Power and Thermal enhancement proposal

Redfish Forum

Version 0.8 – May 2020

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.

Providing Feedback

- Redfish Forum is soliciting feedback on this proposal
- Items show in RED are open questions that need answers
  - Several proposals are shown in mockups and schema, Forum desires feedback from end users on direction for these items
- Feedback to the DMTF Redfish Forum is encouraged
  - Submit items using the DMTF feedback portal
  - https://www.dmtf.org/standards/feedback
- Questions and comments can be posted on the Redfish User Forum
  - https://www.redfishforum.com

Problems with existing Power and Thermal schemas

- Thermal and Power schemas have grown significantly in scope
  - Both were defined in “pre-1.0” Redfish with an IPMI-replacement scope
- Original design avoided use of Resource Collections
  - Items such as Fans, Power Supplies, Temperature sensors rendered as arrays
  - Now, each power supply has more than enough data to stand alone as a resource
- Numerous arrays used – these are cumbersome
  - Difficult to access and correlate data due to JSON array structure
- Large amount of static data mixed with multiple sensor readings
  - Performance issue at scale for both Power and Thermal resources
- Existing models do not leverage Sensor definition
  - Inconsistent definitions of properties (Power, Temperature in Processor / Memory)
  - Cannot retrieve sets of sensor data by type

Goals

- Retain compatibility for existing implementations
  - Allow migration but don't break compatibility
- Provide updated models for power and cooling subsystems
  - Use learnings over last 5 years of Redfish modeling
  - Incorporate Sensor model
  - Add connectivity to show additional and interoperable power / thermal relationships
- Separate "metrics" (sensor readings) from large static resources
  - Support individual sensor polling as well as efficient metric gathering
- Provide consistent support for external power sources
  - Not just "power supplies" or links to Chassis
  - Include support for battery systems (in conjunction with UPS schema definitions)
- Provide support for liquid-cooled units
  - Parallel the support provided for air-cooled (fan) units

