Power and Thermal enhancement proposal

Redfish Forum

Version 0.9 – October 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 limits of JSON arrays
- 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

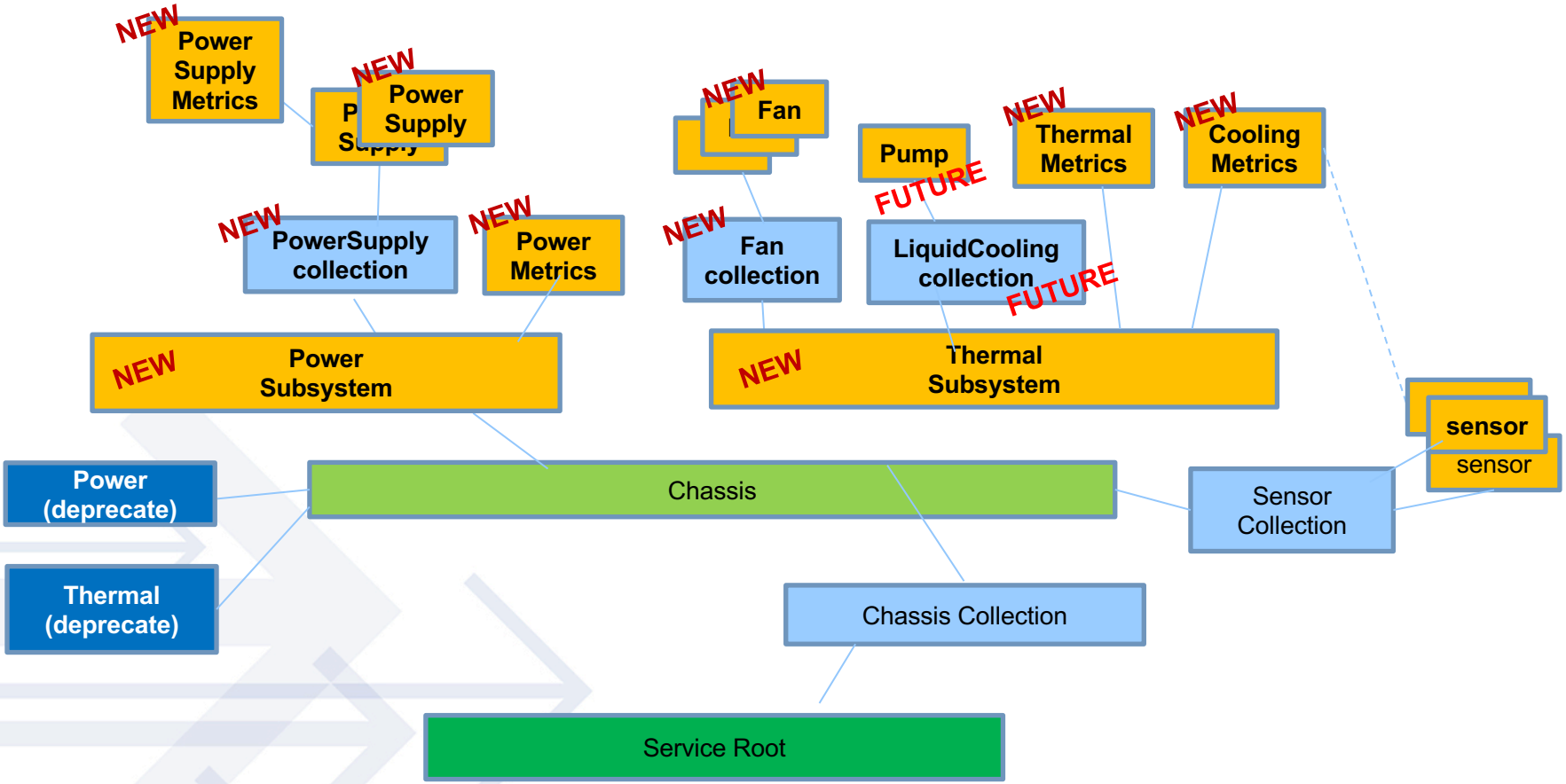
Approach

- Phase 1: Disposition of existing Power and Thermal properties
  - Decide on “new” locations, migration path, and deprecation
  - Work-in-progress v0.8 released in 2Q20
  - Incorporated feedback for v0.9 work-in-progress release
- 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
- Educational material
  - Provide open source tools / libraries to convert between models
    - Allow client to assume “new” model, library will convert from “old”
    - Seeking feedback on what tools or functions would be helpful
  - White paper to explain migration and client usage

