Redfish Telemetry Streaming and Reporting

WORK IN PROGRESS

Redfish Forum, September 2024 v0.8

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

Getting involved in Redfish

- Redfish Standards page
  - Schemas, Specs, Mockups, White Papers & more
  - http://www.dmtf.org/standards/redfish
- Redfish Developer Portal
  - Redfish Interactive Resource Explorer
  - Educational material, documentation & other links
  - http://redfish.dmtf.org
- Redfish User Forum
  - User forum for questions, suggestions and discussion
  - http://www.redfishforum.com
- DMTF Feedback Portal
  - Provide feedback or submit proposals for Redfish standards
  - https://www.dmtf.org/standards/feedback
- DMTF Redfish Forum
  - Join the DMTF to get involved in future work
  - http://www.dmtf.org/standards/spmf

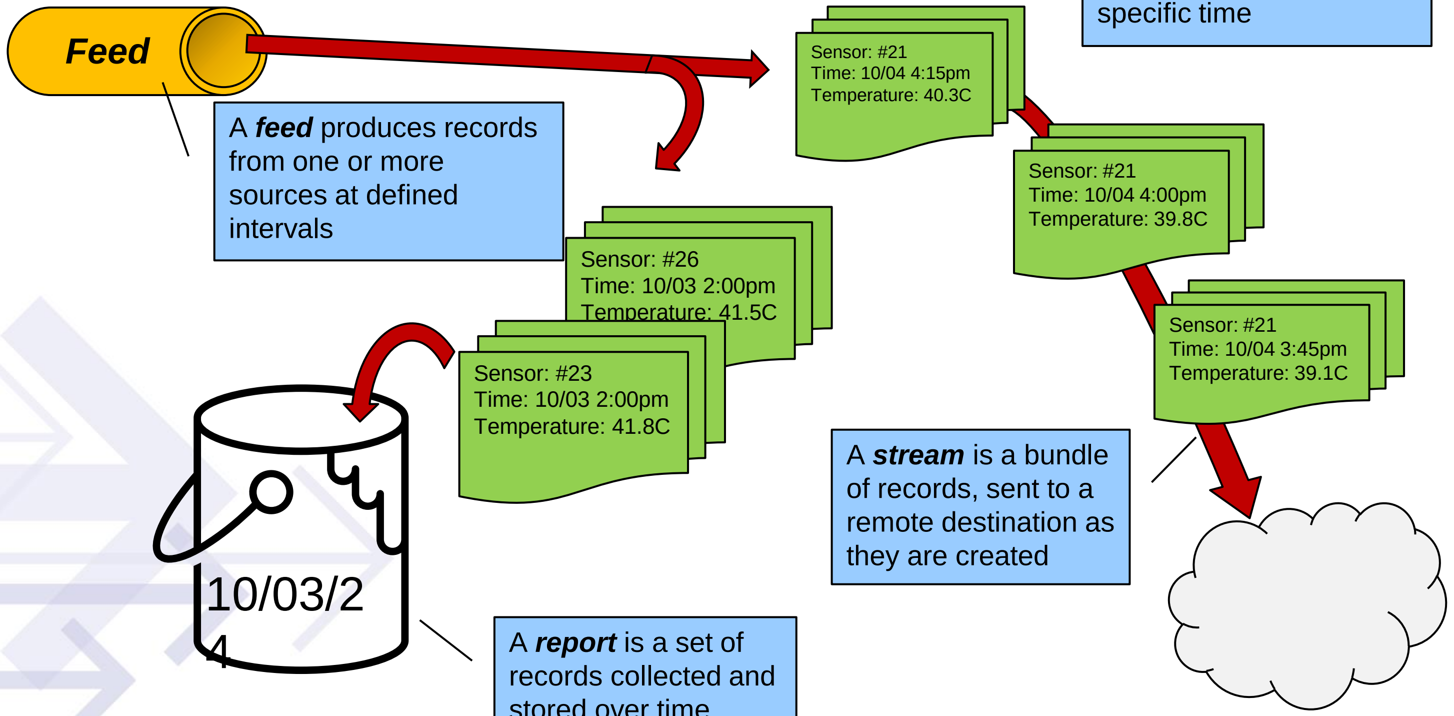
Introduction

- Redfish is built for inventory, configuration, and ad hoc monitoring
  - Continuous polling Redfish resources for telemetry is not efficient
- Existing **TelemetryService** and **MetricReport** needs improvement
  - Implementations tend to only support vendor-defined, non-interoperable reports
  - Resulting report data cannot be easily correlated with Redfish resources
    - Contents of the report are not tied to their source in the Redfish data model
  - Requires a priori knowledge of product / service to set up
    - Cannot deploy same report definition across a multivendor fleet
- Desire to increase ecosystem adoption and interoperability
  - Telemetry support must operate well with popular telemetry clients
    - E.g. Prometheus, Telegraf, OpenTelemetry, etc.
  - Need a simpler scheme to encourage support on small-footprint devices

Requirements

- Support three methods of gathering telemetry:
  - **Polling** – Client performs GET on resource(s) with minimal overhead
  - **Streaming** – Service sends optimized bundles of data at regular intervals
  - **Reporting** – Service records data over time, periodically produces a report
- Ability to create vendor and device-independent reports
- Ability to “blind deploy” telemetry configuration with no a priori knowledge
  - POST to create a configuration, POST to create a subscription
- Preserve and leverage the investment in the Redfish data model
  - Output must match, or enable client to transform to, resource definitions
- Encourage adoption and interoperability
  - Minimize creation of new resources or structures
  - Minimize new functions or implementation options to encourage adoption

Terminology