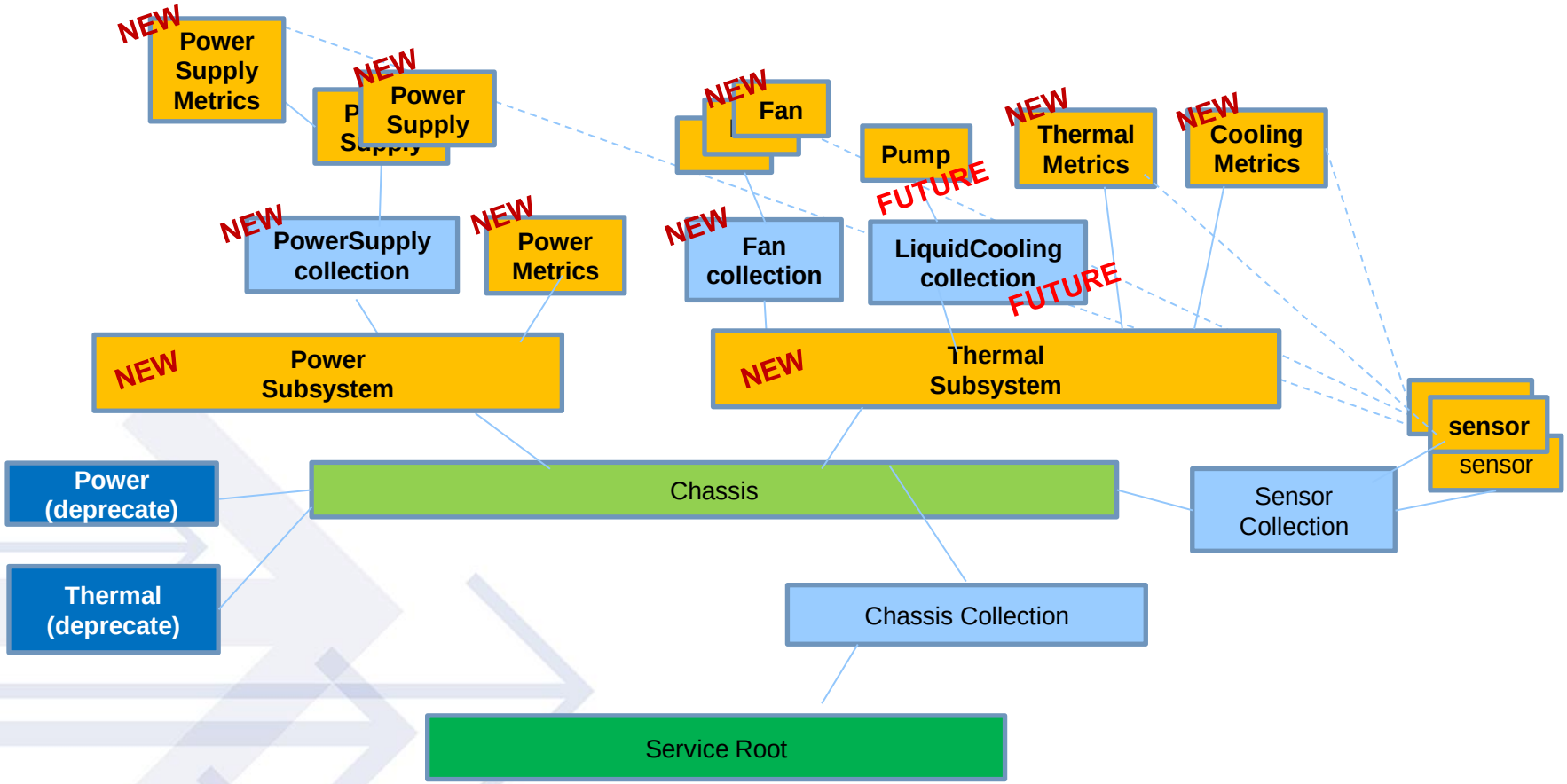
Approach

- Phase 1: Disposition of existing Power and Thermal properties
  - Decide on “new” locations, migration path, and deprecation
  - Produce a work-in-progress release of this work in 2020.2 timeframe
  - Provide time for feedback and design work on Phase 2 before finalizing
- Phase 2: Extend power / thermal model coverage
  - External power sources not covered by power supply model
    - Battery sources, shared infrastructure
  - Liquid cooling systems – both internal and external
  - Enhanced power management support
- Tools
  - Provide tools / libraries to convert between models
    - Allow client to assume “new” model, library will convert from “old”

Migration Summary

- *Fans[]* and *PowerSupplies[]* become Resource Collections
  - Contain mostly static data for these devices (and their “bays”)
  - *PowerSupply* gains a Metrics resource due to large sensor count
- *Voltages[]* and *Temperature[]* sensors become Sensor instances
  - Summarized in new PowerSubsystemMetrics and ThermalMetrics
- *PowerControl*[] object split along functional lines
  - Power limits move to the appropriate resources
- Sensor excerpts added to various “Metrics” resources where desired
  - Much easier to correlate “CPU readings” by starting at Processor instead of searching Sensor collection
- “Subsystem” resources lay groundwork for further model expansion
  - Liquid cooling, external power sources

Power and Thermal resource tree additions



POWER MIGRATION

Migration of properties from Power schema

- Allow coexistence of Power with new resources and collections
  - Deprecate use Power schema (will deprecate link in Chassis)
  - Provide indications of which portions of the resource have migrated
- New resources for power subsystem, power supplies, and metrics
  - Separate sensor data from large quantity of static data
  - Lay groundwork to populate batteries and other external power sources
- Use *RelatedItem*[] link in *PowerSupplies*[] to point to PowerSupply?
  - Alternative is to add new 'migration' link such as "PowerSupplyUri"
- Use *RelatedItem*[] link in *PowerControls*[] to point to new location?
  - Since the limits get distributed to Processor, Memory, etc.
  - Alternative is to add new 'migration' link such as "ComponentUri"
  - Requesting feedback on the usefulness of these pointers

NEW PowerSubystem and supporting schemas

- PowerSubsystem
  - The equipment and connectivity that provides power to a Chassis
    - Expect to add a “Power Source” collection
  - Redundancy group information
- PowerMetrics
  - Power consumption as a structured object with Sensor excerpts
  - Voltage regulators as a Sensor excerpt array
- PowerSupply
  - Resource Collection for individual power supply (and bay) resources
- PowerSupplyMetrics
  - Support measurements for a well-instrumented power supply

Migration of Power properties

- *PowerSupplies*[]
  - Contents move to PowerSupply schema
- *Voltages*[]
  - Voltage Regulator Modules become a set of Sensor resources
  - Other voltage measurements shown in PowerSupplyMetrics
  - Is a summary in PowerSubsystemMetrics useful? (likely an array)
- *PowerLimit*
  - Chassis power limit becomes a single property (perhaps two properties for Min / Max)
  - Individual subsystem/device limits move to those resources
    - Example: CPU limit placed in Processor schema with excerpt for power reading
  - May handle *LimitException* and *CorrectionInMs* by Sensor *Thresholds*
- Power Allocation
  - Set of properties for shared infrastructure (bladed chassis) becomes *Allocation* object
- *PowerControl*[] (object array with multiple functions…)
  - Moves to Chassis, PowerSubsystem and individual device resources
- *Redundancy*[]
  - Moves to PowerSubsystem and renaming to handle multiple redundancy topics

PowerSubsystem

- Resource holds static power subsystem data and settings
- Chassis-level power limit
- *PowerAllocation* object for shared power infrastructures (e.g. blades)
- Redundancy information
  - Individual objects by topic (not a single Redundancy[] array)
  - *PowerSupplyRedundancy*[] is the first instance
  - How to model redundant power feeds / sources?
    - Since the redundancy is “external”, perhaps that’s out of scope for this resource
    - Ability to show the power cord connections would allow aggregator to discover