Migration Summary

- Power and Thermal resources are completely replaced
- *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
  - Temperature readings now summarized in ThermalMetrics
  - Voltage readings move to PowerSupplyMetrics or Sensor collection
- *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

NEW PowerSubystem and supporting schemas

- PowerSubsystem
  - The equipment and connectivity that provides power to a Chassis
    - Expect to add a “Power Source” collection in the future
  - Redundancy group information
    - Streamlined definition from existing *Redundancy* object
  - Lays groundwork to populate batteries and other external power sources
- PowerMetrics
  - Power consumption at the chassis level
  - Option for subsystem-level reporting if desired (may be future addition)
- PowerSupply
  - Resource Collection for individual power supply (and bay) resources
- PowerSupplyMetrics
  - Support measurements for a well-instrumented power supply

Migration of properties from Power schema

- *PowerSupplies*[]
  - Contents move to PowerSupply schema
- *Voltages*[]
  - Voltage Regulator Modules become a set of Sensor resources
  - Other voltage measurements shown in PowerSupplyMetrics
- *PowerLimit*
  - Chassis power limit becomes a single object
  - v0.9 WIP defines this object using the proposed Control schema
    - Would handle *LimitException* and *CorrectionInMs* by Sensor *Thresholds*
    - Continuing to refine this in parallel with the proposed Control schema
  - Individual subsystem/device limits move to those resources
    - Example: CPU limit placed in Processor schema with excerpt for power reading
- 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 simplifying, but will handle multiple redundancy groups

NEW PowerSubsystem schema

- This resource contains static power subsystem data and settings
- Chassis-level power limit
  - Allows enable/disable of power limit
  - Reporting of both the limit “set point” and the power sensor “reading”
- *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",
    "CapacityWatts": 2000,
    "PowerLimitControlWatts": {
        "OperatingMode": "Automatic",
        "SetPoint": 450,
        "Reading": 284
    },
    "Allocation": {
        "RequestedWatts": 1500,
        "AllocatedWatts": 1200
    },
    "PowerSupplyRedundancy": [{
        "RedundancyType": "Failover",
        "MaxSuppliesSupported": 2,
        "MinSuppliesNeeded": 1,
        "RedundancySet": [{
              "@odata.id": "/redfish/v1/Chassis/1U/PowerSubsystem/PowerSupplies/Bay1"
            },
            {
              "@odata.id": "/redfish/v1/Chassis/1U/PowerSubsystem/PowerSupplies/Bay2"
            }
        ],
        "Status": {
            "State": "UnavailableOffline",
            "Health": "OK"
        }
    }],
    "PowerSupplies": {
        "@odata.id": < URI of PowerSupply resource collection >
    },
    "Metrics": {
        "@odata.id": < URI of PowerSubsystemMetrics 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": "PowerMetrics",
    "Name": "Summary Power Metrics",
    "TotalPowerWatts": {
        "Reading": 374,
        "DataSourceUri": "/redfish/v1/Chassis/1U/Sensors/TotalPower"
    },
    "EnergykWh": {
        "DataSourceUri": "/redfish/v1/Chassis/1U/Sensors/TotalEnergy",
        "Reading": 325675
    },
    "Oem": {},
    "Actions": {
        "#PowerSubsystemMetrics.ResetMetrics": {
            "target": "/redfish/v1/Chassis/1U/PowerSubsystem/Metrics/PowerSubsystemMetrics.ResetMetrics"
        }
    }
}
```

NEW PowerSupply schema and collection

- Most properties copied from existing *PowerSupplies*[] definition
  - *Redundancy* object moved to PowerSubsystem
  - All measurements moved to PowerSupplyMetrics
- Outlet link to show power connectivity
  - Link to *Outlet* in a *PowerDistributionUnit* instance
- Assembly link
- Action for Reset of power supply
- 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 (1 of 3)

