TAPI v2.6.0 Reference Implementation Agreement

TR-548

TAPI Streaming

Version 3.2 (March 2025)

Document Type: Technical Recommendation

Table of Contents

List of Figures

List of Tables

Document History

Version	**Date**	**Description of Change**
1.1	December 2021	Initial version of the Reference Implementation Agreement document on streaming for TAPI v2.1.3
2.0	December 2022	Updated to cover TAPI v2.4.0.
3.0	October 2023	Updated to cover TAPI v2.5.0. Addition of: • Description of CHANGE_ONLY streaming. • Detailed model and use cases for PM streaming using gNMI/Protobuf.
3.1	October 2023	Corrections to Protobuf enumeration defaults (issue identified during TST review period).
3.2	March 2025	Updated to cover TAPI v2.6.0 with corrections for review comments on the RC. Note that the RC was unchanged from TAPI 2.5.

1 Introduction

1.1 General introduction

This LF Technical Recommendation (TR) is a supplement to the Reference Implementation for the TRANSPORT-API (TAPI) [LF TR-547].

1.2 Introduction to this document

The purpose of this document is to explain TAPI streaming and provide a set of guidelines and recommendations for use of TAPI streaming.

The target architecture for TAPI is provided in [LF TR-547]. The figure below is a copy of the figure provided in that document.

This document focuses on the autonomous flow of information via TAPI from SDN-C[1] to OSS/SDTN and from SDTN to OSS.

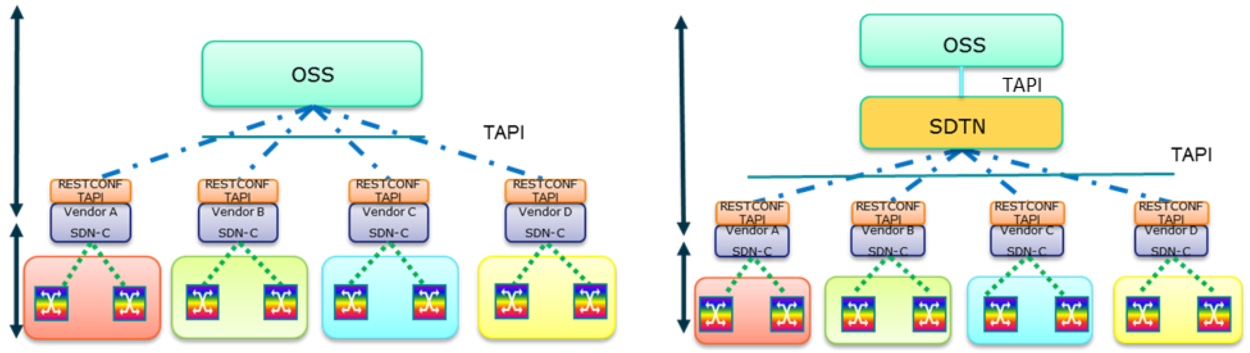


Figure 1 Example SDN architecture for WDM/OTN network

1.3 Specification

For TAPI, the YANG model files for a release are normative. All attributes other than those that cover identification of the entity/structure are essentially conditional. The corresponding UML model highlights which attributes are mandatory, and which are conditional mandatory[2]. For conditional mandatory attributes the UML model highlights the conditions for which the attribute must be present, and this is reflected in the description statement.

This document focusses on the use of TAPI streaming with compacted logs. The document indicates, for this usage, whether properties are mandatory, conditional mandatory or optional and in this respect this document is normative.

Behavioral aspects are not covered by the YANG model. This document covers behavioral aspects in section 7 Use Cases on page 88. Some uses cases are identified as normative and some informative.
Many parts of this document are simply informative and explanatory. Where this document is normative the words “normative” and/or “shall” and/or “must” will be used.

See section 12 Appendix: Changes between versions on page 148 for changes since the previous version of this document.

[1] See definitions in TR-547.
[2] Where mandatory as explained by the conditions stated, the attribute must be present an appropriately populated. Where not mandatory, the attribute does not need to be represented (and in most cases is not expected to be present).

2 Overview

2.1 Essential feature and benefits

Streaming is the name for a type of mechanism that handles the providing of information from one system to another in some form of steady and continuous flow[3].

In the context of a Management-Control solution streaming is used primarily for the reporting of ongoing change of state of the controlled system (and of other events from the controlled system) from one Management-Control entity to another (usually superior/client) Management-Control entity. In this context, as much of the information is derived from readings of instruments, the flow is often called telemetry[4].

The stream provides engineered flow such that peak load is reduced and spread using some mechanism such as back-pressure and/or selective pruning of detail.

In the following discussion various terms such as controller, client and provider will be used. For the usage in this document of these terms see section 10 Definitions and Terminology on page 145.
The definition allows for many alternative stream strategies using the same extensible structure. Streaming approaches are defined that:

- Focus on conveying TAPI global class instance (see also [LF TR-547] section on “TAPI Global and Local objects”), i.e., specific YANG sub-trees[5] (see 3.5 TAPI Application in detail on page 14).
- Provide an opportunity for event time reporting that is structured to allow for reporting of time uncertainty (see 4.8 Time on page 68).
- Allow a client[6] to achieve and maintain eventual consistency[7] with the state of the controlled system simply by connecting to the stream(s), i.e., with no need to retrieve[8] current state prior to processing log reports.
  - Makes it a provider[9] responsibility to send information to the client that ensures eventual consistency is achievable[10], removing sequencing complexity.
  - Improves provide-side scale as simply based on a time-sequenced log of events as opposed to a combination of a time-sequenced log and a repository[11].

  Note: There is still a need for the client to “mark and sweep” to achieve full system realignment on recovery from major loss of communications.
- Allow for efficient recovery from temporary streaming channel communication failures[12].
- Take pressure off of a client when under heavy load, recognizing that loss of some detail when under extreme pressure is inevitable and tolerable[13].
- Remove the need for complex subscription in normal controller-controller interaction.

[3] The flow rate and the presence depend upon the need to send information.
[4] This term loses relevance once the readings have been processed and abstracted but is often still used.
[5] For example, an instance of node-edge-point (global class with its UUID) includes all locally identified branches and leaves but NOT contained instances of connection-end-points, as the connection-end-point is a global class separately identifiable using its uuid. Note also that the attributes that reference (list) the contained global class instances (e.g., connection-end-point attribute in the node-edge-point) are NOT included. This information is conveyed by parent-address (see 3.10.3 Considering parent-address on page 29).
[6] See section 10 Definitions and Terminology on page 107 for usage of “client” in this document
[7] See 3.17 Eventual Consistency and Fidelity on page 29.
[8] For example, using RESTCONF GET.
[9] See section 10 Definitions and Terminology on page 107 for usage of “provider” in this document.
[10] Clearly the client implementation has to take advantage of this correctly as defined in the relevant use cases.
[11] This removes the need to grab a snapshot of the entire repository to service a full-alignment get and removes the need for an alternative get fragment approach
[12] See 7.2.7 Use Case ST-0.7: Short loss of communications on page 48.
[13] See 7.2.7 Use Case ST-0.7: Short loss of communications on page 48

- Are structured to support, in future versions of this specification[14], the:
    - Providing of multiple alternative views of the controlled solution[15]
    - Creation of streams for temporary flows of information such as one or more spotlights on detailed state change or snapshots of current state

The TAPI streaming approach is aligned in principle with the approach taken by the [GNMI] community with respect to the operation of the STREAM described in the [GNMI-SPEC].

2.2 TAPI application

TAPI Streaming can be used in several different applications. The primary application is one where a provider is offering an ongoing flow of state updates to a client, as depicted in Figure 1 above.

In this application the following assumptions apply:

- The client has one or more internal representations of the semantics (models) of the controlled system (network etc.). A representation may:
    - Relate to a subset of the TAPI model (e.g., just physical inventory)
    - Compress or expand parts of the model (e.g., Topology and Node are combined into a ForwardingDomain)
    - Be enriched with associations (e.g., some or all of the one-way navigations are converted to two-way navigations)
- The client maintains (stores in some form of repository) an ongoing live view of the state of the instances of elements in the controlled system so as to populate each of its representational forms
    - A mechanism is available that enables the on-going reporting, from the provider to the client, of change in information known to the provider.
    - Note: A view that is constructed from the currently known state will necessarily be plesiochronous[16] with respect to the actual network state because of differential network and processing delays. After some period, the temporal inaccuracies can mainly be corrected in a view of a particular past time such that the state that was present at some appropriately past time is determinable. This is consistent with the concept of "eventual consistency" (see 3.17 Eventual Consistency and Fidelity).
- When connected for the first time, the client must gain knowledge of current state prior to receiving information on change (changes alone are insufficient to provide a clear view of the system state especially recognizing that most states change very rarely – waiting for a change to determine current state is not viable).
    - On connection to the provider, the client gains alignment with the current state and then maintains alignment as the state changes
    - Through the on-going process the client populates its repository as appropriate and deals with the challenges of asynchronous receipt (e.g., the referencing entity arrives before the referenced entity)

Consequently, the provider aims to optimize the process of maintaining alignment for the client.

Note that TAPI Streaming is not currently intended to directly support:

- A client requiring access to the provider database supporting random queries
- A GUI

See section 7.10 Use cases beyond current release on page 122.

[14] See 3.9.2 Future combination considerations (by example) on page 21
[15] It is assumed here that this would be performed through control of the scope of the context
[16] A plesiochronous system is one where different parts of the system are almost, but not quite, perfectly synchronized.

3 Summary of key considerations

3.1 Overview

This section examines the TAPI streaming capability in detail. Examples of UML and YANG are provided as appropriate.

The characteristics of streaming are described in general and are illustrated using example focusing on alarm reporting.

3.2 RESTCONF notification mechanism (described in [LF TR-547])

RESTCONF specifies a traditional form of notification where the assumption is that a relatively short[17] queue of notifications will be available on the provider to allow the client to receive recent changes[18] and where alignment with current state (in the case of alarms, alignment with current active alarms) will be achieved by retrieving appropriate information from the TAPI context, e.g., using GET operations via RESTCONF.

3.3 TAPI Streaming

An Event source/server streaming mechanism is made available as an alternative to traditional notifications.

The streaming capability is distinct from TAPI Notification and is designed to better deal with scale and to provide an improved operational approach.

The method defined offers a “compacted log” (see 3.8.1.1 Compacted log on page 27) capability that allows the client to gain and maintain alignment with current state from the stream alone (with no need to get current state). The client can achieve eventual consistency (see 3.17 Eventual Consistency and Fidelity on page 48) by simply connecting to the relevant streams. The client will receive an ongoing view of change, assuming that the client is keeping up reasonably with the stream. The stream is designed to allow for some client delay with no loss of information fidelity.