PowerSubsystem mockup

```
{
    "@odata.id": "/redfish/v1/Chassis/1U/PowerSubsystem",
    "@odata.type": "#PowerSubsystem.v1_0_0.PowerSubsystem",
    "Name": "Power Subsystem for Chassis",
    "LimitWatts": 1200,
    "Allocation": {
       "RequestedWatts": 1500,
       "CapacityWatts": 2000,
       "AllocatedWatts": 1200
    },
    "PowerSupplyRedundancy": [
        {
            "@odata.id": < URI of PowerSupplyRedundancy >,
            "MemberId": "0",
            "Name": "Power Supply Redundancy Group 1",
            "Mode": "Failover",
            "MaxNumSupported": 2,
            "MinNumNeeded": 1,
            "RedundancySet": [
                { "@odata.id": < URI of Power Supply Bay #1 > },
                { "@odata.id": < URI of Power Supply Bay #2 > }
            ],
            "Status": {
                "State": "Offline",
                "Health": "OK"
            }
        }
    ],
    "PowerSupplies": {
        "@odata.id": < URI of PowerSupply resource collection >
    },
    "Metrics": {
        "@odata.id": < URI of PowerMetrics resource >
    },
    "Status": {
        "State": "Enabled",
        "Health": "OK"
    },
    "Oem": {}
}
```

PowerSubsystemMetrics mockup


```
{
    "@odata.id": "/redfish/v1/Chassis/1U/PowerSubsystem/Metrics",
    "@odata.type": "#PowerSubsystemMetrics.v1_0_0.PowerSubsystemMetrics",
    "Id": "PowerSubsystemMetrics",
    "Name": "Summary Power Metrics",
    "Status": {
        "State": "Enabled",
        "Health": "OK"
    },
    "PowerWatts": {
        "General": {
            "Reading": 374,
            "DataSourceUri": "/redfish/v1/Chassis/1U/Sensors/TotalPower"
        },
        "CPUSubsystem": {
            "Reading": 139,
            "DataSourceUri": "/redfish/v1/Chassis/1U/Sensors/CPUSubsystemPower"
        },
        "SystemBoard": {
            "Reading": 40,
            "DataSourceUri": "/redfish/v1/Chassis/1U/Sensors/SysBrdPower"
        },
        "MemorySubsystem": {
            "Reading": 42,
            "DataSourceUri": "/redfish/v1/Chassis/1U/Sensors/MemorySubsystemPower"
        }
    },
```


CONTINUED ON NEXT PAGE…

PowerMetrics mockup, continued


```
    "VoltageSummary": [{
            "Name": "CPU #1 Voltage Regulator",
            "Id": "CPU1",
            "PhysicalContext": "System Board",
            "Voltage": {
                "Reading": 3.31,
                "DataSourceUri": "/redfish/v1/Chassis/1U/Sensors/VRM1"
            }
        },
        {
            "Name": "CPU #2 Voltage Regulator",
            "Id": "CPU2",
            "PhysicalContext": "System Board",
            "Voltage": {
                "Reading": 3.31,
                "DataSourceUri": "/redfish/v1/Chassis/1U/Sensors/VRM2"
            }
        }
    ],
    "Energykwh": {
        "DataSourceUri": "/redfish/v1/Chassis/1U/Sensors/TotalEnergy",
        "Reading": 325675
    },
    "Links": {
        "Oem": {}
    },
    "Oem": {},
    "Actions": {
        "PowerMetrics.ResetMetrics": {
            "target": "/redfish/v1/Chassis/1U/PowerSubsystem/Metrics/PowerMetrics.ResetMetrics"
        }
    }
}
```

PowerSupply

- Most properties copied from existing PowerSupplies[] definition
  - **Need feedback from PMBus group to ensure full data coverage**
  - *Redundancy* object moved to PowerSubsystem
  - Measurements moved to PowerSupplyMetrics
- Outlet link to show power connectivity
- Assembly link, Action for Reset of power supply
- Is *EfficiencyPercent* expected as static value or dynamic (load-based)?
  - Replace percent with enums to report the 80 Plus “Gold”/”Platinum”/etc.?
    - “EfficiencyRating”: “Plus” |“Bronze” |“Silver” |“Gold” |“Platinum” |“Titanium”
  - Dynamic efficiency can be calculated from input/output power readings
- Handling of power supply fans
  - Failures of “simple” internal P/S fans can be handled with PS fault reporting
  - Fans that contribute to overall system cooling should be reported in the Thermal / Fan resources with *PhysicalContext* of “PowerSupply”

PowerSupply mockup


```
{
    "@odata.id": "/redfish/v1/Chassis/1U/PowerSubsystem/PowerSupplies/Bay1",
    "@odata.type": "#PowerSupply.PowerSupply",
    "Id": "Bay1",
    "Name": "Power Supply Bay 1",
    "Status": {
        "State": "Enabled",
        "Health": "Warning"
    },
    "Model": "RKS-440DC",
    "Manufacturer": "Contoso Power",
    "FirmwareVersion": "1.00",
    "SerialNumber": "3488247",
    "PartNumber": "23456-133",
    "SparePartNumber": "93284-133",
    "LocatorBeacon": "Off",
    "HotPluggable": false,
    "InputVoltageType": "AC200To240V",
    "CapacityWatts": 400,
    "EfficiencyRating": "Gold",
    "InputRanges": [{
            "InputVoltageType": "AC200To240V",
            "CapacityWatts": 400
        },
        {
            "InputVoltageType": "AC120V",
            "CapacityWatts": 350
        },
        {
            "InputVoltageType": "DC380V",
            "CapacityWatts": 400
        }
    },   CONTINUED ON NEXT SLIDE
```