```
{
    "@odata.id": "/redfish/v1/Chassis/1U/PowerSubsystem/PowerSupplies/Bay1",
    "@odata.type": "#PowerSupply.v1_0_0.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",
    "LocationIndicatorActive": false,
    "HotPluggable": false,
    "PowerCapacityWatts": 400,
    "PhaseWiringType": "OnePhase3Wire",
    "PlugType": "IEC_60320_C14",
    "InputRanges": [{
            "NominalVoltageType": "AC200To240V",
            "CapacityWatts": 400
        }, {
            "NominalVoltageType": "AC120V",
            "CapacityWatts": 350
        }, {
            "NominalVoltageType": "DC380V",
            "CapacityWatts": 400
        }
    ],

CONTINUED ON NEXT SLIDE
```

PowerSupply mockup (2 of 3)

```
    "EfficiencyRatings": [{
            "LoadPercent": 25,
            "EfficiencyPercent": 75
        },
        {
            "LoadPercent": 50,
            "EfficiencyPercent": 85
        },
        {
            "LoadPercent": 90,
            "EfficiencyPercent": 80
        }
    ],
    "OutputRails": [{
            "NominalVoltage": 3.3,
            "PhysicalContext": "SystemBoard"
        },
        {
            "NominalVoltage": 5,
            "PhysicalContext": "SystemBoard"
        },
        {
            "NominalVoltage": 12,
            "PhysicalContext": "StorageDevice"
        }
    ],

CONTINUED ON NEXT PAGE
```

PowerSupply mockup (3 of 3)

```
    "Location": {
        "PartLocation": {
            "ServiceLabel": "PSU 1",
            "LocationType": "Bay",
            "LocationOrdinalValue": 0
        }
    },
    "Links": {
        "Outlet": {
            "@odata.id": "https://redfishpdu.contoso.com/redfish/v1/PowerEquipment/RackPDUs/1/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"
        }
    }
}
```

NEW *PowerSupplyMetrics*

- Sensor excerpts provide all measurements
  - Sensor details can be obtained by following *DataSourceUri* links
  - All sensors are contained in a Chassis-level *Sensor* collection
- Input and output measurements for Voltage, Current, Power
  - Total power at input and output
  - Separate measurements for individual output rails
- Frequency, Energy measurements
- Temperature and Fan Speed

PowerSupplyMetrics mockup (1 of 3)


```
{
    "@odata.id": "/redfish/v1/Chassis/1U/PowerSubsystem/PowerSupplies/Bay1/Metrics",
    "@odata.type": "#PowerSupplyMetrics.v1_0_0.PowerSupplyMetrics",
    "Id": "Metrics",
    "Name": "Metrics for Power Supply 1",
    "Status": {
        "State": "Enabled",
        "Health": "Warning"
    },
    "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
    },
    "RailVoltage": [{
            "DataSourceUri": "/redfish/v1/Chassis/1U/Sensors/PS1_3VOutput",
            "Reading": 3.31
        }, {
            "DataSourceUri": "/redfish/v1/Chassis/1U/Sensors/PS1_5VOutput",
            "Reading": 5.03
        }, {
            "DataSourceUri": "/redfish/v1/Chassis/1U/Sensors/PS1_12VOutput",
            "Reading": 12.06
        }
    ],
CONTINUED ON NEXT SLIDE
```

PowerSupplyMetrics mockup (2 of 3)

