Redfish Telemetry Streaming and Reporting

WORK IN PROGRESS

Jeff Autor, Vertiv

Redfish Forum, October 2025

Redfish

www.dmtf.org

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: www.dmtf.org

Telemetry proposal goals

- Produce interoperable telemetry in formats easily consumed by time series database packages or Redfish-aware clients
  - Need end user feedback and any client-side requirements
- Enable correlation with Redfish data model
  - Match telemetry data to source in Redfish resources and properties
- Configure & deploy telemetry streaming without a priori knowledge of resource tree or require vendor-specific string matching
  - Should be able to “blind deploy” telemetry configuration across fleet
- Support stream / poll models and report generation
  - Polling still useful for testing or ad hoc data gathering…

Telemetry enhancement proposal components

- **NEW** Time series-focused output format
  - One “metric” per record (TLV), bundle of records delivered together
  - Specify “metric name” and “metric label” string formats for interop
- **TelemetryFeed** definition
  - Revamped proposal to easily define the contents of a stream
  - Provide better leverage and integration from MetricReportDefinition
- Resource subset output format (from original proposal)
  - Deliver “telemetry-only” resource subset on change or time interval
  - Schema annotations to define telemetry subsets

TIME SERIES RECORDS

NEW TimeSeriesRecord

- Schema (non-resource) definition to describe telemetry output
- Contains time-label-value (TLV) records
- *Records[{…}]* contains:
  - *Time* : Epoch-format (ms) (or seconds as float?)
  - *Name* : Metric name constructed from schema
  - *Label* : Instance name constructed from location in resource tree
  - *Value* : Property value
  - *RedfishId* : Service-unique, durable identifier for Name-Label pair
- Possible option to omit Name/Label after first record for efficiency
  - Client stores lookup table based on *RedfishId* values

TimeSeriesRecord: *Name* construction

- Construct *Name* using schema and property names
  - Format: Redfish_<schema name>_[<object path>]_<property name>
  - Omit indexing of array properties (those are included in Label)
- Simplify frequent use of *Reading* in **Sensor** excerpt objects
  - Excerpt property name is sufficient, omit *Reading* as unnecessary
  - Example: `Redfish_PowerSupply_InputVoltage`*`[_Reading]`*
- Perhaps insert value of key property (if defined in schema)
  - Provide “type” granularity if needed (e.g. “Temperature_Sensor”)
    - Or this could be an exception just for Sensor schema
  - *Is this better addressed by proper resource selection in Feed?*

TimeSeriesRecord: *Name* and *Label* style choices

- Do we need to follow the “best practices” for *Name* and *Label*?
  - Those include “style” choices, but also rules to avoid name collisions
    - We would preface our names with “Redfish_” or perhaps “rf_”
    - Inclusion of schema (resource type) ensures unique names
  - Conversion requires client-side recovery of Redfish property names
- Following best practices of time series client package(s):
  - CamelCase becomes lower_case (some fixups needed: e.g. “kWh”)
  - Name: `rf_<schema name>[_<object name>]_<property name>`
  - Will need to document (few) exceptions as schema annotation
- Is this necessary, and/or can this be accomplished in exporter(s)?

TimeSeriesRecord: *Name* examples

- Native Redfish style:
  - `Redfish_PortMetrics_Networking_TXBroadcastFrames`
  - `Redfish_Temperature_Sensor`
  - `Redfish_Outlet_EnergykWh`
  - `Redfish_EnvironmentMetrics_HumidityPercent`
  - `Redfish_EnvironmentMetrics_FansSpeedPercent_SpeedRPM`
- Prometheus “Best practice” style:
  - `redfish_port_metrics_networking_tx_broadcast_frames`
  - `redfish_temperature_sensor`
  - `redfish_outlet_energy_kwh`
  - `redfish_environment_metrics_humidity_percent`
  - `redfish_environment_metrics_fans_speed_percent_speed_rpm`

Sensor example showing “subset type” prepended to schema name and Reading property name removed