PowerSupply mockup, continued

```
    "Location": {
        "PartLocation": {
            "ServiceLabel": "PSU 1",
            "LocationType": "Bay",
            "LocationOrdinalValue": 0
        }
    },
    "Links": {
        "Outlet": {
            "@odata.id": "/redfish/v1/Chassis/1U/PowerSubsystem/Outlets/A4"
        }
    },
    "Assembly": {
        "@odata.id": "/redfish/v1/Chassis/1U/PowerSubsystem/PowerSupplies/Bay1/Assembly"
    },
    "Metrics": {
        "@odata.id": "/redfish/v1/Chassis/1U/PowerSubsystem/PowerSupplies/Bay1/Metrics"
    },
    "Actions": {
        "#PowerSupply.Reset": {
            "target": "/redfish/v1/Chassis/1U/PowerSubsystem/PowerSupplies/Bay1/PowerSupply.Reset"
        }
    }
}
```

PowerSupplyMetrics

- Use objects for input voltage ranges from PowerDistribution?
  - Existing property is an array of supported ranges with min/max values
  - Replace those with objects for AC (2 ranges) and DC ranges
    - "AC", "AC2", "DC" (rationale that most supplies have a single AC range)
  - Could Voltage/Frequency ranges be replaced with enumeration?
    - Use values of LineInputVoltageType and add range in normative description?
- Adds separate input and output measurements for Voltage and Current
  - Output metrics structures
    - Follows repeating object pattern for simple, interoperable software access
    - "ThreeVolt", "FiveVolt", "TwelveVolt", "FortyEightVolt", "Neg48Volt"?
  - Is *OutputPower* needed? Single reading or structured object?
- Power, Energy, Frequency measurements
- Show Power Supply temperature, fan speed here or in separate EnvironmentMetrics resource?

PowerSupplyMetrics mockup (1 of 3)


```
{
    "@odata.id": "/redfish/v1/Chassis/1U//PowerSubsystem/PowerSupplies/Bay1/Metrics",
    "@odata.type": "#PowerSupplyMetrics.PowerSupplyMetrics",
    "Id": "Metrics",
    "Name": "Metrics for Power Supply 1",
    "Status": {
        "State": "Enabled",
        "Health": "Warning"
    },
    "NominalInputVoltage": "AC200To240V",
    "InputVoltage": {
        "DataSourceUri": "/redfish/v1/Chassis/1U/Sensors/PS1InputVoltage",
        "Reading": 230.2
    },
    "InputCurrentAmps": {
        "DataSourceUri": "/redfish/v1/Chassis/1U/Sensors/PS1InputCurrent",
        "Reading": 5.19
    },
    "InputPowerWatts": {
        "DataSourceUri": "/redfish/v1/Chassis/1U/Sensors/PS1InputPower",
        "Reading": 937.4,
        "ApparentVA": 937.4,
        "ReactiveVAR": 0.0,
        "PowerFactor": 0.98
    },
    "InputFrequencyHz": {
        "DataSourceUri": "/redfish/v1/Chassis/1U/Sensors/PS1InputFrequency",
        "Reading": 60.0
    },
```


CONTINUED ON NEXT SLIDE

PowerSupplyMetrics mockup (2 of 3)

```
"OutputVoltages": {
        "FiveVolt": {
            "DataSourceUri": "/redfish/v1/Chassis/1U/Sensors/PS1_5VOutput",
            "Reading": 5.03
        },
        "ThreeVolt": {
            "DataSourceUri": "/redfish/v1/Chassis/1U/Sensors/PS1_3VOutput",
            "Reading": 3.31
        },
        "TwelveVolt": {
            "DataSourceUri": "/redfish/v1/Chassis/1U/Sensors/PS1_12VOutput",
            "Reading": 12.06
        }
    },
    "OutputCurrentAmps": {
        "FiveVolt": {
            "DataSourceUri": "/redfish/v1/Chassis/1U/Sensors/PS1_5VCurrent",
            "Reading": 1.25
        },
        "ThreeVolt": {
            "DataSourceUri": "/redfish/v1/Chassis/1U/Sensors/PS1_3VCurrent",
            "Reading": 9.84
        },
        "TwelveVolt": {
            "DataSourceUri": "/redfish/v1/Chassis/1U/Sensors/PS1_12Current",
            "Reading": 2.58
        }
    },

CONTINUED ON NEXT SLIDE
```