```
    "RailCurrentAmps": [{
            "DataSourceUri": "/redfish/v1/Chassis/1U/Sensors/PS1_3VCurrent",
            "Reading": 9.84
        },
        {
            "DataSourceUri": "/redfish/v1/Chassis/1U/Sensors/PS1_5VCurrent",
            "Reading": 1.25
        },
        {
            "DataSourceUri": "/redfish/v1/Chassis/1U/Sensors/PS1_12Current",
            "Reading": 2.58
        }
    ],
    "OutputPowerWatts": {
        "DataSourceUri": "/redfish/v1/Chassis/1U/Sensors/PS1OutputPower",
        "Reading": 937.4
    },
    "RailPowerWatts": [{
            "DataSourceUri": "/redfish/v1/Chassis/1U/Sensors/PS1_3VPower",
            "Reading": 79.84
        },
        {
            "DataSourceUri": "/redfish/v1/Chassis/1U/Sensors/PS1_5VPower",
            "Reading": 26.25
        },
        {
            "DataSourceUri": "/redfish/v1/Chassis/1U/Sensors/PS1_12VPower",
            "Reading": 91.58
        }
    ],
```

CONTINUED ON NEXT SLIDE

PowerSupplyMetrics mockup (3 of 3)

```
  "EnergykWh": {
       "DataSourceUri": "/redfish/v1/Chassis/1U/Sensors/PS1Energy",
       "Reading": 325675
    },
    "FrequencyHz": {
       "DataSourceUri": "/redfish/v1/Chassis/1U/Sensors/PS1InputFrequency",
       "Reading": 60
    },
    "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 properties from Thermal schema

- *Temperatures*[]
  - Array content 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 content 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
  - Future support for liquid cooling
- 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

Reading for Fan speed

- Unify a primary fan *ReadingUnit*
  - Most usable for end users 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
- Added *SpeedRPM* as a new property in Sensor
  - Follows pattern established for power sensor excerpts when additional data is provided alongside a primary *Reading* value
- Desire to report the PWM setting (controller output) for a fan
  - Comparison of PWM to output speed has been used for failure prediction
  - Add as a Control to Fan (Control schema is a Work-in-Progress)
    - Control SetPoint in percent units (PWM devices typically use a 1-byte value)

Fan resource with Sensor and Control excerpts

- Contains a Sensor excerpt, and may expose a Control as well

```
{
    “SpeedControlPWM": {
        “SetPoint": 125,
        "DataSourceUri": "/redfish/v1/Chassis/1U/Controls/FanBay1"
    },
    “SpeedPercent”: {
        “Reading”: 55,
        “SpeedRPM”: 2300,
        “DataSourceUri”: “/redfish/v1/Chassis/1U/Sensors/FanBay1”
    }
}
```

Control excerpt to show SetPoint, if supported

Sensor excerpt to show Reading and “extra fan excerpt property” SpeedRPM

Fan instances that expose the control include the *SpeedControlPWM* object (excerpt of Control), while fans without the control exposed include only the *SpeedPercent object (excerpt of Sensor)*

ThermalSubsystem mockup

```
{
    "@odata.id": "/redfish/v1/Chassis/1U/ThermalSubsystem",
    "@odata.type": "#ThermalSubsystem.v1_0_0.ThermalSubsystem",
    "Name": "Thermal Subsystem for Chassis",
    "FanRedundancy": [
        {
            "RedundancyType": "NPlusM",
            "MaxFansSupported": 2,
            "MinFansNeeded": 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"
    },
    "Oem": {}
}
```

ThermalMetrics mockup

```
{
    "@odata.id": "/redfish/v1/Chassis/1U/ThermalSubsystem/ThermalMetrics",
    "@odata.type": "#ThermalMetrics.v1_0_0.ThermalMetrics",
    "Id": "ThermalMetrics",
    "Name": "Chassis Thermal Metrics",
    "TemperatureCelsius": {
        "Internal": {
            "Reading": 39,
            "DataSourceUri": "/redfish/v1/Chassis/1U/Sensors/CPU1Temp"
        },
        "Intake": {
            "Reading": 24.8,
            "DataSourceUri": "/redfish/v1/Chassis/1U/Sensors/IntakeTemp"
        },
        "Ambient": {
            "Reading": 22.5,
            "DataSourceUri": "/redfish/v1/Chassis/1U/Sensors/AmbientTemp"
        },
        "Exhaust": {
            "Reading": 40.5,
            "DataSourceUri": "/redfish/v1/Chassis/1U/Sensors/ExhaustTemp"
        }
    },
```