When the client has a significant delay, there will be a loss of fidelity, due to compaction (see 3.8.1 Log storage strategy on page 27), but no impact on eventual consistency. If the client has a very large delay[19], then a resync will be initiated by the provider. Resynchronization will be achieved simply by the client reconnecting to the stream from offset zero. This will again allow the client to achieve eventual consistency.

The streaming capability provides a reliable pipeline for reporting of change[20]. This improves the information flow integrity and reduces the need for recovery and resynchronization.

[17] Sufficient to buffer against variable client performance.
[18] It will also allow the client to request changes starting from a specified sequence number.
[19] The provider controls feeding the stream per client based upon backpressure from the client and is aware where it is reading from in the log, if the log record read is older than the tombstone-retention (see definition/explanation later in this document), then the client will have potentially lost relevant tombstones and hence has possibly lost “eventual consistency”
[20] The capability assumes reliable communications such as TCP.

3.4 Stream content

The streaming approach is generally applicable to all information available over TAPI from the provider to client. Different stream and log strategies will apply to different types of information. This document focusses on the compacted log approach.

The streaming capability also offers an alarm structure definition. In TAPI 2.4 the alarm structure used by tapi-streaming and that used by tapi-notification have been aligned (as defined in tapi-fm.yang). This structure is described in section 4 Using the compacted log approach for alarm reporting on page 54.

3.5 TAPI Application in detail

A Management-Control system, such as an Orchestrator, high level controller, OSS etc., has the role of configuring and adjusting the controlled system (network) to achieve intended capability (intent, service etc.). By monitoring and processing information (e.g., alarms) from the controlled system, the overall assembly of Management-Control systems can determine actions necessary to enable ongoing support of intent/service. The Management-Control systems can also identify repair action prioritization (via analysis of problems).

Management-Control system components use TAPI to acquire, from the subordinate systems, information from a fragment of the overall network, e.g., the devices controlled by a controller, where that information is presented in terms of TAPI entities[21] within a TAPI Context.

The client maintains history and live views of the state of the things in the network so as to do the necessary analysis, hence that Management-Control system uses a mechanism providing autonomous updates and need NOT query the provider for states.

The overall solution is expected to have the following characteristics for the provider:

- Few (~2) direct clients[22]
    - For example, a single OSS/orchestrator with several separate internal systems (fault, provisioning, equipment inventory) and potentially some form of resilience
    - It is expected that TAPI is used at a point low in the Management-Control hierarchy close to the controlled devices (see Figure 1):
        - Interfaces above the OSS are unlikely to use TAPI such as:
            - Interface to customer management solutions
            - Interfaces direct to end user
        - At this point it is likely that the solution is somewhat traditional in nature with an OSS, or Orchestrator, or potentially an orchestrator and OSS operating in conjunction
        - It is assumed that the OSS/Orchestrator will be composed of functionally focused components (e.g., fault analysis, path computation) but that it will provide a relatively unified interface to the subordinate systems
        - The OSS/Orchestrator may be operating some form of resilience
    - It is allowed for a provider to divide up the information based upon entity type
        - This allows simple separation of topology from equipment from alarms
        - This simple split probably matches the normal gross partition of roles in an OSS/Orchestrator

[21] A TAPI entity in this context is a Global Class, i.e., an instance identified by a UUID and hence the corresponding Yang sub-tree.

[22] "Few (~2) direct clients" is intended as order of magnitude, i.e., are not expected several tens of clients. In a future version there will be a broader consideration regarding client multiplicity for other applications

  - It is assumed that there will be one or two clients for each stream type (perhaps up to 4 if, for example, there is both a resilient orchestrator and a resilient OSS which are both providing some alarm capability)
- Long-lived clients:
  - Clients remain “connected” for a very long time and if the connection is dropped the same client will usually try to reconnect
  - Because of the point of use of TAPI in the Management-Control hierarchy and the role and purpose of the clients (OSS/Orchestrator), it is expected that the clients will be permanently connected.
- Provider maintains alignment with underlying system
  - The TAPI realization assumes a reliable input that ensures eventual consistency with current network state
  - As the client is an OSS/Orchestrator, it will have a repository.
    - The normal mode or operation is to align the repository with the view provided by the underlying system and to build a broader view of the network by integrating the views from many underlying systems.
    - For the OSS/Orchestrator to perform its expected functions it is necessary for it to maintain alignment.

The primary focus for Streaming is simple and efficient ongoing alignment of a client with a view presented by a provider.

3.6 Summary of Streaming Characteristics

The key characteristics of the TAPI Streaming solution:

1. Ensures “eventual consistency” of the client with the view presented by the provider
   - Essentially, if the controlled system stops changing, once the whole stream has been received and processed in order by the client, the client view will be aligned with the corresponding controlled system state (assuming communication with all components in the controlled system is operating correctly etc.)
2. Is built on a provider log of records detailing change in the controlled system
   - The log is designed to enable “eventual consistency”
3. Guarantees delivery of each log record “at least once”
   - Clearly, this guarantee applies within a particular operational envelope as defined in this document
   - The provider may deliver some information more than once, but this will be in a pattern that ensures “eventual consistency”
4. Is highly scalable and available
   - Boundless scale (with corresponding system resources)
5. Is highly reliable (network fault tolerant)
   - Provides an inherent high availability solution (assuming necessary implementation can be realized on a resilient server)
   - Is tolerant to network communications disruption allowing the client to resume from where it last successfully received a record.
   - Can feed multiple instances of client
6. Has low latency and high throughput on big data scale
   - Assuming the appropriate implementation technology
7. Offers flexibility in the division of information across streams
   - There can be multiple streams offered by a provider to a client where each stream differs from the others in terms of information content and/or protocol

- In the case where there are multiple streams offered, the client may have to connect to several streams to get all the information it needs

8. Allows the client to re-consume records from a given stream any time.
9. Supports back-pressure[23] from client to enable a reactive producer.

3.7 Supported and available streams

The interface can offer many streams for a context. The client can determine, using calls on the provider, both the types of stream supported, and the available streams that are active for connection and streaming. A variety of connection protocol, content, record strategy and storage strategy combinations might be offered. Clearly, some combinations will not be useful.

The next sections provide some fragments of YANG. The formal YANG deliverable shall be used as the definitive source of information on the encoding. This section indicates where properties are mandatory or optional for a Compacted Log based stream. Other streaming strategies will be documented in future releases.

3.7.1 Supported stream type

This structure allows the provider to report the streams that it can support, regardless of whether they are active or not.

Note that “record-retention” and “segment-size” (see section 8.1 Appendix – Considering compacted logs on page 125) are both string fields. They both have potential for complex structuring and may require future formalization. For example, “record-retention” is either time or capacity and also has a key word when the retention is “FOREVER”. In a future release this may become a complex structure.

[23] Applying some control to reduce the flow from the provider such that the client does not lose information.

```
grouping supported-stream-type {
      leaf stream-type-name {
          type string;
          config false;
          description "Name of the stream type.
              CONDITION: Mandatory where assisted human interpretation is required.";
      }
      leaf record-retention {
          type string;
          default "FOREVER";
          config false;
          description "Time in minutes.
              Statement of retention time and/or retention capacity in bytes.
              Key word 'FOREVER' means that records will never be removed from the log.
              May be overridden for particular cases of specific LogStorageStrategy (via augment).
              Applies to all record types in the stream unless overridden by another parameter (such as tombstone retention for a compacted log).
              CONDITION: Mandatory where not default.";
      }
      leaf segment-size {
          type string;
          config false;
          description "Size of sub-structuring of the log.
              CONDITION: Mandatory where log is segmented and segment size is considered relevant for client application usage.";
      }
      leaf-list stream-type-content {
          type tapi-common:object-type;
          config false;
          description "Identifies the classes that are supported through the stream.
              The list may be a subset of the classes within the context.
              CONDITION: Mandatory if the stream propagates TAPI entities. If not present a separate augment MUST explain stream content.";
      }
      leaf log-storage-strategy {
          type log-storage-strategy;
          default "LOG_STORAGE_STRATEGY_COMPACTED";
          config false;
          description "Indicates the storage characteristics of the log supporting the stream.
              CONDITION: Mandatory where not default.";
      }
      leaf log-record-strategy {
          type log-record-strategy;
          default "LOG_RECORD_STRATEGY_WHOLE_ENTITY";
          config false;
          description "Indicates the type of content of each log record.
              CONDITION: Mandatory where not default.";
      }
      leaf record-trigger {
          type record-trigger;
          default "RECORD_TRIGGER_ON_CHANGE";
          config false;
          description "Defines the trigger to log a record.
              CONDITION: Mandatory where not default.";
      }
      uses tapi-common:global-class;
      description "Definition of a supported stream type.";
  }
```

Figure 2 YANG: supported-stream-type

For this structure, a solution shall provide the following support:

- Mandatory:
  - (inherited) uuid
- Conditional (as per description above):
  - stream-type-name
  - record-retention

    - stream-type-content
    - segment-size
    - log-storage-strategy
    - log-record-strategy
    - record-trigger
- Optional
    - (inherited) name

There are several data types used in the supported-stream-type structure, these are set out below.

```
  typedef object-type {
    type identityref {
      base OBJECT_TYPE;
    }
    description "The list of TAPI Global Object Class types on which Notification signals can be raised.
      This extensible enumeration can be augmented with specific object types/classes in the other modules.";
  }

And in tapi-common.yang
  identity OBJECT_TYPE {
    description "none";
  }
  identity OBJECT_TYPE_SERVICE_INTERFACE_POINT {
    base OBJECT_TYPE;
    description "The ServiceInterfacePoint (SIP) class.";
  }
  identity OBJECT_TYPE_TAPI_CONTEXT {
    base OBJECT_TYPE;
    description "The TapiContext class.";
  }
  identity OBJECT_TYPE_PROFILE {
    base OBJECT_TYPE;
    description "none";

And in tapi-connectivity.yang
  identity CONNECTIVITY_OBJECT_TYPE {
        base tapi-common:OBJECT_TYPE;
    description "none";
  }
  identity CONNECTIVITY_OBJECT_TYPE_CONNECTIVITY_SERVICE {
    base CONNECTIVITY_OBJECT_TYPE;
    description "The ConnectivityService class.";
  }
  identity CONNECTIVITY_OBJECT_TYPE_CONNECTIVITY_SERVICE_END_POINT {
    base CONNECTIVITY_OBJECT_TYPE;
    description "The ConnectivityServiceEndPoint (CSEP) class.";
  }
  identity CONNECTIVITY_OBJECT_TYPE_CONNECTION {
    base CONNECTIVITY_OBJECT_TYPE;
    description "The Connection class.";
  }
  identity CONNECTIVITY_OBJECT_TYPE_CONNECTION_END_POINT {
    base CONNECTIVITY_OBJECT_TYPE;
    description "The ConnectionEndPoint (CEP) class.";
  }
  identity CONNECTIVITY_OBJECT_TYPE_SWITCH_CONTROL {
    base CONNECTIVITY_OBJECT_TYPE;
    description "The SwitchControl class.";
  }
… etc.

Each model, tapi-topology.yang etc., provides identities for each of the entities supported by that model.
```