PowerSupplyMetrics mockup (3 of 3)

```
"OutputPowerWatts": {
        "DataSourceUri": "/redfish/v1/Chassis/1U/Sensors/PS1OutputPower",
        "Reading": 937.4,
        "ApparentVA": 937.4,
        "ReactiveVAR": 0.0,
        "PowerFactor": 0.98
    },
    "EnergykWh": {
        "DataSourceUri": "/redfish/v1/Chassis/1U/Sensors/PS1Energy",
        "Reading": 325675
    },
    "TemperatureCelsius": {
        "DataSourceUri": "/redfish/v1/Chassis/1U/Sensors/PS1Temp",
        "Reading": 43.9
    },
    "FanSpeedPercent": {
        "DataSourceUri": "/redfish/v1/Chassis/1U/Sensors/PS1Fan",
        "Reading": 68,
        "SpeedRPM": 3290
    },
    "Actions": {
        "#PowerSupplyMetrics.ResetMetrics": {
            "target": "/redfish/v1/Chassis/1U/PowerSubsystem/PowerSupplies/Bay1/Metrics/PowerSupplyMetrics.ResetMetrics"
        }
    }
}
```

THERMAL MIGRATION

Migration of Thermal resource

- Allow coexistence of Thermal with new resources and collections
  - Deprecate *Thermal* link in Chassis after migration
- Add *SensorUri* pointer to *Temperatures*[] and *Fans*[]?
  - Follow decision on Power Supplies (is this useful?)
- Separate resources for temperature and cooling metrics
  - Large quantity of data if bundled together
  - Cooling data needs to comprehend both air (fan) and liquid cooling

Migration of Thermal properties

- *Temperatures*[]
  - Array moves to *ThermalMetrics*
  - *Sensor* instance for each temperature reading/sensor
    - All properties covered by *Sensor* instances
  - Add schema excerpts in model where physical context exists
    - Processor, Drive, Chassis, Memory
- *Fans*[]
  - Array moves to *FanCollection*
  - Summary of fan data becomes *CoolingMetrics*
  - *Fan* resource
    - Product identification (part #, serial #, etc.)
    - Fan speed, status, etc.
    - *Redundancy* moves to ThermalSubsystem

NEW ThermalSubystem and supporting schemas

- ThermalSubsystem
  - The equipment and connectivity that provides cooling for a Chassis
  - Redundancy group information
- CoolingMetrics
  - Fan metrics, summarized in a single array of metrics
  - Future expectation for a summary of liquid cooling metrics
    - Complexity of liquid cooling may warrant a separate resource for metrics on an individual unit basis, following the pattern of PowerSupplyMetrics
- ThermalMetrics
  - Temperature measurements
  - Humidity measurements
- Fan
  - Resource Collection for individual fan (and bay) resources
- Future support for liquid cooling subsystems

Reading for Fan speed

- Need to unify a “primary” fan *ReadingUnit* type
  - Easiest for end users to comprehend is a utilization percentage
    - RPM values are not comparable / meaningful across products or vendors
    - Percent value can be reported regardless of fan implementation
    - Simple fans without a reading could report a static value (suggest 100%)
  - But the RPM / tach values are interesting to many users
    - Engineers use RPM values and fan models to calculate airflow
- Add *SpeedRPM* as a new property in Sensor
  - Alongside *Reading*, follows pattern for power sensors and excerpts when additional data is provided alongside a primary reading value
- Desire to report the PWM setting (controller output) for a fan
  - Desired state vs. result
  - Add *InputPWM* property to Sensor (or perhaps Fan)
  - Comparison of PWM to output speed has been used for failure prediction

ThermalSubsystem mockup


```
{
    "@odata.id": "/redfish/v1/Chassis/1U/ThermalSubsystem",
    "@odata.type": "#ThermalSubsystem.v1_0_0.ThermalSubsystem",
    "Name": "Thermal Subsystem for Chassis",
    "FanRedundancy": [{
            "@odata.id": "/redfish/v1/Chassis/1U/ThermalSubsystem/#FanRedundancy/0",
            "Name": "Fan Group 1",
            "MemberId": "0",
            "RedundancyEnabled": true,
            "Mode": "N+1",
            "MaxNumSupported": 2,
            "MinNumNeeded": 1,
            "RedundancySet": [ { "@odata.id": "/redfish/v1/Chassis/1U/ThermalSubsystem/Fans/Bay1“ },
                { "@odata.id": "/redfish/v1/Chassis/1U/ThermalSubsystem/Fans/Bay2“ } ],
            "Status": {
                "State": "Enabled",
                "Health": "OK"
            }
    ],
    "Fans": {
        "@odata.id": "/redfish/v1/Chassis/1U/ThermalSubsystem/Fans"
    },
    "ThermalMetrics": {
        "@odata.id": "/redfish/v1/Chassis/1U/ThermalSubsystem/ThermalMetrics"
    },
    "CoolingMetrics": {
        "@odata.id": "/redfish/v1/Chassis/1U/ThermalSubsystem/CoolingMetrics"
    },
    "Status": {
        "State": "Enabled",
        "Health": "OK"
    }
}
```