A ***record*** contains data from a single source at a specific time

A ***feed*** produces records from one or more sources at defined intervals

A ***stream*** is a bundle of records, sent to a remote destination as they are created

A ***report*** is a set of records collected and stored over time

TELEMETRY RECORDS

Selecting properties for telemetry

- Many properties in Redfish resources are static data for a given instance, or only updated upon configuration or state changes
- 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
  - Ensures client can correlate the subset with the full resource
  - An instance of this subset retrieved at a given time is a “record”

Telemetry verbosity

- Record definition must balance efficiency vs. completeness
  - Dashboards, control systems, and other real-time users desire efficiency due to higher-frequency sampling rates
  - Analysis tools desire detailed, complete data at lower sampling rates
- Two verbosity levels defined for each schema or resource
  - “Compact” – Data likely to change given expected sampling rates
    - Intent is to minimize payload for efficiency
    - Example: Sensor readings, utilization levels, performance counters
  - “Detailed” – Adds data less likely to change, but useful for analysis
    - Provide any non-static data that could be classified as “telemetry”
    - Example: Device state, error counters, average/low/peak readings

Telemetry record definitions (1 of 2)

In general, “Compact” records contain sensor readings and key performance counters
“Detailed” records add Status (State/Health), additional performance and error counters

- **BatteryMetrics** – charge rates, current, voltage
- **Circuit, Outlet, PowerDistributionMetrics** – current, voltage, power, energy, frequency
- **CoolantConnector, CoolingLoop** – pressure, flow rate, temperature
- **DriveMetrics –** Read/write counters, uncorrectable error counts, power, temperature
    - **Detailed**: adds corrected errors, NVMe statistics
- **EnvironmentMetrics** – power, energy, temperature, humidity
- **HeaterMetrics –** heating time, power, temperature
- **MemoryMetrics -** Read/write counters, uncorrectable error counts, power, temperature
    - **Detailed:** adds corrected error counters, predicted media life
- **NetworkAdapterMetrics –** TX/RX counters
    - **Detailed:** Specific counters for NCSI, Multicast/Unicast TX/RX, etc
- **NetworkDeviceFunctionMetrics –** TX/RX counters
    - **Detailed:** Specific counters for FibreChannel, Multicast/Unicast TX/RX, etc