CONTINUED ON NEXT PAGE

ThermalMetrics mockup, continued


```
    "TemperatureSummaryCelsius": [
        {
            "Reading": 40,
            "DeviceName": "SystemBoard",
            "DataSourceUri": "/redfish/v1/Chassis/1U/Sensors/SysBrdTemp"
        },
        {
            "Reading": 24.8,
            "DeviceName": "Intake",
            "DataSourceUri": "/redfish/v1/Chassis/1U/Sensors/IntakeTemp"
        },
        {
            "Reading": 39,
            "DeviceName": "CPUSubsystem",
            "DataSourceUri": "/redfish/v1/Chassis/1U/Sensors/CPUTemps"
        },
        {
            "Reading": 42,
            "DeviceName": "MemorySubsystem",
            "DataSourceUri": "/redfish/v1/Chassis/1U/Sensors/MemoryTemp"
        },
        {
            "Reading": 33,
            "DeviceName": "PowerSupply",
            "DataSourceUri": "/redfish/v1/Chassis/1U/Sensors/PSTemp"
        },
        {
            "Reading": 40.5,
            "DeviceName": "Exhaust",
            "DataSourceUri": "/redfish/v1/Chassis/1U/Sensors/ExhaustTemp"
        }
    ],
    "Oem": {}
}
```

CoolingMetrics mockup


```
{
    "@odata.id": "/redfish/v1/Chassis/1U/ThermalSubsystem/CoolingMetrics",
    "@odata.type": "#CoolingMetrics.v1_0_0.CoolingMetrics",
    "Name": "Chassis Fan and Liquid Cooling Metrics",
    "FanPercentSummary": [{
            "DeviceName": "Chassis Fan #1",
            "Reading": 45,
            "SpeedRPM": 1900,
            "DataSourceUri": "/redfish/v1/Chassis/1U/Sensors/FanBay1"
        },
        {
            "DeviceName": "Chassis Fan #2",
            "Reading": 55,
            "SpeedRPM": 2100,
            "DataSourceUri": "/redfish/v1/Chassis/1U/Sensors/FanBay2"
        }
    ],
    "Oem": {}
}
```

Fan mockup


```
{
    "@odata.id": "/redfish/v1/Chassis/1U/ThermalSubsystem/Fans/Bay1",
    "@odata.type": "#Fan.v1_0_0.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",
    "LocationIndicatorActive": true,
    "HotPluggable": true,
    "SpeedPercent": {
        "Reading": 45,
        "SpeedRPM": 2200,
        "DataSourceUri": "/redfish/v1/Chassis/1U/Sensors/FanBay1"
    },
    "Location": {
        "PartLocation": {
            "ServiceLabel": "Chassis Fan Bay 1",
            "LocationType": "Bay",
            "LocationOrdinalValue": 0
        }
    }
}
```

SENSOR INTEGRATION

User-defined Thresholds

- Add User-defined Threshold support in Sensor
  - Clearly define service vs user-defined thresholds
  - Implementation may support user-defined thresholds for each sensor
  - Some existing *Thresholds* usage may move to *UserThresholds*
    - If user can define reaction behavior (but perhaps not change value)
- Adds to *Thresholds* structure with additional User instances
  - *UpperCautionUser, UpperCriticalUser, LowerCautionUser, LowerCriticalUser*