ThermalMetrics mockup

```
{
    "@odata.id": "/redfish/v1/Chassis/1U/ThermalMetrics",
    "@odata.type": "#ThermalMetrics.v1_0_0.ThermalMetrics",
    "Id": "ThermalMetrics",
    "Name": "Chassis Thermal Metrics",
    "Status": {
        "State": "Enabled",
        "Health": "OK"
    },
    "TemperaturesCelsius": {
        "Internal": {
           "Reading": 39,
           "DataSourceUri": "/redfish/v1/Chassis/1U/Sensors/CPU1Temp"
        },
        "Intake": {
           "Reading": 23,
           "DataSourceUri": "/redfish/v1/Chassis/1U/Sensors/IntakeTemp"
        },
        "CPUSubsystem": {
           "Reading": 39,
           "DataSourceUri": "/redfish/v1/Chassis/1U/Sensors/CPUTemps"
        },
        "SystemBoard": {
           "Reading": 40,
           "DataSourceUri": "/redfish/v1/Chassis/1U/Sensors/SysBrdTemp"
        }
        "Exhaust": {
           "Reading": 44,
           "DataSourceUri": "/redfish/v1/Chassis/1U/Sensors/ExhaustTemp"
        }
    }
}
```

CoolingMetrics mockup


```
{
    "@odata.id": "/redfish/v1/Chassis/1U/CoolingMetrics",
    "@odata.type": "#CoolingMetrics.v1_0_0.CoolingMetrics",
    "Name": "Chassis Fan and Liquid Cooling Metrics",
    "Status": {
        "State": "Enabled",
        "Health": "OK"
    },
    "FanSummary": [{
            "Id": "Bay1",
            "PhysicalContext": "System Board",
            "SpeedPercent": {
                "Reading": 45,
                "SpeedRPM": 1900,
                "InputPWM": 55,
                "DataSourceUri": "/redfish/v1/Chassis/1U/Sensors/FanBay1"
            }
        },
        {
            "Id": "Bay2",
            "PhysicalContext": "System Board",
            "SpeedPercent": {
                "Reading": 55,
                "SpeedRPM": 2100,
                "InputPWM": 50,
                "DataSourceUri": "/redfish/v1/Chassis/1U/Sensors/FanBay2"
            }
        }
    ],
    "Oem": {}
}
```

Fan mockup


```
{
    "@odata.id": "/redfish/v1/Chassis/1U/ThermalSubsystem/Fans/Bay1",
    "@odata.type": "#Fan.Fan",
    "Id": "Bay1",
    "Name": "Fan Bay 1",
    "Status": {
        "State": "Enabled",
        "Health": "OK"
    },
    "PhysicalContext": "Chassis",
    "Model": "RKS-440DC",
    "Manufacturer": "Contoso Fans",
    "PartNumber": "23456-133",
    "SparePartNumber": "93284-133",
    "LocatorBeacon": "Off",
    "HotPluggable": true,
    "FanSpeedPercent": {
        "Reading": 45,
        "SpeedRPM": 2200,
        "PWMInput": 55,
        "DataSourceUri": "/redfish/v1/Chassis/1U/Sensors/FanBay1"
    },
    "Location": {
        "PartLocation": {
            "ServiceLabel": "Chassis Fan Bay 1",
            "LocationType": "Bay",
            "LocationOrdinalValue": 0
        }
    }
}
```

NEW PROPERTY TOPICS

New *UserThresholds* in Sensor

- Add *UserThresholds* parallel structured object in Sensor
  - Follows *Threshold* structure definition with additional properties
  - Allows user to set multiple thresholds with a *Reaction*
    - Implementation owns the reactions in the existing *Threshold* entries
  - Clearly defines 'owner' while providing user with flexibility
    - But implementation defines what support is enabled for each sensor
- *Reaction* values
  - Provide user the ability to define what action (reaction) is taken when threshold is violated
  - Follow trigger criteria defined by equivalent *Threshold*
  - Allow this reaction to be enabled/disabled individually
- Service may need to define supported range for *UserThresholds*
- Some existing *Thresholds* may move to *UserThresholds*
  - If user can define reaction behavior (but perhaps not change value)?