Telemetry record definitions (2 of 2)

- **PortMetrics -** TX/RX counters
  - **Detailed:** Specific counters for FibreChannel, GenZ, Transceivers, Multicast/Unicast TX/RX, etc.
- **PowerSupplyMetrics** – input and output: current, voltage, power, energy, frequency
- **ProcessorMetrics -** TX/RX counters
  - **Detailed:** Specific counters for NCSI, Multicast/Unicast TX/RX, etc.
- **Pump –** Speed, speed control
- **Reservoir** – Fluid level, pressure
- **Sensor –** Reading, apparent power/energy
  - **Detailed:** Average / lowest / peak Reading
- **StorageControllerMetrics –** Read/write bytes and units, uncorrected error counts
  - **Detailed**: NVMe SMART properties, correctable error counts
- **ThermalMetrics –** Power, energy, temperature readings
  - **Detailed**: Heater usage, lifetime readings

Example: Sensor telemetry record

```
GET /redfish/v1/Chassis/1/Sensors/ServerTemp?telemetry=Compact



{
   “@odata.id”:  "/redfish/v1/Chassis/1/Sensors/ServerTemp",
   “@Redfish.Time”: 1696261238,
   "Reading": 21.3
}
```

New *@Redfish.Time* annotation included in telemetry payload (this will be covered later)

For a **Sensor** resource, the *Reading* is the primary piece of data, which can change frequently.

Additional sensor data (average, peak, lowest values) would be available in the “detailed” telemetry record

@odata.type and other schema-required properties are not included – since client explicitly requested the telemetry subset and therefore is aware of that result

Example: Outlet record

```
{
   “@Redfish.Id”:  “OutA3",
   “@Redfish.Time”: 1696261238,
   "Voltage": {
      "Reading": 202.3
   },
   "CurrentAmps": {
      "Reading": 1.73
   },
   "PowerWatts": {
       "Reading": 349.9,
       "ApparentVA": 349.9,
       "ReactiveVAR": 0.1,
       "PowerFactor": 0.99
   },
   "EnergykWh": {
       "Reading": 61848
   }
}
```

@Redfish.Id annotation replaces @odata.id to reduce payload size after first record in a stream or report

TELEMETRY ANNOTATIONS

Payload and Schema annotation additions to support telemetry

NEW Telemetry schema annotation

- Telemetry record contents are defined in schema
  - As definition is backed by schema, clients can correlate received telemetry records to full copies of resources gathered separately
  - Record contents defined for applicable standard resource types
- *Redfish.Telemetry* annotation tags property for record inclusion
  - Property will appear in each telemetry record, whenever retrieved
  - The property does not become required, and the content of each record reflects the properties supported by each resource instance
  - Annotation indicates the verbosity level when the property appears:
    - “Compact” – Included in all telemetry records
    - “Detailed” – Included only in extended detail record requests

NEW Telemetry Filter Key schema annotation

- *Redfish.TelemetryFilterKey* annotation enables filtering
  - One property, per resource type, is defined in schema as the filter key
    - Typically the "<*Thing*>*Type*" property in a schema
- Value of key property filters the resources for a telemetry feed
  - Ex: *ReadingType* in **Sensor** allows filtering for "Temperature" sensors
- In some cases, filter keys are defined in the parent resource
  - E.g. create a telemetry feed for "**DriveMetrics** of all NVMe drives"
    - *Protocol (*Drive "type"*)* is in the parent **Drive** resource, not **DriveMetrics**
  - These properties will be duplicated in those "metrics" schemas
    - **DriveMetrics** (*Protocol*), **MemoryMetrics** (*MemoryType*)
    - **ProcessorMetrics** (*ProcessorType*), **PortMetrics** (*PortProtocol*)