Integration using Schema Excerpts

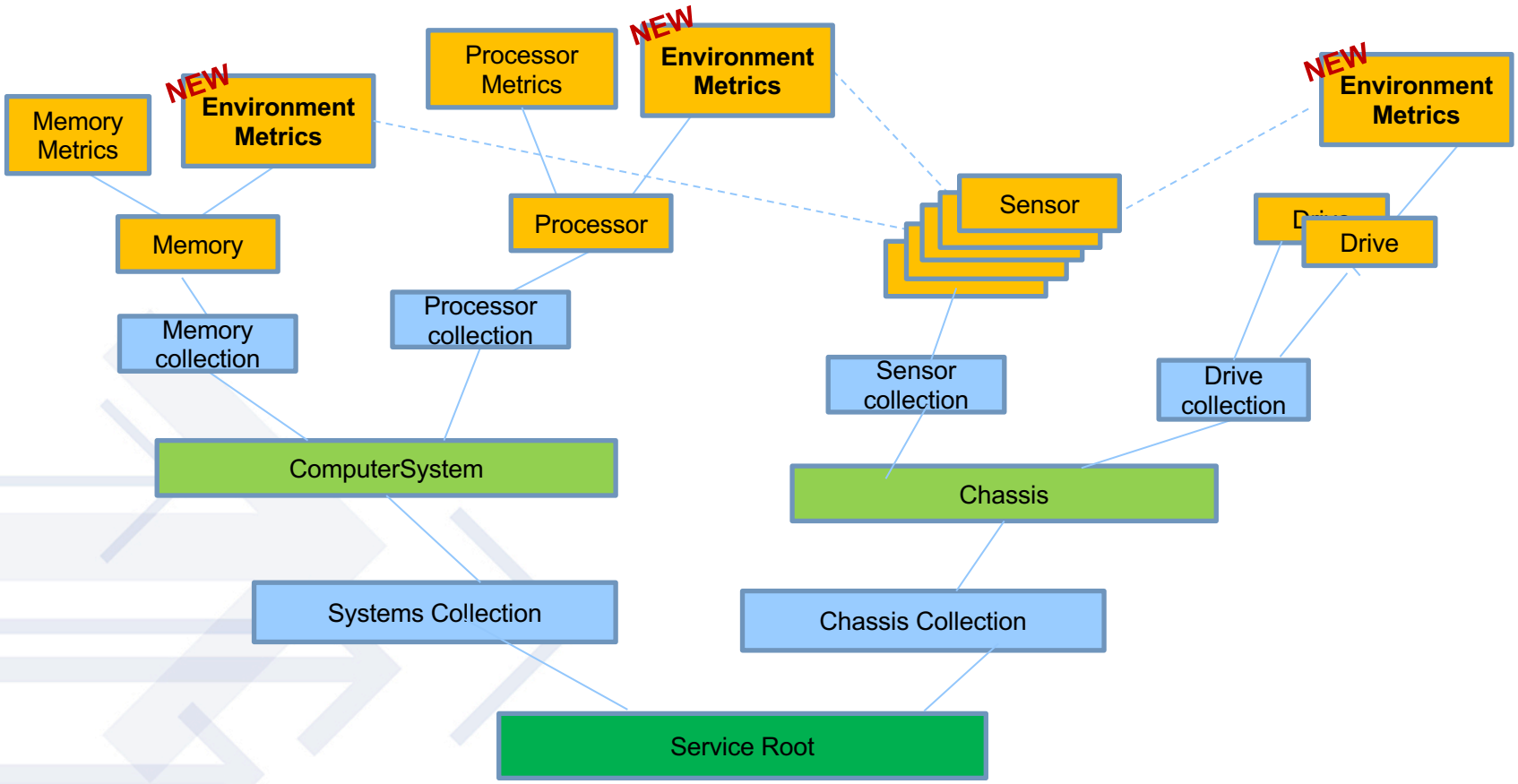
- Reference to a single Sensor is simple and low impact

```
"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/437XR1138R2/Processors/1/EnvironmentMetrics",
    "@odata.type": "#EnvironmentMetrics.v1_0_0.EnvironmentMetrics",
    "Name": "Processor Environment Metrics",
    "TemperatureCelsius": {
        "DataSourceUri": "/redfish/v1/Chassis/1U/Sensors/CPU1Temp",
        "Reading": 44
    },
    "PowerWatts": {
        "DataSourceUri": "/redfish/v1/Chassis/1U/Sensors/CPU1Power",
        "Reading": 12.87
    },
    "FanSpeedPercent": {
        "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 and replace with new EnvironmentMetrics property
  - *ThrottlingCelsius* should migrate to a Threshold value in Sensor
    - May be multiple values? Caution and Critical?
    - May also be able to use the proposed Control schema for this
- Processor Power
  - *ConsumedPowerWatt* already defined in ProcessorMetrics