TimeSeriesRecord: *Label* construction

- Construct using path to source property in Redfish resource tree
  - Only includes the fork “choices”, generating a minimal path
  - Starts with schema name and append identifiers at each fork choice
- Define identifiers for collection members and array properties
  - Use schema name or property name (when multiple paths in resource)
  - Same format questions as *Name*
- Provide *Label* as either a string or a structure (need feedback)
  - Don’t want to require string parsing to “break apart”
  - Adapter code could craft one from the other if desired
  - Structure likely an array of key-value pairs {schema_name : identifier}

TimeSeriesRecord: *Label* identifiers

- Need interoperable identifier values equivalent to “path to property”
  - X Resource collection *Id* values are opaque strings, not interoperable
  - X Resource collection *Members*[0..n] order may change on service reset
- Add *Ordinal* property for Resource Collection members
  - Ordinal values defined by service (0-based integer)
  - Values are stable unless “significant configuration change”, same as *Id*
  - Create annotation rather than property? *@Redfish.Ordinal*
  - Add *“?ordinal=<int>”* query parameter to retrieve resources without *Id*
- For array properties, add index (0-based) key-value to *Label*
  - Example: “*FanSpeedsPercent*=5” in **EnvironmentMetrics**

TimeSeriesRecord: *Name* and *Label* examples

- Redfish_PortMetrics_Networking_TXBroadcastFrames
  - Label: "ComputerSystem=0,FabricAdapter=1,Port=1"
- Redfish_PowerSupplyMetrics_OutputPowerWatts_ReactiveVAR
  - Label: "Chassis=0,PowerSupply=3"
- Redfish_Outlet_EnergykWh
  - Label: "RackPDUs=0,Outlet=9"
- Redfish_EnvironmentMetrics_HumidityPercent
  - Label: "Chassis=2"
- Redfish_EnvironmentMetrics_FansSpeedPercent_SpeedRPM
  - Label: "Chassis=2,FansSpeedPercent=7"

TimeSeriesRecord: Full record example

```
{
   “Time”: 1743984255000,
   “Records”: [{
      “Time": 1743984253000,
      “RedfishId”: “KAN3dr4”,
      “Name”: “Redfish_Temperature_Sensor”,
      “Label”: “Chassis=0,Sensor=17”,
      "Value": “46.1”,
      },
      {
      “Time": 1743984253000,
      “RedfishId”: “LK9hy72”,
      “Name”: “Redfish_Temperature_Sensor”,
      “Label”: “Chassis=0,Sensor=33”,
      "Value": “29.7”,
      },
      . . . More records ...
    ]
}
```

Adopt millisecond-granularity epoch-style timestamps (bundle plus per-record) – shown as Int64 (ms), could be float (s)

RedfishId provides service-unique identifier, allowing option to omit Name and Label after first record for efficiency

TELEMETRY FEEDS

Selecting properties for a telemetry feed

- Redfish separates fast-changing data into separate resources
  - But even these resources include some static, supporting properties
- Choose telemetry-focused subsets of properties for each schema
  - Omit configuration data, supporting properties, links to resources, etc.
  - Example: In **Sensor**, normally, only the *Reading* value changes
- Define these subsets as part of the standard schema
  - Schema annotation concept unchanged from previous proposal
- Telemetry feed contains the subset defined for a selected resource
  - Simplifies feed creation by pre-selecting properties per schema
  - A resource instance of this subset at a given time is a “record”

NEW TelemetryFeed schema

- A feed is produced using a schema-backed subset of properties from a selected set of resources in the tree
  - URI patterns (with wildcards) and “type” filters select resources
  - Schema annotations define which properties are included
    - Can receive “Compact”, “Detailed”, or “All” subset of properties
  - Concept is mostly unchanged from previous proposal
- Local “reports” built from feed are placed in **TelemetryData**
  - Provides means to create “daily” / “hourly” reports stored on service
  - Includes controls for report generation, duration, etc.
  - Feed can also be used to define OEM-specific TelemetryData output