Figure 3 YANG: object-type (showing some of the identities)

```
typedef log-storage-strategy {
    type identityref {
        base LOG_STORAGE_STRATEGY;
    }
    description "Defines the storage (record retention) approach.";
}

identity LOG_STORAGE_STRATEGY {
    description "none";
}
identity LOG_STORAGE_STRATEGY_COMPACTED {
    base LOG_STORAGE_STRATEGY;
    description "The log uses some mechanism to remove noisy detail whilst enabling the client to achieve eventual consistency (alignment)
    with current state.";
}
identity LOG_STORAGE_STRATEGY_TRUNCATED {
    base LOG_STORAGE_STRATEGY;
    description "The log only maintains recent records and disposes of old records.
        This log does not alone enable the client to achieve alignment with current state.";
}
identity LOG_STORAGE_STRATEGY_FULL_HISTORY {
    base LOG_STORAGE_STRATEGY;
    description "Maintains a history from system initiation with no missing records.
        Provides initial state at the beginning of the history";
}
identity LOG_STORAGE_STRATEGY_FULL_HISTORY_WITH_PERIODIC_BASELINE {
    base LOG_STORAGE_STRATEGY;
    description "Provides a history with initial state and periodic/occasional statements of current state at a particular point in time.";
}
```

Figure 4 YANG: log-storage-strategy

```
typedef log-record-strategy {
    type identityref {
        base LOG_RECORD_STRATEGY;
    }
    description "Defines the different approaches for logging information about an event covering the log trigger and the log content.";
}

identity LOG_RECORD_STRATEGY {
    description "none";
}
identity LOG_RECORD_STRATEGY_CHANGE_ONLY {
    base LOG_RECORD_STRATEGY;
    description "Each record only provides a view of the changes that have occurred (on a per entity change basis).
        E.g., the log only includes the attribute that has changed and not other attributes that have not changed.";
}
identity LOG_RECORD_STRATEGY_WHOLE_ENTITY {
    base LOG_RECORD_STRATEGY;
    description "A record provides a snapshot of a whole entity.
        The record includes all properties and values whether they have changed or not.";
}

And two DEPRECATED strategies
identity LOG_RECORD_STRATEGY_WHOLE_ENTITY_ON_CHANGE {
    base LOG_RECORD_STRATEGY;
    description "DEPRECATED Replaced by WHOLE_ENTITY with record trigger ON_CHANGE.
        A record provides a snapshot of a whole entity and a snapshot is taken on each change.
        The record includes all properties and values whether they have changed or not.";
}
identity LOG_RECORD_STRATEGY_WHOLE_ENTITY_PERIODIC {
    base LOG_RECORD_STRATEGY;
    description "DEPRECATED Replaced by WHOLE_ENTITY with record trigger PERIODIC.
        A snapshot of an entity is recorded periodically regardless of whether there has been change or not.";
}
```

Figure 5 YANG: log-record-strategy

```
typedef record-trigger {
    type identityref {
        base RECORD_TRIGGER;
    }
    description "The trigger for logging a record.";
}

identity RECORD_TRIGGER {
    description "none";
}
identity RECORD_TRIGGER_ON_CHANGE {
    base RECORD_TRIGGER;
    description "A record is logged each time the value of the item to be recorded changes.";
}
identity RECORD_TRIGGER_PERIODIC {
    base RECORD_TRIGGER;
    description "A record is logged for the item on a periodic basis (independent of whether the values have changed or
    not).";
}
identity RECORD_TRIGGER_DEFINED_TRIGGER {
    base RECORD_TRIGGER;
    description "The trigger will follow a strategy that is complex and specified via additional detail.";
}
```

Figure 6 YANG: record-trigger

Considering the supported-stream-type YANG the provider can indicate the entity types, the storage strategy, record strategy and record trigger supported by a stream type.

The segment-size is a choice made by the provider depending upon system engineering considerations. Larger segments may interfere with compaction delay where the stream is handling entities with a low rate of change whereas smaller segments will require additional segment creation activity.

Information may be divided into separate streams. There is no restriction, other than it being at an entity type granularity, on choice of division of the information into streams. A provider could choose to have a stream per class or to have streams that aggregate classes together that have similar lifecycles etc. It should be noted that for ALL instances in the context of any object-class-identifier listed in record-content will be streamed, i.e., there is (intentionally) no client control of filtering[24].

The supported-stream-type can also be augmented with:

- compacted-log-details which provides additional parameters for compacted log applications. This augmentation shall be applied when the solution is running compacted logs.
- connection-protocol-details which provides a list of allowed-connection-protocols and an encoding-format. These are formalized enumerations represented as identities as set out below.
- information-record-strategy. This structure is experimental and is not used at this stage.

For a compacted log, record-retention time should be “FOREVER”. For a non-compacted log, record-retention is a choice made by the provider depending upon system engineering. Clearly, the longer the retention for a non-compacted log, the larger the log.

The description in this document focusses on Compacted logs (some other log strategies are also considered). Compacted logs are explained at various points in this document. The key is as noted in the YANG description below, which is essentially that the log holds only the latest record about each entity. Once an entity is deleted a TOMBSTONE log record (see section 8.1.6 The TOMBSTONE on page 126) for the entity is appended to the log (which then becomes the latest record for the thing) [25]. The TOMBSTONE records are only held for a relatively short period[26] of time (the tombstone-retention).

[24] It is assumed that the Context has been designed to include all the entities that the client is interested in and hence the process of Context formation at the provider does all the necessary filtering to ensure that the client gets what it needs.
[25] A DELETE record will also be appended just before the TOMBSTONE record (hence the TOMBSTONE record is the latest record).
[26] The time is determined by system engineering considerations. For example, in a specific solution, comms failures between the client and the provider of greater than 4 hours may be considered as extremely unlikely, and general system recovery from a 4 hour log may take 2 hours. In this case a tombstone-retention of slightly more than 6 hours is suitable. This is a relatively short period of time.

```
grouping compacted-log-details {
    leaf tombstone-retention {
        type string;
        default "FOREVER";
        config false;
        description "Time in minutes.
            The time period for which a Tombstone record will be held in the log from when it was logged.
            This provides an adjustment to the essential Compaction strategy such that after the tombstoneRetention period there will be no
            records about a particular thing that existed but no longer exists.
            Tombstone retention overrides recordRetention for Tombstones.
            Key word 'FOREVER' means that Tombstone records will never be removed from the log.
            Can be adjusted by an administrator (via a separate view) through the life of the stream.
            CONDITION: Mandatory where not default.";
    }
    leaf compaction-delay {
        type string;
        default "0";
        config false;
        description "Time in minutes.
            The delay between logging the record and making the record available for compaction.
            This provides an adjustment to the essential Compaction strategy such that there may be several distinct records for the same thing
            in the where those records are not older than the Compaction Delay.
            Can be adjusted by an administrator (via a separate view) through the life of the stream.
            CONDITION: Mandatory where not default.";
    }
    leaf max-allowed-segment-roll-delay {
        type string;
        default "NOT_APPLICABLE";
        config false;
        description "The maximum time the log head segment can be allowed to be not made available for compaction.
            Applicable where the log is segmented, and the head segment is not available for compaction.
            The setting influences the compaction behavior and may cause a delay before compaction that is much greater than the defined
            compaction delay.
            Time in seconds.
            Can be 'FOREVER'.
            Can be 'NOT_APPLICABLE' (which indicates that compaction can act on the head segment).
            CONDITION: Mandatory if log is segmented in such a way that the active head segment is not available for compaction.";
    }
    leaf max-compaction-lag {
        type string;
        default "NOT_APPLICABLE";
        config false;
        description "The maximum delay, in seconds, beyond the defined compaction delay for compaction processing to take place.
            May be 'NOT_APPLICABLE' if compaction is essentially immediate (i.e., there is negligible delay).
            CONDITION: Mandatory where not default.";
    }
    description "Details relevant for a CompactedLog.
        The essential Compacted Log strategy is to remove historic records about a particular thing such that only the latest record about each
        thing exists in the log.
        The essential strategy is refined by the parameters of this structure.";
}
```

Figure 7 YANG: compacted-log-details

For this structure, a solution shall provide the following support:

- Conditional (as per description above):
    - tombstone-retention
    - compaction-delay
    - max-allowed-segment-roll-delay
    - max-compaction-lag

For the compacted log, the supported-stream-type information is augmented with compacted-log-details that includes properties related to compacted log behavior such as tombstone-retention[27] (essentially retention of records about deleted events – see 3.8.1 Log storage strategy on page 27) and compaction-delay settings. All properties are set by the provider depending upon system engineering. The properties provide the client with information on log behavior so that it can assess whether it is in the compaction zone etc.

```
grouping connection-protocol-details {
    leaf-list allowed-connection-protocols {
        type connection-protocol;
        default "CONNECTION_PROTOCOL_WEBSOCKETS";
        config false;
        description "Name of the allowed protocol(s).
            Where there is a list:
            - all protocols must use the same encoding format
            - there will be one or more available streams per connection protocol
            CONDITION: Mandatory where not default.";
    }
    leaf encoding-format {
        type encoding-format;
        default "ENCODING_FORMAT_JSON";
        config false;
        description "The encoding format of the streamed records.
            CONDITION: Mandatory where not default.";
    }
    description "Details of the connection protocols available for the specific stream.";
```

Figure 8 YANG: connection-protocol-details

For this structure, a solution shall provide the following support:

- Conditional (as per description above):
    - allowed-connection-protocols
    - encoding-format

```
typedef connection-protocol {
    type identityref {
        base CONNECTION_PROTOCOL;
    }
    description "The connection protocols.";
}
…
identity CONNECTION_PROTOCOL {
    description "none";
}
identity CONNECTION_PROTOCOL_WEBSOCKETS {
    base CONNECTION_PROTOCOL;
    description "WebSockets as defined at https://datatracker.ietf.org/doc/html/rfc6455.";
}
identity CONNECTION_PROTOCOL_SSE {
    base CONNECTION_PROTOCOL;
    description "Server Sent Events as defined at https://www.w3.org/TR/2015/REC-eventsource-20150203/.";
}
identity CONNECTION_PROTOCOL_GNMI {
    base CONNECTION_PROTOCOL;
    description "Google network Management Interface as specified at https://github.com/openconfig/reference/tree/master/rpc/gnmi.";
}
```

Figure 9 YANG: connection-protocol

[27] A TOMBSTONE record provides sufficient to express the delete of an entity, tombstone-retention is the time for which a TOMBSTONE record will be held in the log.

```
typedef encoding-format {
    type identityref {
        base ENCODING_FORMAT;
    }
    description "The list of possible encoding formats.";
}
…
identity ENCODING_FORMAT {
    description "none";
}
identity ENCODING_FORMAT_JSON {
    base ENCODING_FORMAT;
    description "JavaScript Object Notation as defined at https://www.json.org/json-en.html.";
}
identity ENCODING_FORMAT_PROTOBUF {
    base ENCODING_FORMAT;
    description "Protocol Buffers as defined at github.com/protocolbuffers/protobuf.";
}
identity ENCODING_FORMAT_XML {
    base ENCODING_FORMAT;
    description "eXtensible Markup Language as defined at https://www.w3.org/standards/xml/.";
}
```

Figure 10 YANG: encoding-formats

```
grouping information-record-strategy {
    leaf record-suppression {
        type record-suppression;
        default "RECORD_SUPPRESSION_NO_SUPPRESSION";
        config false;
        description "Indicates whether records are suppressed and if so, what the suppression strategy is.
            CONDITION: Mandatory where not default.";
    }
    leaf value-expectation {
        type value-expectation;
        default "VALUE_EXPECTATION_NO_EXPECTATION";
        config false;
        description "Where there is record suppression this indicates what the relevant expected value is.
            If the value is as expected the record will be suppressed.
            CONDITION: Mandatory where not default.";
    }
    leaf allowed-dither-from-value-expectation {
        type value-expectation-dither;
        default "VALUE_EXPECTATION_DITHER_NO_DITHER";
        config false;
        description "Defines the dither in an expected value that is allowed for the value to still be considered as expected.
            CONDITION: Mandatory where not default.";
    }
    description "Properties relevant for a stream that may convey records of INFORMATION record type.";
}
```

Figure 11 YANG: information-record-strategy

This information-record-strategy structure is experimental, provided in preparation for future cases on information record streaming. This structure does not need to be supported at this stage.

For this structure, a solution shall provide the following support:

- Conditional (as per description above):
    - record-suppression

- value-expectation
- allowed-dither-from-value-expectation

3.7.1.1 Supported stream type combinations

The supported-stream-type structure allows a stream to be fully defined and can be used to provide a full unique combination of data set (stream-type-content), protocol and encoding. It is expected at this stage that each structure defines one unique stream fully specifying all individual property.

It is recommended that the allowed connection protocol field has only one entry for each supported-stream-type. The stream type will then specify a single explicit encoding format and connection protocol combination for the connection protocol details.

The provider can advertise as many supported-stream-types as is necessary to represent the combination of data sets, protocol details and encoding formats that it offers. The supported-stream-type list deals with the various combinations. If a data set is provided via various alternative formats, then there will be multiple supported-stream-types, each one defining a single specific format.

An available-stream (see below) can then reference a supported-stream-type with no ambiguity and the available-stream need not define the connection-protocol.

For example, if the provider supports a specific set of data over a stream that is PROTOBUF over gNMI it would specific a supported-stream-type with only that combination (i.e., a single allowed-connection-protocol of gNMI and a single encoding-format of PROTOBUF). If it supports the same data over a gNMI stream that is not PROTOBUF encoded but uses JSON, then this is specified via a separate supported-stream-type.

3.7.2 Available Streams

This structure allows the provider to report the streams that are currently available for connection.

```
grouping available-stream {
    leaf-list connection-address {
        type string;
        config false;
        description "Provides the address for the connection.
            The format of the address and attachment mechanism will depend on the connection protocol defined in another attribute of this
            class.
            There may be a sequence of operations required, in which case, these should be listed as separate strings.
            A string may include wildcard sub-statements.
            A single string may list alternatives separated by an appropriate delimiter.";
    }
    leaf stream-state {
        type stream-state;
        default "STREAM_STATE_ACTIVE";
        config false;
        description "The state of the stream.
            CONDITION: Mandatory where stream state is not ALWAYS default.";
    }
    container supported-stream-type {
        uses supported-stream-type-ref;
        config false;
        description "Identifies the type of stream that is available for connection.";
    }
    leaf stream-id {
        type string;
        config false;
        description "The id of the stream (alternative to the uuid).
            CONDITION: Mandatory where an alternative id to the uuid is available.";
    }
    leaf connection-protocol {
        type connection-protocol;
        default "CONNECTION_PROTOCOL_WEBSOCKETS";
        config false;
        description "Names the connection protocol for this particular available stream.
            The connection protocol is chosen from the list of connection protocols identified in the referenced SupportedStreamType.
            CONDITION: Mandatory where not default and multiple options offered in the supported stream type.";
    }
    uses tapi-common:global-class;
    description "Details of a stream that can be connected to by a client application.";
}
```

Figure 12 YANG: available-stream

For this structure, a solution shall provide the following support:

- Mandatory:
    - (inherited) uuid
    - connection-address: the location of the stream
        - The address structure and connection method will depend upon the connection-protocol
    - supported-stream-type: references the description of a stream supported by the provider
- Conditional (as per description above):
    - stream-state: indicates whether the stream will deliver records to the client or not
    - stream-id: id of the stream
    - connection-protocol: the protocol for this particular stream instance chosen from the list of available protocols for the stream type. This enables interpretation of the connection-address.
- Optional
    - (inherited) name

```
  typedef stream-state {
    type identityref {
      base STREAM_STATE;
    }
    description "The state of the available stream.";
  }
…
  identity STREAM_STATE {
    description "none";
  }
  identity STREAM_STATE_ALIGNING {
    base STREAM_STATE;
    description "The log that underpins the stream is aligning with other backend services and hence may not be providing full service.
      If events are provided, they will be completely valid.";
  }
  identity STREAM_STATE_ACTIVE {
    base STREAM_STATE;
    description "The stream is operating such that if a client connects records will be provided as per back pressure etc.";
  }
  identity STREAM_STATE_PAUSED {
    base STREAM_STATE;
    description "Although the stream is available it has been paused by the administrator such that the records are being appended to the log
    but a new client will not receive any events whilst the stream is paused.";
  }
  identity STREAM_STATE_TERMINATED {
    base STREAM_STATE;
    description "The stream is essentially no longer available. It will be removed from the AvailableStreams list shortly.";
  }
```

Figure 13 YANG: stream-state

3.8 Streaming approach and log strategy

The streaming solution assumes that the provider is delivering information in sequence from a log. In TAPI 2.4 a log approach oriented towards maintaining alignment is provided. The stream mechanism defined allows for different log strategies[28] along with corresponding augmentation.

3.8.1 Log storage strategy

There are four log-storage-strategy provided (LOG_STORAGE_STRATEGY_): COMPACTED, TRUNCATED, FULL-HISTORY and FULL_HISTORY_WITH_PERIODIC_BASELINE, these are described in the subsections below.

3.8.1.1 Compacted log

Where the solution is to report TAPI entities (such as NEPs, CEPs) and/or report alarms/TCAs an implementation conforming to this specification MUST support the COMPACTED log-storage-strategy (the characteristics of this mechanism are described in 8.1.1 Essential characteristics of a compacted log on page 125). This log-storage-strategy enables the client to achieve and maintain eventual consistency without the need to get current state.

[28] These are described in the Yang/UML.

Not all change statements (LOG_RECORD_STRATEGY_WHOLE_ENTITY, RECORD_TRIGGER_ON_CHANGE statements (see 3.8.2 Log record strategy on page 29)) related to an entity are retained. Older changes are intentionally pruned out of the log in favor of newer changes.

- The log will have the latest record related to each entity that exists in the controlled system.
  - This enables achievement of eventual consistency.
- Some additional recent changes will also be present in the log as compaction is intentionally delayed.
  - This enables a delayed client to catch back up with no loss of fidelity.
- Records that are not the latest for an entity and that are older than the compaction delay time (i.e., the "Cleaner Point" as shown in Figure 34 Kafka compaction on page 125) will be removed.
  - This allows an overloaded client to maintain a view of non-fleeting changes whilst suffering an acceptable loss of fidelity where there is high intermittency.
  - This allows a new client to align without having to receive large volumes of uninteresting history.

When an entity is deleted, a DELETE record for that entity is appended to the log followed immediately by a TOMBSTONE record for that entity[29, 30]. The TOMBSTONE records are held for tombstone-retention:

- Compaction, after the compaction delay, will remove all but that TOMBSTONE record[31].
- TOMBSTONE records are removed once they have persisted for the tombstone-retention period (i.e., the "Deletion Retention Point" as shown in Figure 34 Kafka compaction on page 125)
  - Note that without this special tombstone-retention behavior, the log growth would be unbounded.

3.8.1.2 Truncated log

Where the solution is to report measurement data for performance monitoring, an implementation conforming to this specification MUST support the TRUNCATED log-storage-strategy along strategies for other aspects of the log defined in section 6 Streaming of measurement data for performance monitoring on page 75.

The log holds records about all recent changes relevant to the target content for the log as defined by record-content in the supported-stream-type definition. Truncation will occur as a result of volume of records, age of records or some other criteria. Hence, the log content is limited either by log size or by content age.

- For a size bounded log, once the log has reached its content limit, as each new record arrives, space is cleared, to allow the new record to be added, by deleting the oldest records.
- For an age bounded log, records older than the age limit are deleted.

This can be considered as a traditional notification queue where recent records have been retained. This log-storage-strategy does not allow the client to achieve eventual consistency without getting the current state.

See 3.9.2 Future combination considerations (by example) on page 33 for potential uses. Some uses may require augmentation of the supported-stream-type structure to allow full definition. Possible augments will be defined in a future version.

[29] The TOMBSTONE record is extremely small minimising the log space used indicating that an entity no longer exists and hence allowing for a relatively long buffer of delete indications supporting maintenance of alignment from the compacted section of the log.
[30] The delete record can carry information about the delete reason that may be of interest to a client that is maintaining alignment and is not overloaded. When in the compaction zone, the client is attempting to regain alignment and this detail is superfluous.
[31] The delete record will also be removed through the compaction process.