    - Deprecate and replaced with new EnvironmentMetrics property

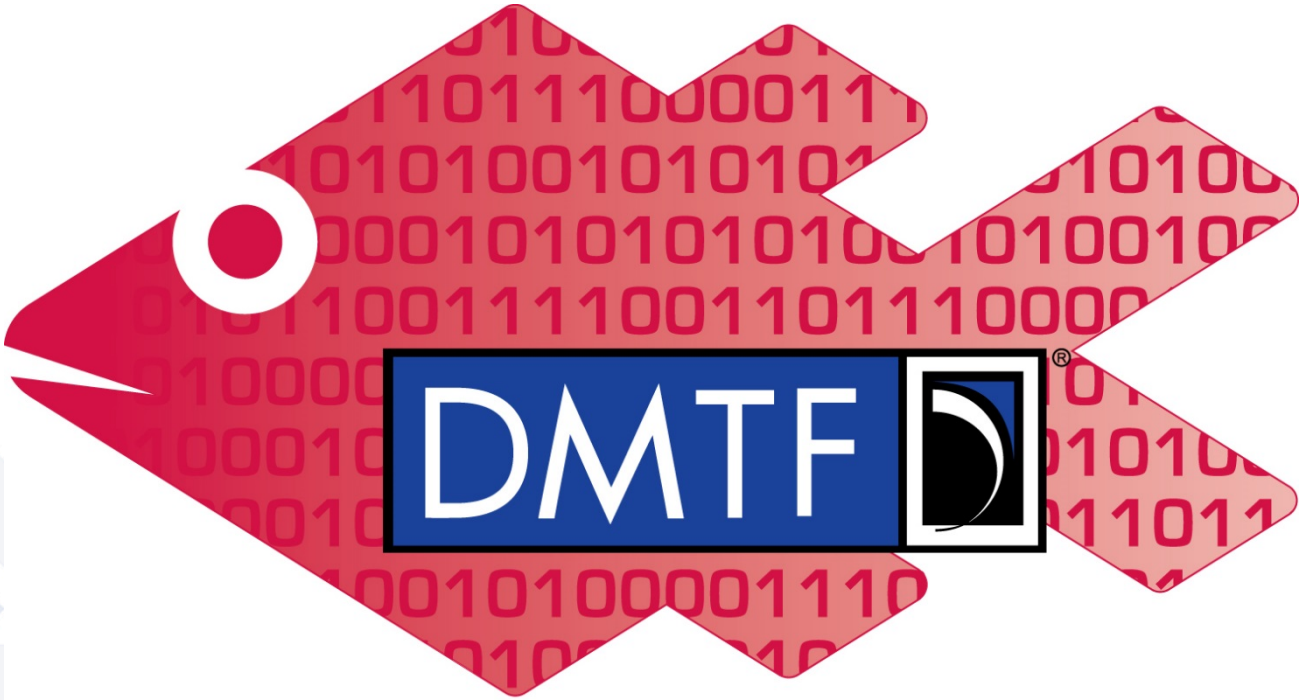
Memory sensor integration

- Add EnvironmentMetrics resource under Memory
  - Expect *TemperatureCelsius* and *PowerWatts* to be populated
- 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” sensor data?

- 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
  - Need interoperable terms that are commonly implemented for this to be useful
  - See ThermalMetrics temperatures as an example
- Arrays are more complex to parse, but highly flexible
  - Use this structure to allow quick retrieval of “all sensors per type”
- For v0.9 work-in-progress:
  - *TotalPowerWatts* and *EnergykWh* are shown in PowerSubsystemMetrics
  - *FanPercentSummary* is an array of fan speeds in CoolingMetrics
  - *TemperatureCelsius* is a structure in ThermalMetrics
    - This structure defines temperatures at locations used by ASHRAE and others
  - *TemperatureSummaryCelsius* is an array of temperature sensors in ThermalMetrics

Q&A & Discussion



Redfish