TelemetryFeed mockup

```
{
    "@odata.type": "#TelemetryFeed.v1_0_0.TelemetryFeed",
    "Id": “HourlyTemperature",
    "Name": "Temperature data from 1U servers collected every hour",
    "Enabled": true,
    “PollTimeSeriesURI”: “/redfish/v1/TelemetryService/TelemetryFeeds/HourlyTemperature/TimeSeries”,
    “PollJSONLinesURI”: “/redfish/v1/TelemetryService/TelemetryFeeds/HourlyTemperature/JSONLines”,
    "FeedSources": [
        {
            "URIPattern": "/redfish/v1/Chassis/{1U_*}/Sensors",
            "FilterKeyValues": [ "Temperature“ ]
        }
    ],
    "SamplingInterval": "PT60M",
    "SamplingType": "Interval",
    "Verbosity": "Compact",
    "IncludeStatus": false,
    "TelemetryDataOutput": {
        "Enabled": true,
        "Duration": "PT24H",
        “Format”: “TimeSeries”,
        "StartTime": "2024-09-01T00:00",
        "KeepAtMost": 4
    },
    "@odata.id": "/redfish/v1/TelemetryService/TelemetryFeeds/HourlyTemperature“
}
```

Enable predictable polling support, a GET returns a single set of record(s)

Array allows collecting telemetry from multiple resources (with wildcard support) in a single feed

A “Report” is a JSON Lines concatenated file of telemetry records (in any supported format). Reports built from the feed are placed in TelemetryData collection

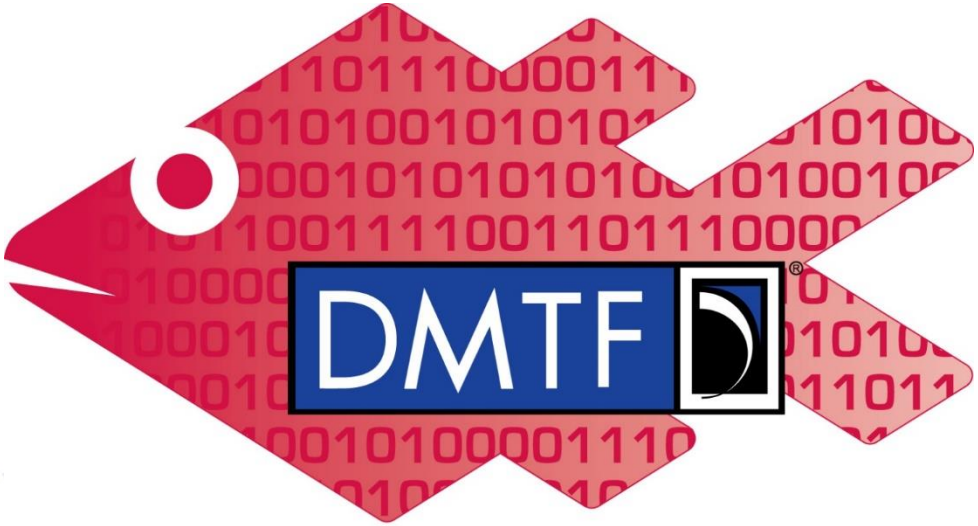
Subscribing to TelemetryFeed

- New *EventFormatType* values in **EventDestination**
  - "TimeSeriesRecord" : receive **TimeSeriesRecord**
  - "TelemetryFeed" : receive telemetry subsets in JSON Lines
- *TelemetryFeedId – Id* value of **TelemetryFeed**
  - Can be used for "MetricReport" format as well
- Supports both POST Event method or Server-Sent Eventing (SSE)
- HTTP headers define content type and subscriber context
  - `Content-Type: application/jsonlines`
  - `X-Redfish-Context: <context>` ← Custom HTTP Header defined for this purpose

Call to Action

- Watch for updated work-in-progress bundle for more details
  - See: https://www.dmtf.org/redfish under “Work in Progress”
- Evaluate “Name” and “Label” construction concepts
  - Are there existing formats to which we can align?
  - Do we need to support multiple formats / styles?
  - Any telemetry package-specific needs beyond TLV fields?
- Provide feedback on updated proposal
  - *Need your requirements to ensure we create an interoperable solution*
  - *What else is needed to produce interoperable telemetry?*

Q&A & Discussion



Redfish