3.8.1.3 Full history log

The log holds all records about changes relevant to the target content for the log as defined by record-content in the supported-stream-type definition since the context was created. Unlike the compacted and truncated cases, this is essentially a boundless log holding the entire history.

See 3.9.2 Future combination considerations (by example) on page 33 for potential uses. Some uses may require augmentation of the supported-stream-type structure to allow full definition. Possible augments will be defined in a future version.

3.8.1.4 Full history with periodic baseline log

The log holds all records about changes relevant to the target content for the log as defined by record-content in the supported-stream-type definition since the context was created and also provides periodic baselines[32] within the stream that allow the client to interpret recent changes without needing to review the entire history. The form of the baseline records and settings for the log have not been defined in TAPI 2.4. The application of this type of log is for further study.

See 3.9.2 Future combination considerations (by example) on page 33 for potential uses. Some uses may require augmentation of the supported-stream-type structure to allow full definition. Possible augments will be defined in a future version.

3.8.2 Log record strategy and record trigger

There are two log-record-strategy options: LOG_RECORD_STRATEGY_CHANGE_ONLY and LOG_RECORD_STRATEGY_WHOLE_ENTITY and two specific record-trigger options: RECORD_TRIGGER_ON_CHANGE and RECORD_TRIGGER_PERIODIC[33], these are described in the subsections below. The values will be abbreviated to CHANGE_ONLY, WHOLE_ENTITY, ON_CHANGE and PERIODIC.

3.8.2.1 Whole entity on change

This approach uses log-record-strategy WHOLE_ENTITY with a record-trigger of ON_CHANGE. The combination of this log-record-strategy with log-storage-strategy of COMPACTED is described in detail in this document to enable the client to achieve eventual consistency (the provider may alternatively use log-record-strategy CHANGE_ONLY described in section 3.8.2.2 Change only on page 30 to enable the client to achieve eventual consistency).

In this case, each record in the log holds a full copy of the entity. A full copy of the entity is stored when the entity is created and when any state changes, i.e., the solution logs (stores) a full representation of the entity with current values after each change, not just the changed property. Hence, the whole entity that has the changed property is streamed. The specific changes can be identified by comparing the current record with the previous record. Note that the entity could have more than one changed property.

The model is such that:

[32] A baseline is a statement of absolute state (in this case of all resources relevant to the stream). Updates alone cannot be meaningfully interpreted as they are updates against some state. A full history will have all state changes, but starting from the first moment in the history to determine the current state, running through all changes would be extremely expensive. Hence providing a baseline from time to time can speed up alignment when a system connects for the first time.

[33] Note that the RECORD_TRIGGER_DEFINED_TRIGGER is not covered here as it is for future use and has not been fully developed.

- Each property that changes regularly is isolated in its own dedicated small class[34]
    - E.g., The alarm (detector) is considered as a class. Alarms are isolated from configuration data for the related entity and from each other.
- Large data structures that are invariant or change rarely can be grouped in composite classes.
    - E.g., The CEP where Configuration data (both intent and actual) is collected into a single class. The data in an instance changes rarely.
- For optimum support of change of properties, the small property class reference the configuration items that they relate to and NOT the reverse (i.e., be a decoration, e.g., an alarm references the CEP or MEP)

This record strategy can be used with any of the storage strategies.

3.8.2.2 Change only

A record that is CHANGE_ONLY (where the record-trigger is ON_CHANGE) provides a skeleton of the instance of the class (identifiers and basic structure) with only the changed property (or properties) present.

It is anticipated that this record strategy can be used with any of the storage strategies, but it requires specific behaviour for the COMPACTED log storage strategy. This behaviour is described in section 5 Reporting Change Only on page 70.

Mechanism and approached are described to deal with:

- compaction of records that are a combination of whole entity statements and change statements
- removal of properties in a tree
- repositioning of elements in an ordered list

Streaming using a log-record-strategy CHANGE_ONLY with record-trigger ON_CHANGE and log-storage-strategy COMPACTED enables the client to gain and maintain eventual consistency efficiently.

See 3.9.2 Future combination considerations (by example) on page 33 for potential uses. Some uses may require augmentation of the supported-stream-type structure to allow full definition. Possible augments will be defined in a future version.

3.8.2.3 Whole entity periodic

Using this strategy (defined by a combination of WHOLE_ENTITY and PERIODIC), the information is streamed on some periodic basis even if it has not changed. This approach has not been fully developed. See 3.9.2 Future combination considerations (by example) on page 33 for potential uses. Some uses may require augmentation of the supported-stream-type structure to allow full definition. Possible augments will be defined in a future version. This log-record-strategy could be used to enable the client to achieve eventual consistency.

3.8.2.4 Change-only periodic

Using this strategy (defined by a combination of CHANGE_ONLY and PERIODIC), the information is streamed on some periodic basis where a record is sent detailing only the change and is sent only when there has been a change. This approach has not been fully developed. See 3.9.2 Future combination considerations (by example) on page 33 for potential uses. Some uses may require augmentation of the supported-stream-type structure to allow full definition. Possible augments will be defined in a future version. This log-record-strategy could be used to enable the client to achieve eventual consistency.

[34] Currently Operational State is part of the main entity where there is a large volume of slow changing data. Operational State actually has an alarm like behaviour and should be reported as an alarm.

3.9 Using the stream

Further details are provided in the use cases in section 7 Use Cases on page 88.

3.9.1 Streaming the context

A primary application of streaming is in a solution where a client needs to gain and maintain alignment with a context presented by a provider:

- The provider presents a view in terms of a context and all of its contained instances.
- The client maintains alignment with that view.
- The stream is used for gaining and maintaining alignment with a view, i.e., a TAPI context.

The next subsections consider the client connection to and receiving from a stream. The approach is described for the case where the provider offers COMPACTED, WHOLE_ENTITY and ON_CHANGE (alternatively, instead of WHOLE_ENTITY, CHANGE_ONLY can be used). The description assumes that the client is capable of consuming the stream at a far greater rate than the stream is being filled. The following provides a brief sketch of alignment. The process is discussed in far greater detail later in the document (see section 7 Use Cases on page 88).

3.9.1.1 Effect of streaming approach and compacted log characteristics on alignment

There are two distinct aspects of alignment:

- Absolute State:
  - Eventual Consistency: Ensures that the client view of state of controlled system aligns with the state of the view of the actual controlled system as presented by the provider.
    - If the controlled system stops changing, once the stream (all changes) has been absorbed by the controller, its view of the current state of the system will be aligned with the actual state of the system.
  - Context Detail: The information agreed to be conveyed from the provider to the client.
    - Note that some clients will choose to selectively prune out information that is not relevant to them.
    - Note that in future, information conveyed in a context may be temporarily increased as a result of a recognized short-term need (see spotlighting in section 7.10 Use cases beyond current release on page 122)
- Change of state:
  - Detail will necessarily be lost (loss of fidelity) when there are long communications failures, but the loss will initially be such that less relevant information is lost first.
    - Removal of noise (such as rapid clearing and then re-raising alarms) will be generally beneficial.

3.9.1.2 Preparing to connect

Once the client has identified the available streams to connect to, the client simply acquires the necessary authorization (see 7.11 Message approach (WebSocket example) on page 123 and 7.9 Message Sequence on page 118) and connects.

3.9.1.3 Initial connection

On initial connection, the client provides a null token. This causes the provider to stream from the oldest record. The client can continue to consume records from the stream ongoing.

The initial records received by the client will be for the entities that have not changed for the "longest" time[35].

[35] For example, if the system has been running for three years and a thing was created when the system started. If that thing has never changed since creation, then the record of its creation from three years ago will still be in the log.

3.9.1.4 TOMBSTONE record (delete) retention passed

As the client continues to consume the stream it progresses past the tombstone-retention (essentially delete retention) point, i.e., is receiving records that have a timestamp that is less than the tombstone-retention (delay) from the current time (see 3.8.1 Log storage strategy on page 27 and 8.1.1 Essential characteristics of a compacted log on page 125 for a brief explanation of the log structure), and recent TOMBSTONE records will be received along with newer changes[36].

Compaction will have removed multiple reports about the same entity, but as the client progresses further it is possible that an update is received that overwrites previously received entity state or a TOMBSTONE record is received that deletes an entity that was read earlier. This is where compaction had not yet removed the entity when the stream was started (potentially because the event causing the newer record had not yet occurred).

3.9.1.5 Compaction delay passed

After some time, the client consumes past the compaction delay point (i.e., is receiving records that have a timestamp that is less than the compaction delay from the current time). From this point onwards the client is receiving all recent changes (either using the WHOLE_ENTITY or CHANGE_ONLY approach) and is aligned with network state as it was perceived by the provider at some recent point in time. Whilst receiving records that are newer than the compaction delay point the client will receive all event reports for the context (see 3.8.1 Log storage strategy on page 27 and 8.1.1 Essential characteristics of a compacted log on page 125 for a brief explanation of the log structure).

3.9.1.6 (Eventual) Consistency achieved

If the controlled system stopped changing, then the client would eventually reach the newest record and would be aligned with the provider view of the state of the controlled system[37].

3.9.1.7 Degraded performance

Information fidelity is reduced if the client slips back by more than the compaction delay as compaction will remove some change detail.

3.9.1.8 Need for realignment

If the client processing of a stream is delayed by more than the tombstone-retention[38] such that the backpressure on the stream takes the provider log read point to a record older than the tombstone-retention time, the provider will cause the connection to drop.

When the client detects this, it will reconnect, normally with the token for the most recently processed record from the stream. The provider will recognize this as for a record older than the tombstone-retention and will stream from offset zero (i.e., the oldest record in the log). This will cause the client to enter into full realignment.

The behavior of the provider at this point is equivalent to that when there is an initial connection. The client may use a “mark and sweep” strategy to minimize the disruption to its view of current state.

3.9.1.9 Summary

The above detail can be summarized:

1. The client will connect for the first time and the provider will stream from the oldest record in the log.
2. Where the client loses communications or loses a record for some other reason, it can reconnect to the provider indicating the last record that it successfully received. Missed records are then streamed as appropriate where tombstone-retention has not been exceeded.

[36] TOMBSTONE records (representing deletes) are only retained for a limited time. TOMBSTONE records older than the tombstone-retention are removed from the log.
[37] Because the provider logs the whole entity on each change then the most recent record for an entity, retained after compaction has removed earlier records, will include all of its properties.
[38] The TCP buffering will provide some additional time for a client beyond the tombstone-retention.

