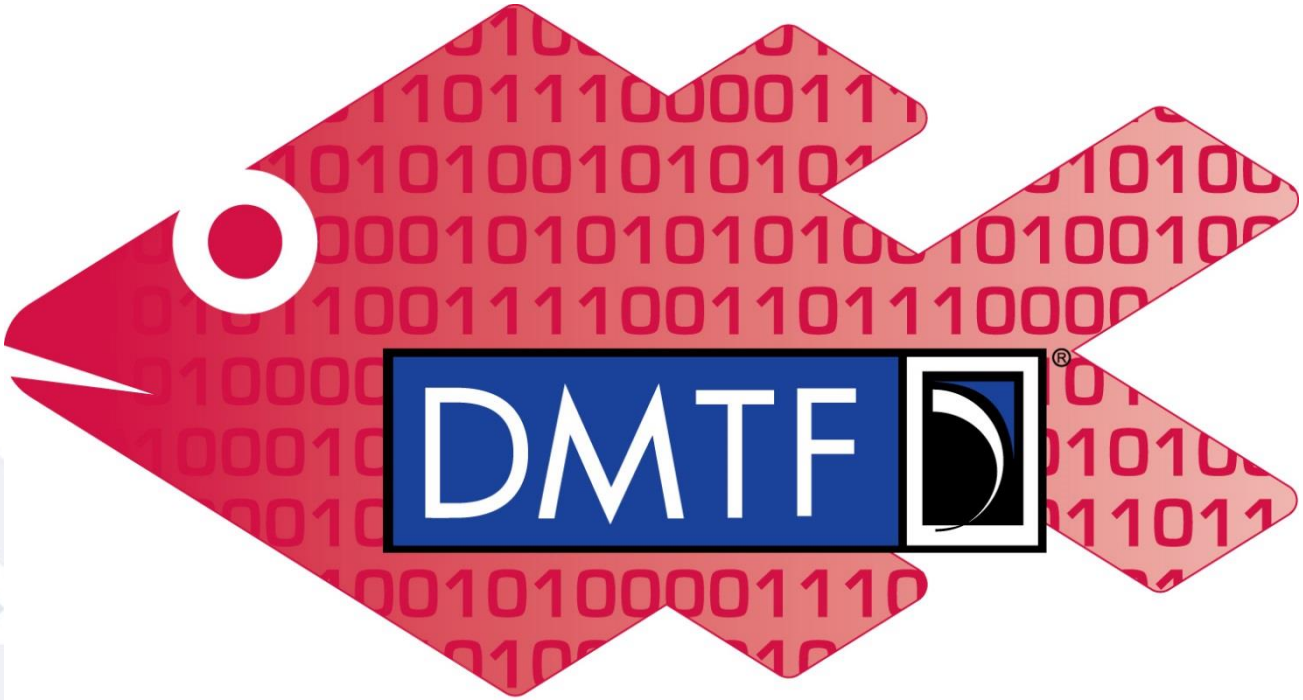
VOLTAGE	NEMA	15 AMPERE		20 AMPERE		30 AMPERE		50 AMPERE		60 AMPERE	
		Receptacle	Plug	Receptacle	Plug	Receptacle	Plug	Receptacle	Plug	Receptacle	Plug
125 V	1	1-15R	1-15P								
250V	2		2-15P	2-20R	2-20P	2-30R	2-30P				
125V	5	5-15R	5-15P	5-20R	5-20P	5-30R	5-30P	5-50R	5-50P		
250V	6	6-15R	6-15P	6-20R	6-20P	6-30R	6-30P	6-50R	6-50P		
277V, A.C.	7	7-15R	7-15P	7-20R	7-20P	7-30R	7-30P	7-50R	7-50P		
125/ 250V	10			10-20R	10-20P	10-30R	10-30P	10-50R	10-50P		
3Ø250V	11	11-15R	11-15P	11-20R	11-20P	11-30R	11-30P	11-50R	11-50P		
125/ 250V	14	14-15R	14-15P	14-20R	14-20P	14-30R	14-30P	14-50R	14-50P	14-60R	14-60P
3Ø250V	15	15-15R	15-15P	15-20R	15-20P	15-30R	15-30P	15-50R	15-50P	15-60R	15-60P
3ØY 120/208V	18	18-15R	18-15P	18-20R	18-20P	18-30R	18-30P	18-50R	18-50P	18-60R	18-60P

QUESTIONS FOR INDUSTRY

Open topics and questions

- Does the Redfish Sensor model fit your use cases?
  - How much data belongs in the Sensor vs. a “control system” schema
    - Modeling of a thermostat (and HVAC system) vs. temperature sensor
  - Would actuators be modeled appropriately as Sensors?
- What additional types of Sensors or readings are needed?
  - Additional types easily added in the future, but…
  - Would like to have a rich set for v1.0
- Allow user to override power on/off delays on Outlet Controls?
  - Normal operations (turn on, turn off power) would adhere to delays
  - Extra enumerations (“turn off w/no delay”) would bypass delays
  - Bypassing “off” delay seems reasonable, what about “on”?

Q&A & Discussion



Redfish