NEW Telemetry query parameter

- Returns the telemetry record subset instead of the entire resource
  - “telemetry” parameter with value of “Compact”, “Detailed”, or “All”
- Enables “polling” for simple clients or discovery of record contents
  - Ex: `GET /redfish/v1/Chassis/1/Sensors/MainPower?telemetry=Compact`
- Service ignores this parameter if not supported
  - Simply returns the entire resource as normally expected
- Also used to gather telemetry from data providers
  - Data provider must be aware of the telemetry record definition
  - Service can gather telemetry without a priori knowledge of schemas
  - “All” value allow retrieval of entire resource with telemetry annotations
    - Payload includes *@Redfish.Time* and *@Redfish.Id* for easy aggregation

NEW Time payload annotation

- *@Redfish.Time* annotation created for telemetry records
  - Added to every telemetry record to report the data acquisition time
  - Intended for data provider to include in returned telemetry payloads, but telemetry service would insert annotation if not provided
- Uses the UNIX epoch (uint64) format for compactness
  - Based on UTC time for consistency across services
  - Clients are programs (human readability not a factor)
  - Lighter payload and reduced processing
- Returned in GET requests that include *telemetry* query parameter
- Example: `“@Redfish.Time”: 1696261238`

NEW Resource Identifier for reduced payload size

- User option to lightly encode *@odata.id* as *@Redfish.Id* in payload
  - *@odata.id* always appears in first record of a stream or report
    - Further records omit *@odata.id* from payload
- *@Redfish.Id* contains a hash of *@odata.id* for each resource
  - This significantly reduces string length of this required, static data
  - Service can create a hash or can use any unique replacement string
  - Value must be unique within the service (not universal)
- Returned in GET requests using the *telemetry* query parameter
- Examples for: `"/redfish/v1/PowerEquipment/RackPDUs/1/Outlets/A3“`
  - `“@Redfish.Id”: “ZjU1YT”    (truncated base64-encoded SHA-1 hash)`
  - `“@Redfish.Id”: “OutA3”     (service-defined replacement string)`

TELEMETRY FEEDS

GENERATING TELEMETRY RECORDS

Telemetry feeds

- Allow creation of telemetry ***feeds*** that continuously produce telemetry records from selected resources at a defined interval
  - Service can also pre-define feeds ready for subscribers
- Each feed produces records delivered to subscribers
  - A single record contains the subset of properties from any applicable resource instance defined for the feed
- Each feed can serve multiple types of subscribers:
  - A stream of telemetry records sent as they are created
  - A report created from records stored over a defined interval
- Regardless of destination, the format of the feed is the same

JSON Lines

- JSON Lines is the chosen record format
  - A stream consists of a bundle of records in JSON Lines format
  - A report is a JSON Lines-formatted file
- Allows easy concatenation of multiple JSON documents
  - Appeared ~2017 from post-SQL database crowd (Apache Spark, etc.)
  - Simple description: “CSV for JSON”, see https://jsonlines.org
- Allows for simple accumulation of multiple JSON payloads
  - Strip every “\n” from JSON payload, append to file, add “\n”, repeat…
- Well-supported by open source tools, libraries, etc.
  - Barely need more “Readline File I/O” support + JSON encoder