3. Where the client has crashed it connects to the stream as if for the first time in (1) above
4. Where tombstone-retention has been exceeded the provider takes the client back to the start of the stream
    - Compaction removes noisy history and allows rapid alignment with current state through a "replay" of the compacted history.

3.9.2 Future combination considerations (by example)

The choice of the protocol for streaming is in principle independent from the stream characteristics, although the protocol chosen must match the stream reliability needs. There are several potential applications beyond those documented this first release.

3.9.2.1 Many clients

Although, at this stage, all TAPI applications appear to have a small number of clients, it is possible that applications will emerge with many clients for a single type of information. stream of information.

For some "many clients" application a combination of TRUNCATED, WHOLE_ENTITY and PERIODIC would appear to be suitable. In this case the TAPI provider may supply the clients directly or via an intermediate solution to deal with load. It is also possible that the TAPI provider might stream full content and then the intermediate solution might support a subscription method.

3.9.2.2 Many views and many clients (with a few clients per view)

In this case, the solution would appear to be one with multiple contexts, one per view. Here any relevant log-record-strategy and log-storage-strategy could be supported. Multiple contexts would benefit from support of further features such as building context (discussed briefly in section 7.10 Use cases beyond current release on page 122).

It is assumed that:

- Each context will appear as a separate independent instance of a TAPI sever.
    - Each context will have its own dedicated streams.
- A context will expose a subset/subgraph of the overall network where that subset/subgraph/ may be a small compared to the overall network.
- The identifiers in each context
    - May be different for the same underlying entity, to improve security of access where views are being offered to distinct and independent clients.
    - May be the same for a related entity across two or more contexts, to allow for partitioned views between related clients. In this case it is expected that the clients will be sharing data. In a basic example, one client may deal with equipment whist another deals with network topology. The model the equipment client supports has references to the model the network client supports.

3.9.2.3 Many short-lived clients

In this case, where each client samples the information from some combination of streams for a very short period. This appears to be some combination of many views and many clients.

3.9.2.4 Live measurements

Where there is a low number of clients, each requiring a reasonably up-to-date view including values related to measurements (e.g., delay, temperature, power) which may:

- tend to dither around a value or tends to increase (or decrease) on an ongoing basis, a combination of COMPACTED, WHOLE_ENTITY and PERIODIC would appear to be suitable.
- tend to change rarely, a combination of COMPACTED, WHOLE_ENTITY and ON_CHANGE would appear to be suitable.

Other strategy combinations will be developed in future to deal with various information-record-strategies supporting suppression of same value and of dither.

3.9.2.5 Threshold Crossing

This depends upon the behavior of the threshold source. For a source that reports threshold crossing:

- at some point within a period of measurement, at the end of the measurement period resets the threshold crossing without a clear and then potentially reports the threshold crossing in the next period etc., the solution would best use log-storge-strategy = TRUNCATED.
- and recovery over a continuous measurement, the solution would best be the same as the alarm solution, i.e., COMPACTED, WHOLE_ENTITY and ON_CHANGE

3.9.2.6 Periodic measurement data

Where a measurement is taken over a period of time and where the measurement is repeated regularly (potentially over adjacent periods of time). An example of this is the 15-minute Performance measurement.

- At the end of the measurement period the result of measurement is streamed
- The natural approach would be to use TRUNCATED, WHOLE_ENTITY and PERIODIC
  - This provides a short history to handle communications problems.

Some periodic measurements have predictable normal characteristics.

- Consider an error measurement on a photonic system.
  - This is normally zero and hence the errored second count in a 15-minute period is normally zero as is the severely errored second count. On that basis an appropriate information-record-strategy (e.g., RECORD_SUPPRESSION_SUPPRESS_EXPECTED and VALUE_EXPECTATION_VALUE_IS_ZERO) could be used to reduce report volume.
  - In this case the measurements use a log with TRUNCATED, WHOLE_ENTITY and PERIODIC where the record-type is RECORD_TYPE_INFORMATION
  - Note that further formalization of this approach is required.
- Consider a power measurement on a photonic system averaged over a 15-minute period.
  - This normally changes slowly such that a sequence of measurements will have the same value. On this basis reporting an initial value and subsequently only changes lead to significant efficiency gain. On that basis an appropriate information-record-strategy (e.g., RECORD_SUPPRESSION_SUPPRESS_EXPECTED and VALUE_EXPECTATION_VALUE_IS_SAME_AS_LAST) could be used to reduce report volume.
  - In this case the measurements use a log with TRUNCATED and WHOLE_ENTITY along with either ON_CHANGE or PERIODIC depending upon the measurement approach where the change is considered from one measurement period completion to the next.
  - Note that further formalization of this approach is required.

3.9.2.7 Bulk Performance Monitoring (PM) data

Essentially covered using multiple periodic measurement response where the current-data instance is streamed once the granularity period ends and the current-data is rolled to history-data (i.e., streaming of history-data creation)[39].

It is expected that PM data may most appropriately be streamed using one of the suppression techniques available and be best encoded in a binary format such as ProtoBuf (ENCODING_FORMAT_PROTOBUF) potentially with a gNMI protocol (CONNECTION_PROTOCOL_GNMI).

Note that further formalization of this approach is required.

[39] Further work is required in the area of PM collection and PM streaming.

3.10 Record content

The stream-record allows for multiple log-records each of which includes conditionally a log-record-header (for all records formalized in this document) and conditionally a log-record-body (conditions identified later in this document).

3.10.1 Log Record Header

The log-record-header provides information common to all records.

The TOMBSTONE record may have only a header.

The log-record-header is as below.

```
grouping log-record-header {
    leaf tapi-context {
        type tapi-common:uuid;
        config false;
        description "The identifier of the context.
            CONDITION: Mandatory where there is information related to more than one tapi context in the stream.";
    }
    leaf token {
        type string;
        config false;
        description "A coded (and compact) form of the fullLogRecordOffsetId.
            This property is used to request streaming from a particular point (e.g., the last correctly handled record).
            For a basic log solution this may simply be the sequence number.
            CONDITION: Mandatory where the stream type is from a compacted log OR it offers an opportunity to recover from a particular
            record using the token.";
    }
    list full-log-record-offset-id {
        key 'value-name';
        config false;
        min-elements 1;
        uses tapi-common:name-and-value;
        description "This property must minimally provide a logging sequence number.
            Note that when compaction is active, the streamed sequence may not have sequence numbers that simply increment by one.
            In a complex log solution there may be various parts to the log.
            The record token is a compressed form of log record reference.
            This property provides the verbose form
            For example, it may include:
            - stream id
            - topic
            - partition
            - partition offset
            - sequence number (the offset is essentially the sequence number associated with the partition)";
    }
    leaf log-append-time-stamp {
        type tapi-common:date-and-time;
        config false;
        description "The time when the record was appended to the log.
            CONDITION: Mandatory where the log is compacted.";
    }
    leaf entity-key {
        type string;
        config false;
        description "The identifier of the entity that is used in a Compacted log as the compaction key.
            The entityKey value, where appropriate, may be based upon the identifiers from the event source.
            It can be built from some specific detail combination that meets the necessary uniqueness and durability requirements.
            entityKey is the value used during compaction.
            Ideally it is a UUID format, if this can be formed from the source identifier.
            CONDITION: Mandatory where the log is compacted.";
    }
    leaf record-type {
        type record-type;
        default "RECORD_TYPE_INFORMATION";
        config false;
        description "The type of the record.
            Can be used to understand which elements of the record will be present.
            CONDITION: Mandatory where not default.";
    }
    leaf record-authenticity-token {
        type string;
        config false;
        description "A token generated using a method that allows the client to validate that the record came from the expected provider.
            CONDITION: Mandatory where authenticity method providing a token is required.";
    }
    description "The header of the log record providing general parameters of the record common to all records.";
}
```

Figure 14 YANG: log-record-header

For this structure, a solution shall provide the following support for a compacted log capability with explanation provided in the YANG above:

- Mandatory:
    - token
    - full-log-record-offset-id[40]
    - log-append-time-stamp
    - entity-key: The entity-key could reasonably be the uuid of the entity concatenated with the appropriate sequence of local ids for the substructure (i.e., the tree). See also 3.10.3 Considering parent-address on page 42).
    - record-type (as the default record type is INFORMATION)
- Conditional Mandatory:
    - record-authenticity-token: allows the provider to supply a value with each record, that is usually different from record to record (e.g., a digital signature where the value of which is set by combining some shared secret with the record content[41]), that the client has a mechanism for confirming so as to validate that the record came from the expected originator and hence to protect against records being inserted in the stream from some unauthorized source.
- Optional
    - tapi-context: This field can be omitted for TAPI 2.4 as focus is a single context.
        - This uuid identifies the applicable TAPI context. For TAPI 2.4 the mapping between the context and its uuid is unspecified.
        - In future this will enable systems that support more than one context to ensure the context of the stream is present in the record.

[40] Kafka, described very briefly in 8.1 Appendix – Considering compacted logs on page 38, supports partitioning of logs to improve scale and performance.
[41] The mechanism for generation of the value and validation of the value has not be set at this stage, it is intended that in a later TAPI version a formal approach to setting the value will be specified.

3.10.2 Log Record Body

The log record body includes generalized content and is augmented with specific content depending upon the value of a control property, record-content.

```
grouping log-record-body {
    container event-time-stamp {
        config false;
        uses approx-date-and-time;
        description "Time of the event at the origin of the event that triggered the generation of the record.
            The structure allows for time uncertainty.
            CONDITION: Mandatory where event time is not conveyed via another property.";
    }
    leaf event-source {
        type event-source;
        default "UNKNOWN";
        config false;
        description "Indicates whether the source is controlled (under management control) or potentially chaotic (under resource control).
            The time characteristic of the source may be determined from the metadata describing the resource (e.g., a detector).
            Where there is an alternative (and probably more detailed) source of information on time characteristic this attribute can be omitted.
            CONDITION: Mandatory where not default.";
    }
    list additional-event-info {
        key 'value-name';
        config false;
        uses tapi-common:name-and-value;
        description "Addition information related to the event such as change reason where changeReason would be the name and the value
            text would provide information on the reason for change.
            CONDITION: Mandatory where there is additional info to convey.";
    }
    leaf-list parent-address {
        type string;
        config false;
        description "Where the entity is a local class this provides the ordered list of ids from the closest global class (a UUID cast as a string)
            to the direct parent (which may be the global class).
            The field can include all entities back to the Context and hence can be used for global classes where the tree is being represented in
            full.
            Gives the position of the entity in the address tree (usually containment) that is raising the event by providing the name/id values in
            the address of the parent.
            Is the sequence of named levels in the tree up to but excluding the entity of the notification.
            It includes the device id where relevant.
            CONDITION: Mandatory where the class has a parent, and the parent is not context.";
    }
    leaf record-content {
        type tapi-common:object-type;
        config false;
        description "The identifier of the object class in the record body detail.
            This property is used to control the conditional augmentation of the body with detail.
            CONDITION: Mandatory where the record content is (the whole of or part of) a standard TAPI object.";
    }
    description "The specific details of the Record.";
```

Figure 15 YANG: log-record-body

For this structure, a solution shall provide the following support for a compacted log capability with explanation provided in the YANG above:

- Mandatory:
    - event-time-stamp
- Conditional Mandatory
    - event-source
    - additional-event-info
    - parent-address (see more details in section 3.10.3 Considering parent-address on page 42):

        - Note that the parent-address may be omitted:
            - For record-type DELETE and TOMBSTONE
- record-content: Note that for a DELETE, there is no need for content and hence this field may be omitted.