SENSOR INTEGRATION

Integration using Schema Excerpts

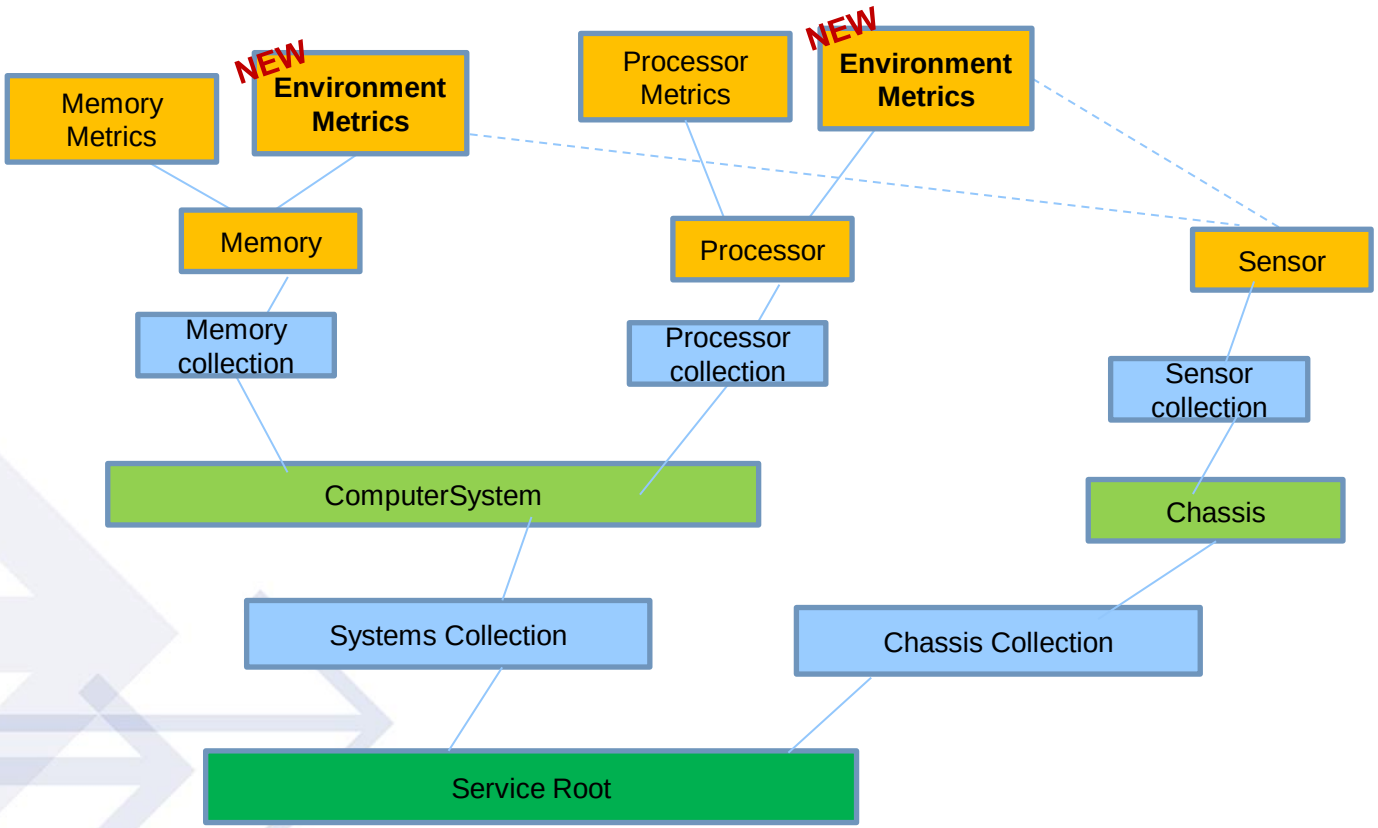
- Reference to a single Sensor is simple and low impact

```
"TemperatureCelsius": {
    "Reading": 27.3,
    "DataSourceUri": "/redfish/v1/Chassis/1/Sensors/Drive3Temp"
    }
```

- Set of multiple sensors (single type) is also possible
  - ElectricalContext model gave us a better answer than cumbersome arrays
    - Create an object structure using context to name excerpt objects
  - Easy when there can be max of one instance of a particular context

```
"TemperaturesCelsius": {
    "Intake": {
       "Reading": 27.3,
       "DataSourceUri": "/redfish/v1/Chassis/1/Sensors/BezelTemp"
    },
    "Chassis": <Excerpt>,
    "Exhaust": <Excerpt>
    }
```

Proposed resource tree additions



NEW EnvironmentMetrics resource

- A summary of sensor data related to a specific component
  - Environment readings are shown as Sensor excerpts
    - Performance metrics read directly from component, not a Sensor resource
  - Separates “performance” metrics from the environmental metrics
    - Different use cases, with “other half” of data thrown away
- Includes Reading / Sensor excerpts for:
  - Power
  - Temperature
  - Fan
  - Voltage?
- Single schema definition used for instrumented components
  - Processor
  - Memory
  - Drive

EnvironmentMetrics mockup


```
{
    "@odata.id": "/redfish/v1/Systems/1/Processors/FPGA1/EnvironmentMetrics",
    "@odata.type": "#EnvironmentMetrics.v1_0_0.EnvironmentMetrics",
    "Name": "Processor Environment Metrics",
    "Status": {
        "Health": "Ok"
    },
    "TemperatureCelsius": {
        "DataSourceUri": "/redfish/v1/Chassis/1U/Sensors/CPU1Temp",
        "Reading": 44
    },
    "PowerWatts": {
        "DataSourceUri": "/redfish/v1/Chassis/1U/Sensors/CPU1Power",
        "Reading": 12.87
    },
    "FanPercent": {
        "DataSourceUri": "/redfish/v1/Chassis/1U/Sensors/CPU1Fan",
        "Reading": 80
    },
    "Oem": {}
}
```