Sample JSON Lines file of Sensor telemetry records

#	Telemetry report file contents in JSON Lines format
1	{ "@odata.id": "/redfish/v1/Chassis/1/Sensors/Temp", "Reading": 41.7, "@Redfish.Time": 1696282838 }</n>
2	{ "@odata.id": "/redfish/v1/Chassis/1/Sensors/CPU1Temp", "Reading": 46.9, "@Redfish.Time": 1696282838 }</n>
3	{ "@odata.id": "/redfish/v1/Chassis/1/Sensors/CPU2Temp", "Reading": 48.2, "@Redfish.Time": 1696282838 }</n>
4	{ "@odata.id": "/redfish/v1/Chassis/1/Sensors/Temp", "Reading": 41.7, "@Redfish.Time": 1696283136 }</n>
5	{ "@odata.id": "/redfish/v1/Chassis/1/Sensors/CPU1Temp", "Reading": 46.9, "@Redfish.Time": 1696283136 }</n>
6	{ "@odata.id": "/redfish/v1/Chassis/1/Sensors/CPU2Temp", "Reading": 48.2, "@Redfish.Time": 1696283136 }</n>
7	{ "@odata.id": "/redfish/v1/Chassis/1/Sensors/Temp", "Reading": 41.7, "@Redfish.Time": 1696283431 }</n>
8	{ "@odata.id": "/redfish/v1/Chassis/1/Sensors/CPU1Temp", "Reading": 46.9, "@Redfish.Time": 1696283431 }</n>
9	{ "@odata.id": "/redfish/v1/Chassis/1/Sensors/CPU2Temp", "Reading": 48.2, "@Redfish.Time": 1696283431 }</n>

Sample telemetry report using *@Redfish.Id*

#	Telemetry report file contents in JSON Lines format
1	{"@odata.id": "/redfish/v1/Chassis/1/Sensors/Temp", "@Redfish.Id": “JK893F", "Reading": 41.7, "@Redfish.Time": 1696282838 }</n>
2	{"@odata.id": "/redfish/v1/Chassis/1/Sensors/CPU1Temp", "@Redfish.Id": “U97WR3”, "Reading": 46.9, "@Redfish.Time": 1696282838 }</n>
3	{"@odata.id": "/redfish/v1/Chassis/1/Sensors/CPU2Temp", "@Redfish.Id": “N5TR4C", "Reading": 48.2, "@Redfish.Time": 1696282838 }</n>
4	{ "@Redfish.Id": "JK893F", "Reading": 41.7, "@Redfish.Time": 1696283136 }</n>
5	{ "@Redfish.Id": "U97WR3", "Reading": 46.9, "@Redfish.Time": 1696283136 }</n>
6	{ "@Redfish.Id": "N5TR4C", "Reading": 48.2, "@Redfish.Time": 1696283136 }</n>
7	{ "@Redfish.Id": “JK893F", "Reading": 41.7, "@Redfish.Time": 1696283431 }</n>
8	{ "@Redfish.Id": “U97WR3", "Reading": 46.9, "@Redfish.Time”: 1696283431 }</n>
9	{ "@Redfish.Id": "N5TR4C", "Reading": 48.2, "@Redfish.Time": 1696283431 }</n>

NEW TelemetryFeed schema

- *DataFeedSources* [{ }]– describe the data to gather
  - *BaseURI* – Gather records starting at this URI, traverse down tree
  - *ResourceType* – The resource type (schema) to gather (e.g. **Sensor**)
  - *DetailedRecords* – Provide “Compact” or “Detailed” records (Boolean)
  - *FilterKeyValues*[] – Include record if “key property” matches value(s)
    - Example: **Sensor** defines *ReadingType* as key, match “Temperature”
- *Schedule{ } – frequency of sampling and start time for the feed*
  - *RecurrenceInterval* – Sampling frequency of the resource(s)
  - *StartTime* – Provide means to sync reports and samples
    - Specifying the start time can align the samples across a fleet

NEW TelemetryFeed schema, continued

- *HashIdentifiers* – Option to reduce payload size of feed
- *IncludeEntireResource – Option to include full resource in feed*
  - *"Once" – First record per resource includes entire payload, all further records include just the telemetry-defined properties*
  - *"Always" – Entire payload is included in the feed*
  - *"Never" – Records include only the telemetry properties*
- *ReportDuration* – Duration for each report
- *LocalReportsEnabled* – Service generates and stores reports
- *Reports* [{}] – describes available reports stored locally by service
  - URI for the file, creation time, file size