The log-record-body is augmented with an instance of a class. This allows for any class from the model to be reported. Hence, when using the Compacted Log approach, the streaming mechanism can be used to gain and maintain alignment with all entities in a context.

The following tables provide details on the types used in the structure above.

```
typedef event-source {
    type enumeration {
        enum RESOURCE_OPERATION {
            description "The event is from the operation of the network resources.
                The event source has a relatively fast time characteristic.";
        }
        enum MANAGEMENT_OPERATION {
            description "Event is from a Management operation (slow control).
                The event source has a relatively slow time characteristic.";
        }
        enum UNKNOWN {
            description "The origin of the event is not known.";
        }
    }
    description "Source of the event.
        Use to give some idea of the time characteristics of the event source.";
}
```

Figure 16 YANG: event-source

```
grouping approx-date-and-time {
    leaf primary-time-stamp {
        type tapi-common:date-and-time;
        config false;
        description "Time of the event at the origin where known precisely.
            Where the event is known to be before particular time, this field records that time.
            Where the event is known to be after a particular time, this field records that time (this is an unusual case where there is no
            proposed before time).
            Where the event is known to have occurred in a time window, this field records the end time (the time before which the event must
            have occurred).";
    }
    leaf start-time-stamp {
        type tapi-common:date-and-time;
        config false;
        description "The time after which the event is known to have occurred when the event is known to have occurred between two times.
            The primaryTimeStamp provides the end time.
            CONDITION: Mandatory where the time is only approximately known and where the event is known to have occurred after a
            particular time.";
    }
    leaf spread {
        type spread;
        default "SPREAD_AT";
        config false;
        description "Indicates the knowledge of the time of occurrence of the event.
            CONDITION: Mandatory where not default.";
    }
    leaf source-precision {
        type source-precision;
        default "SOURCE_PRECISION_UNKNOWN";
        config false;
        description "Indicates how well the source time is synchronized with network time.
            CONDITION: Mandatory where not default.";
    }
    description "Allows for recording of an aspect of imprecise time.";
}
```

Figure 17 YANG: approx.-date-and-time

For this structure, a solution shall provide the following support:

- Mandatory:
    - primary-time-stamp
- Conditional Mandatory
    - start-time-stamp
    - spread
    - source-precision

The following tables provide details on the types used in the structure above.

```
  typedef spread {
    type identityref {
      base SPREAD;
    }
    description "The alternative time of occurrence statements.";
  }

…

  identity SPREAD {
    description "none";
  }
  identity SPREAD_AT {
    base SPREAD;
    description "The event occurred at a particular time.";
  }
  identity SPREAD_BEFORE {
    base SPREAD;
    description "The event occurred before a particular time.";
  }
  identity SPREAD_AFTER {
    base SPREAD;
    description "The event occurred after a particular time.";
  }
  identity SPREAD_BETWEEN {
    base SPREAD;
    description "The event occurred between two stated times.";
  }
```

Figure 18 YANG: spread

```
  typedef source-precision {
    type identityref {
      base SOURCE_PRECISION;
    }
    description "Alternative statements about timing precision at the event source.";
  }

…

  identity SOURCE_PRECISION {
    description "none";
  }
  identity SOURCE_PRECISION_UNKNOWN {
    base SOURCE_PRECISION;
    description "The state of the clock at the event source is not known.
      The view of time of day at the source is suspect.";
  }
  identity SOURCE_PRECISION_FREE_RUNNING {
    base SOURCE_PRECISION;
    description "The clock at the event source is free-running.
      The view of time of day at the source may be significantly different from that at other sources.";
  }
  identity SOURCE_PRECISION_SYNCHRONIZED {
    base SOURCE_PRECISION;
    description "The clock at the event source is appropriately synchronized to the timing master.
      The view of time of day at the source should be essentially the same as that at other time-synchronized sources.";
  }
```

Figure 19 YANG: source-precision

3.10.3 Considering parent-address

The parent-address shall provide the position of the entity in the YANG tree to enable a client to position an entity in the tree.

- Where the entity is a Global class and hence has a UUID[42]
    - Each element (string) in the parent-address shall provide a uuid.
    - The address elements shall be in the order of the tree from the highest to lowest (e.g., for a connection-end-point, the list should start with the uuid of the topology, then have the uuid of the node and finally the uuid of the node-edge-point).
    - The element shall be formed by taking the class name and appending "-uuid" (e.g., node-edge-point-uuid).
- Where the entity is a local class and hence has a local id relative to the UUID of its containing Global class
    - The parent address shall provide any necessary levels of nesting of local-id elements in addition to the YANG tree (including the UUID of its Global class) from highest uuid to lowest local-id in the tree.
    - The local class element shall be formed by taking the class name and appending "-local-id" (e.g., rule-local-id).
- ~~For alarms and events that are reported as independent entities, the parent-address may include address of the indirect parent (i.e., the entity against which the alarm is raised).~~[43]

Note that in a solution with a single context, the context should be omitted from the parent address and where the parent address is solely context (topology-context, connectivity-context etc.), the parent address can be left empty (and hence not provided).

The table below covers each of the cases.

Table 1: Clarifying parent-address for Global Classes

Global Class	**Direct parent class**	**Property in parent covered by parent-address behavior (red highlights where the property name is not the class name)**
context	None	None
service-interface-point	context	service-interface-point
profile	context	profile
topology	topology-context / context	topology
node	**topology**	node
node-edge-point	**node**	owned-node-edge-point
node-rule-group	**node**	node-rule-group
inter-rule-group	**node**	inter-rule-group
link	**topology**	link
connectivity-service	connectivity-context / context	connectivity-service
connection	connectivity-context / context	connection

[42] Can be considered as a Domain Driven Design [DDD] Aggregate.
[43] Note that TAPI 2.1.3 proposed this approach. This is no longer recommended.

Global Class	Direct parent class	Property in parent covered by parent-address behavior (red highlights where the property name is not the class name)
connection-end-point	cep-list / **node-edge-point**	connection-end-point
switch-control	**connection**	switch-control
equipment	**device**	equipment
holder	**equipment**	contained-holder
device	physical-context / context	device
access-port	**device**	access-port
physical-span	physical-context / context	physical-span
path-computation-service	path-computation-context / context	path-comp-service
path	path-computation-context / context	path
oam-service	oam-context / context	oam-service
meg	oam-context / context	meg
oam-job	oam-context / context	oam-job
oam-profile	oam-context / context	oam-profile
oam-job-service	oam-context / context	oam-job-service
oam-job-descriptor	oam-context / context	oam-job-descriptor
cep-fm-data	oam-context / context	cep-fm-data
cep-pm-data	oam-context / context	cep-pm-data
mep-pm-data	oam-context / context	mep-pm-data
mip-pm-data	oam-context / context	mip-pm-data
available-stream	stream-context / context	available-stream
supported-stream-type	stream-context / context	supported-stream-type
stream-monitor	stream-context / context	stream-monitor

In the table above the parent address is formed by chaining the direct parent of the direct parent etc. where only the direct parents listed in bold black are considered (see example structure later in this section).

Parent address can also be used when streaming an instance of local class in the context of the instance of the containing global class. The table below provides a list of local classes. As is the case for the global classes, the referencing property in the parent need not be updated via the stream as the client can determine the necessary update using the parent address.

Table 2: Clarifying parent-address for Local Classes

Local Class	**Direct parent (and parent global class)**	**Property in parent covered by parent-address behavior** (red highlights where the property name is not the class name)
rule	**node-rule-group,** **inter-rule-group**	rule
connectivity-service-end-point	**connectivity-service**	end-point
route	**connection**	route
path-service-end-point	**path-computation-service**	end-point
abstract-strand	**physical-span**	abstract-strand
strand-joint	**abstract-strand (physical-span)**	strand-joint
physical-route	**connection**	physical-route
physical-route-element	**physical-route (connection)**	physical-route-element
topology-constraint	**connectivity-service**	topology-constraint
resilience-constraint	**connectivity-service,** **switch-control**	resilience-constraint[44] (connectivity-service), control-parameters (switch-control)
resilience-route	**route (connection)**	resilience-route
resiliency-route-constraint	**resilience-constraint**	resiliency-route-constraint
layer-protocol-constraint	**resilience-route (connection/route)**	layer-protocol-constraint
routing-constraint	**connectivity-service,** **resiliency-route-constraint**	routing-constraint[45]
switch	**switch-control**	switch
path-objective-function	**path-computation-service**	objective-function
path-optimization-constraint	**path-computation-service**	optimization-constraint
oam-service-point	**oam-service**	oam-service-point
mep	**connection-end-point,** **meg,** **node-edge-point** **oam-service-point (oam-service)** [46]**,** **current-data (oam-job)**	mep

[44] Note that there is only a single instance of resilience-constraint so there is no local id and the field is not a list. This applies to other properties in the list.
[45] Note that there is only a single instance of routing-constraint so there is no local id and the field is not a list. This applies to other properties in the list.
[46] Using the parent-address format highlighted in this section.