Processor sensor integration

- Add EnvironmentMetrics resource under Processor
- Processor Temperature
  - *TemperatureCelsius* already defined in ProcessorMetrics
    - Deprecate this in favor of new EnvironmentMetrics property
  - *ThrottlingCelsius* should migrate to a Threshold value in Sensor
    - May be multiple values? Caution and Critical?
- Processor Power
  - *ConsumedPowerWatt* already defined in ProcessorMetrics
    - Deprecate this in favor of new EnvironmentMetrics property

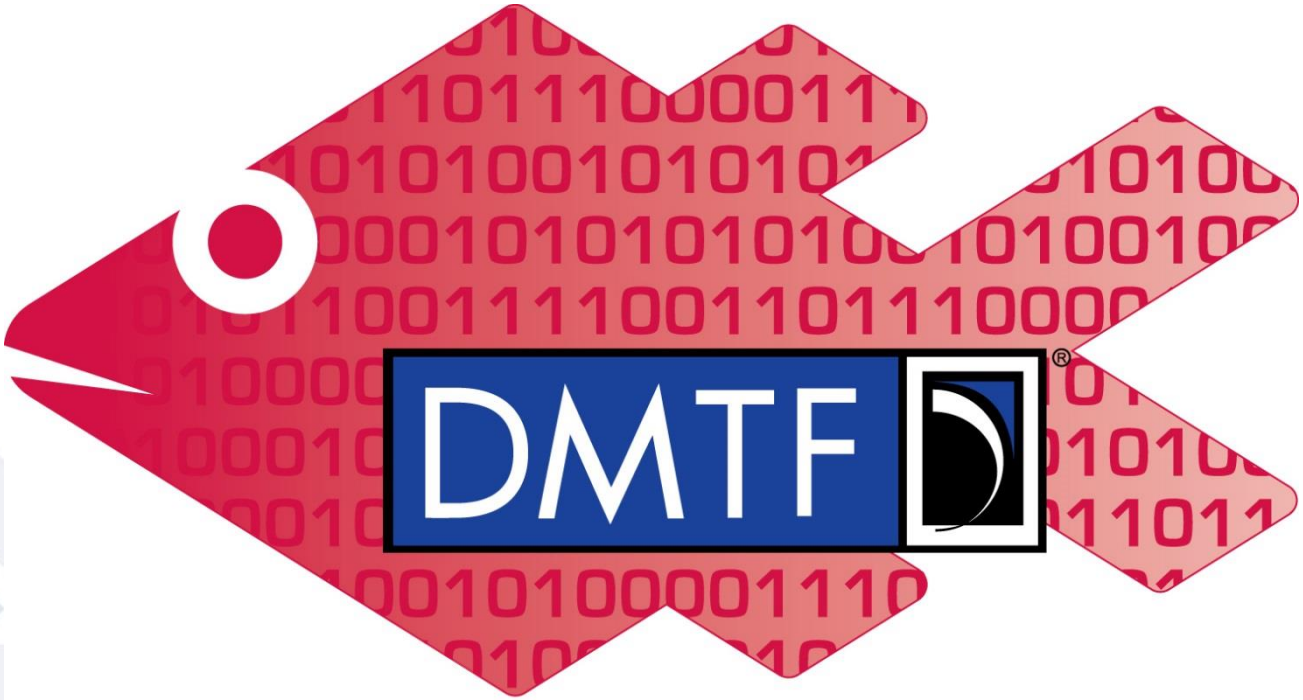
Memory sensor integration

- Add EnvironmentMetrics resource under Memory
  - *TemperatureCelsius*
  - *PowerWatts*
- Add EnvironmentMetrics resource under *MemorySummary?*
  - Temperature would be “average” or “highest” memory value
    - An average value would point to a synthesized sensor
    - Highest value would point to the specific memory device sensor being reported
    - This could be covered with a “MemorySubsystem” reading in ThermalMetrics
  - Power would be total power for memory subsystem
    - Use the Synthesized sensor concept to provide total if desired
    - This can be easily covered with a “MemorySubsystem” reading in PowerMetrics
  - This is probably better handled by “Memory” instances in the Chassis-level PowerSubsystemMetrics and ThermalMetrics resources
    - Temperature and Power readings for “MemorySubsystem”

How far to go with “Summary” data using excerpts?

- Single sensor references are easy and an obvious use case
  - Temperature of a specific device, etc.
- Object with multiple, named-for-context properties allows for easy access
  - How far do we go to provide “single” context values?
  - See Chassis temperatures as an example
- Arrays are more complex to parse, but highly flexible
  - Likely want this structure to allow quick retrieval of all sensors per type
- Follow pattern from Circuit with “simple” and “complex” types?
  - Single-phase voltage & current values are shown twice
    - One without context for simple, deterministic client access
    - One within electrical context
  - Do we create structured object for temperatures (and others?) that defines a “few” summary values, but show everything in the array version?
  - Define a base context for temperature sensor (always present)?
    - Suggest “Internal”, other choices include: “Overview”, “Summary”, “General”, others?

Q&A & Discussion



Redfish