NEW TelemetryFeed mockup

```
{
    "@odata.type": "#TelemetryFeed.v1_0_0.TelemetryFeed”,
    "Id": “Temperature",
    “Name”: “Temperature Sensor Telemetry”,
    “TelemetryFeedId”: “Temperature”,
    “Enabled”: true,
    “LocalReportsEnabled”: true,
    “LocalReportsKeepAtMost”: 3,
    “DataFeedSources”: [{
        “BaseURI”: “/redfish/v1/Chassis”,
        “ResourceType”: “Sensor”,
        “DetailedRecords”: false,
        “FilterKeyValues”: [“Temperature”]   ← Filter by “ReadingType” for Sensor
     }],
     “IncludeEntireResource”: “Once”,   ← “Once”, “Never”, “Always”
     “HashIdentifiers”: true      ← Replaces “@odata.id” with “@Redfish.Id”
     “ReportDuration”: “PT24H”,
     “Schedule”: {
        “RecurrenceInterval”: “PT5M”,
        “InitialStartTime”: “2023-10-03T00:00”,  ← Ability to sync reports across many services
     },
     “Reports”: [ {
         “ReportURI”: “/redfish/v1/TelemetryService/Temperature-20231003T0000.jsonl”,
         “StartTime”: “2023-10-03T00:00”,
         “SizeBytes”: 23426
     }]
}
```

User-supplied *TelemetryFeedId* ensures reports can be referenced without prior knowledge of the *Id* values

Array allows multiple resource types in a single report, primarily for use cases where identical / equivalent properties exist across multiple schemas

TelemetryService and EventService additions

- **NEW** properties for discovery and configuration of **TelemetryFeed**
- **EventService**
  - *SupportedTransferProtocols* – Protocols supported for remote reports
- **TelemetryService**
  - *MaxTelemetryReportSizeBytes* – Maximum size of an individual report that can be held in memory before written to storage (local or remote)
  - *TotalLocalStorageBytes* – Total amount of local storage for reports
  - *AvailableLocalStorageBytes* – Available storage
  - *TelemetryReportOverwritePolicy* – Stop collecting or overwrite oldest

TELEMETRY SUBSCRIPTIONS

GENERATING TELEMETRY RECORDS

NEW Streaming telemetry support

- New *EventFormatType* of “TelemetryFeed” in **EventDestination**
  - Subscribe to receive telemetry feed in JSON Lines format
  - Supports both POST Event method or Server-Sent Eventing (SSE)
  - *TelemetryFeedId* references the specific telemetry feed to receive
    - Create a subscriptions without searching **TelemetryFeedCollection**
- POST Event payload is a JSON Lines bundle
- HTTP headers define content type and subscriber context
  - `Content-Type: application/jsonlines`
  - `X-Redfish-Context: <context>     ← Custom HTTP Header defined for this purpose`

EventDestination mockup for SSE telemetry stream

```
{
    "@odata.id": "/redfish/v1/EventService/Subscriptions/42"
    "@odata.type": "#EventDestination.v1_16_0.EventDestination",
    "Id": “42",
    "Name": “Telemetry streaming for power measurements",
    "Destination": "http://www.dnsname.com/Destination1",
    “EventFormatType": “TelemetryFeed",
    “SubscriptionType”: “SSE”,
    "DeliveryRetryPolicy": "TerminateAfterRetries",
    "Status": {
        "State": "Enabled"
    },
    "Context": "WebServer3",
    "Protocol": "Redfish",
    “TelemetryFeedIds”: [ “Power” ]
}
```

User-supplied *Context* ensures the event destination can identify the telemetry when feeds are received from multiple sources

TelemetryFeed SSE payload example

SSE subscription request:

```
GET https://192.168.1.32/sse-uri?$filter=TelemetryFeed eq ‘Temperature’
```

SSE event stream:

```
id: 1
data: {"@odata.id": "/redfish/v1/Chassis/1/Sensors/Temp", "@Redfish.Id": “JK893F", "Reading": 41.7, "@Redfish.Time": 1696282838 }
data: {"@odata.id": "/redfish/v1/Chassis/1/Sensors/CPU1Temp", "@Redfish.Id": “U97WR3”, "Reading": 46.9, "@Redfish.Time": 1696282838 }
data: {"@odata.id": "/redfish/v1/Chassis/1/Sensors/CPU2Temp", "@Redfish.Id": “N5TR4C", "Reading": 48.2, "@Redfish.Time": 1696282838 }


id: 2
data: { "@Redfish.Id": "JK893F", "Reading": 41.7, "@Redfish.Time": 1696283136 }
data: { "@Redfish.Id": "U97WR3", "Reading": 46.9, "@Redfish.Time": 1696283136 }
data: { "@Redfish.Id": "N5TR4C", "Reading": 48.2, "@Redfish.Time": 1696283136 }


id: 3
data: { "@Redfish.Id": “JK893F", "Reading": 41.7, "@Redfish.Time": 1696283431 }
data: { "@Redfish.Id": “U97WR3", "Reading": 46.9, "@Redfish.Time": 1696283431 }
data: { "@Redfish.Id": "N5TR4C", "Reading": 48.2, "@Redfish.Time": 1696283431 }
```

EventDestination mockup for Redfish Event style

```
{
    "@odata.id": "/redfish/v1/EventService/Subscriptions/43"
    "@odata.type": "#EventDestination.v1_16_0.EventDestination",
    "Id": “43",
    "Name": “Telemetry streaming for temperature and power measurements",
    "Destination": "http://www.dnsname.com/Destination1",
    “EventFormatType": “TelemetryFeed",
    "SubscriptionType": “RedfishEvent”,
    "DeliveryRetryPolicy": "TerminateAfterRetries",
    "Status": {
        "State": "Enabled"
    },
    "Context": "WebUser3",
    "Protocol": "Redfish",
    “TelemetryFeedIds”: [ “Temperature”, “Power” ]
}
```

Local and Remote-delivered Telemetry Reports

- Reports contain records from a single telemetry feed
  - Report delivered at the end of the specified reporting interval
- Service can store reports locally
  - User downloads using URIs provided in **TelemetryFeed**
  - *Reports* property provides information and links to all report instances
- Users can subscribe to a telemetry report
  - Create **EventDestination** with *EventFormatType* of “TelemetryReport”
  - *Destination* must be a file folder location the service can access
  - New properties added to supply credentials for remote location

Filename conventions for Telemetry Reports

- Subscriber-provided *Context* is used to construct filename to allow client to subscribe to feeds from multiple services
- Filename conventions for reports
  - Local reports: `<TelemetryFeedId>-<start time>.jsonl`
    - `Example: Temperature-20240411T0000.jsonl`
  - Remote clients: `<Context>-<TelemetryFeedId>-<start time>.jsonl`
    - `Example: WebServer3-Temperature-20240411T0000.jsonl`

EventDestination mockup for remote Telemetry Report

```
{
    "@odata.id": "/redfish/v1/EventService/Subscriptions/44"
    "@odata.type": "#EventDestination.v1_16_0.EventDestination",
    "Id": “44",
    "Name": “Telemetry report for temperature measurements",
    "Destination": “ftp://www.dnsname.com/reports/",
    “EventFormatType": “TelemetryFeed",
    "SubscriptionType": “FileTransfer",
    "DeliveryRetryPolicy": "TerminateAfterRetries",
    "Status": {
        "State": "Enabled"
    },
    "Context": "WebServer3",
    "Protocol": “Redfish",
    “TransferProtocol”: “FTP”,
    “Username”: “dumptruck”,
    “Password”: null,
    “Certificates”: { },
    “ClientCertificates”: { },
    “TelemetryFeedIds”: [ “Temperature” ]
}
```

Subscriber’s *Context* is used to construct report filename

New properties provide transfer protocol and credentials for placing file at destination

Single *TelemetryFeedId* value for the report

Q&A & Discussion

www.dmtf.org