Local Class	Direct parent (and parent global class)	Property in parent covered by parent-address behavior (red highlights where the property name is not the class name)
mip	**connection-end-point,** **meg,** **node-edge-point,** **oam-service-point (oam-service)**[47]**,** **current-data (oam-job)**	mip
current-data	**oam-job**	current-data
history-data	**current-data (oam-job)** **cep-pm-data** **mep-pm-data** **mip-pm-data**	history-data
pm-data	**oam-profile**	pm-data
pm-parameter-config	**oam-job-service**	pm-parameter-config

The parent-address is expected to follow the structure set out in the examples below:

- For a connection-end-point:

```
"parent-address" : [
   "topology-uuid:4e537278-79f8-39ad-804b-f0b553cb2ffb",
   "node-uuid:7f65a8c1-4e0d-3296-b06a-460b2367ca76",
   "node-edge-point-uuid:e35a31b2-1096-38f9-8e3e-dca010e2f191"
    ],
```

- For a rule in a node-rule-group:

```
"parent-address" : [
   "topology-uuid:4e537278-79f8-39ad-804b-f0b553cb2ffb",
   "node-uuid:7f65a8c1-4e0d-3296-b06a-460b2367ca76",
   "node-rule-group-uuid:a35e32c2-1256-3a89-203e-dca111e2f191",
   "rule-local-id:7"
    ],
```

- For a resilience-route:

```
"parent-address" : [
   "connection-uuid:e35a31b2-79f8-39ad-804b-f0b553cb2ffb",
   "route-local-id:4",
   "resilience-route-local-id:1"
    ],
```

[47] Using the parent-address format highlighted in this section.

As parent address is provided, it is not necessary for the provider to update the list reference attribute in the parent (in the example above, the connection-end-point list attribute in the node-edge-point `e35a31b2-1096-38f9-8e3e-dca010e2f191`). Indeed, it is recommended that the provider does not update this attribute with the changes to the containment list property corresponding to the streamed entity. It is expected that the provider does NOT provide any containment list property that reference entities identified by a uuid[48].

Instead, the client system can determine the necessary update from the parent-address data and the relevant structure in the tree can be created and updated by the client. This approach removes the need to send two records for the creation of an entity and hence removes race condition and atomicity challenges.

For the example above the connection-end-point list of the node-edge-point identified in the parent-address would be updated with the uuid of the connection-end-point.

Considering condition-detector (see 4.5 Condition detector and alarm structure on page 56), as it does not have a parent the parent-address should be omitted. The measured-entity-uuid of the condition-detector is sufficient to identify the affected entity.

Where the entity-key is a uuid, the uuid alone is sufficient to identify an entity that is to be deleted, hence the parent-address for a DELETE record can be omitted. The onus is on the client to update the corresponding containment tree references, so in the example above a connection-end-point delete would require the client to remove the connection-end-point from the cep-list of the appropriate node-edge-point.

When the connection-end-point is deleted, the event will provide the connection-end-point uuid. The client will locate the connection-end-point by uuid, remove it and clean up any references as appropriate.

3.11 Considering order/sequence and cause/effect

3.11.1 Time

When determining the cause and effect of any behavior in the controlled system it is necessary to have visibility of relevant state and to know the time of each change of state. The time units must be sufficiently fine to allow all relevant event sequencing to be determined[49].

The time of change at the source of change needs to be propagated as data in the stream and hence needs to be in the report of each instance of the changed entity (and in any stored form that exists between the source and the stream). This is recorded in event-time-stamp.

The time the record was logged is log-append-time-stamp.

3.11.2 Backend stream details

The implementation solution may partition the log supporting a stream (see section 8.1.4 Partitions on page 126). This may cause stream content reordering. Clearly, event reordering across different sources is a fundamental behavior due to relativity and differential delay. It is expected that the order for events from each single state machine will be maintained.

[48] For example, an instance of node-edge-point (global class with its uuid) includes all locally identified branches and leaves but NOT contained instances of connection-end-points, as the connection-end-point is a global class separately identifiable using its uuid. Note also that the attributes that reference (list) the contained global class instances (e.g., connection-end-point attribute in the node-edge-point) are NOT included. This information is conveyed by parent-address.

[49] It may be beneficial to add an indication of time granularity to assist in cause/effect evaluation. For this to be fully beneficial the accuracy of synchronization of time would also need to be determined).

Regardless, the timestamp granularity must be sufficient to ensure relevant chronological order can be recovered. This also assumes that robust time synchronization mechanism is present at each event source and hence in the controller/controlled system as a whole.

3.12 The Context

TAPI is a machine-machine interface. As noted earlier, the client maintains a local repository representation of the information from the provider. The information is defined in terms of the context. As the context will have been “agreed” (currently, up front during system design), the context is what the client wants/needs to see.

Where the client needs less detail than is provided by a specific context the client can:

- Locally filter out information that is not of interest.
- Change the context.
    - The context can be modified[50] as necessary so long as other clients (and the provider) agree with the change.
- Request construction of a specific context
    - Several contexts can be provided.

On this basis:

- Individual specific queries on the provider are not necessary.
- Notification filters, that are not simply the realization of the view, are not necessary. The context defines the filtering for the client.

Any changes in the required information are handled by changes in the context. See later discussion on changing the context and spotlighting in section 7.10 Use cases beyond current release on page 122.

3.13 Handling changes in the Context

The stream relates to the entities in the context. Changes include:

- An instance of a thing being added/removed to/from context within the current definition of the context.
    - E.g., the creation of a connection
- The context definition being changed such that an instance of a thing appears/disappears.
- An instance of a thing in the context changing such that an element is added/removed from the thing.
- The value of a property of an instance of a thing already in the context changing.

See later discussion on changing the context in section 7.10 Use cases beyond current release on page 122.

3.14 Reporting change

For the compacted log solution, if the log-storage-strategy is WHOLE_ENTITY_ON_CHANGE, whole entities are streamed on creation, deletion and change of a property. Hence, for example, if a single property in a CEP changes, the whole CEP is logged and streamed. The parent-address indicates where the CEP is in the tree and hence which NEP and node have changed (in CEP detail)[51]. See also section 5 Reporting Change Only on page 70.

[50] Context adjustment is not available via TAPI in the current version

[51] Only the CEP needs to be notified as this identifies where the NEP and the node have changed.

3.15 Model implications

Separation of properties that have a slow rate of change (intent-driven config) from properties that have a high rate of change (network-derived state) and isolating the high rate of change properties in independent separate state entities is advisable. The alarm and state model provided for streaming allows a separation of small per-state entities from large slow changing config entities. The current TAPI OAM model also isolates properties related to monitoring results from properties related to configuration.

3.16 System engineering

For the solution to operate reliably:

- System engineering must be such that under normal and “normal bad day” circumstances the client is able to keep up well with the provider (otherwise the client will suffer ongoing alignment issues in terms of lag and potentially in terms of fidelity).
    - Eventual consistency is acceptable in a control solution if there is only a “short” delay to alignment with any specific state.
- For realignment to be successful, the client must be engineered to be able to read all records in the tail of the log (starting from the oldest) up to the tombstone-retention point in under the tombstone-retention time. On this basis
    - The tombstone-retention time may differ for different cases and different streams.
    - For a stream with only limited volumes of long term records the tombstone-retention could be quite short
        - Under these circumstances, tombstone-retention is probably determined by likely communication failure duration.
    - For the case where alignment takes days, tombstone-retention would need to be in terms of days.

The solution can be tuned to balance pace to achieve consistency (eventual consistency) with the fidelity of information when under stress[52].

3.17 Eventual Consistency and Fidelity

3.17.1 Eventual Consistency

A simple way to understand eventual consistency is to imagine a network that is changing such that there is a stream of changes and where the client has not absorbed the stream. If suddenly the network were to stop changing, then, once the client has absorbed the entire stream, the client will be aligned with network state.

3.17.2 Fidelity

In this document, fidelity is the degree of exactness with which the controlled system behavior is represented via TAPI streaming.

Loss of fidelity is the loss of some details of change (without losing eventual consistency). For example, the operational-state property of a CEP may initially be ENABLED, then become DISABLED and then ENABLED again. Loss of fidelity may lead to the operational-state being observed as continuously ENABLED.

[52] Compaction is used intelligently to reduce realignment time whilst minimizing probability of loss of detail.

3.17.3 Related stream characteristics

Considering the supported-stream-type structure discussed in 3.7.1 Supported stream type on page 16, there are two key time settings for the compacted log solution tombstone-retention[53] and compaction-delay[54].

- Client read delay is less than compaction delay.
    - Client is behind on absolute state but is losing no detail.
    - The client can potentially catch back up if:
        - Rate of append (i.e., the rate of arrival of information relevant to the stream) reduces due to conditions in the monitored environment changing.
        - The client gains additional resources that allow it to deal with greater than the current rate of append.
- Client read delay is greater than the compaction delay but less than the tombstone-retention time.
    - Client is behind on absolute state and is losing fidelity as some changes are being compacted out of the log (if the client is less interested in short lived things than long lived things this may not be a significant problem)
        - A rapid intermittency may become completely invisible (e.g., an active – clear alarm pair)
        - All changes that happen at a rate slower than read delay will be visible.
    - The client can potentially catch back up if:
        - The delay was due to a long comms down issue that has now recovered, and the client capacity can readily deal with the current append rate.
        - Rate of append reduces due to conditions in the monitored environment changing.
        - The client gains additional resources that allow it to deal with greater than the current rate of append.
- Client read delay is greater than the tombstone-retention time.
    - Client has now potentially lost eventual consistency and must realign by streaming from the “oldest” record (offset zero)

3.18 Stream Monitor

TAPI 2.4 also offers a rudimentary capability for monitoring the streams. This allows an external stream administration client that has the appropriate capability and authorization to monitor which clients are connected and how their stream is performing.

Where this capability is supported for each client, streaming connection is monitored for the id of the last record written to and read from the log. This allows an administrator to get a view of how delayed a client is.

In TAPI 2.4 this is for PoC (Proof of Concept) analysis. It is expected that the feature will advance significantly in future releases as a result of experience gained from PoC activity.

3.19 Solution structure – Architecture Options

Two options are explored, one provides full compaction support, the other uses a more traditional structure to feed a stream and provides a restricted emulation of compaction. Either can be used to support the current TAPI streaming solution sufficiently.

[53] This is the duration that TOMBSTONE records are retained for. This prevents the log size being unbounded.
[54] This is the duration that all records are guaranteed to be persisted for (after they are logged) before compaction will consider removal based upon more recent logging for the same entity.

There may be other approaches that provide a suitable capability, i.e., reporting current state and change via whole entity reporting through the same single stream, so as to achieve eventual consistency with current state, cost-optimized for potential loss of fidelity.

The key consideration is the external behavior and not the specific mechanism used to support it.

3.19.1 Full compacted log

The figure below shows a stylized view of a controller controlling a network (the controlled system) and providing a stream to a client.

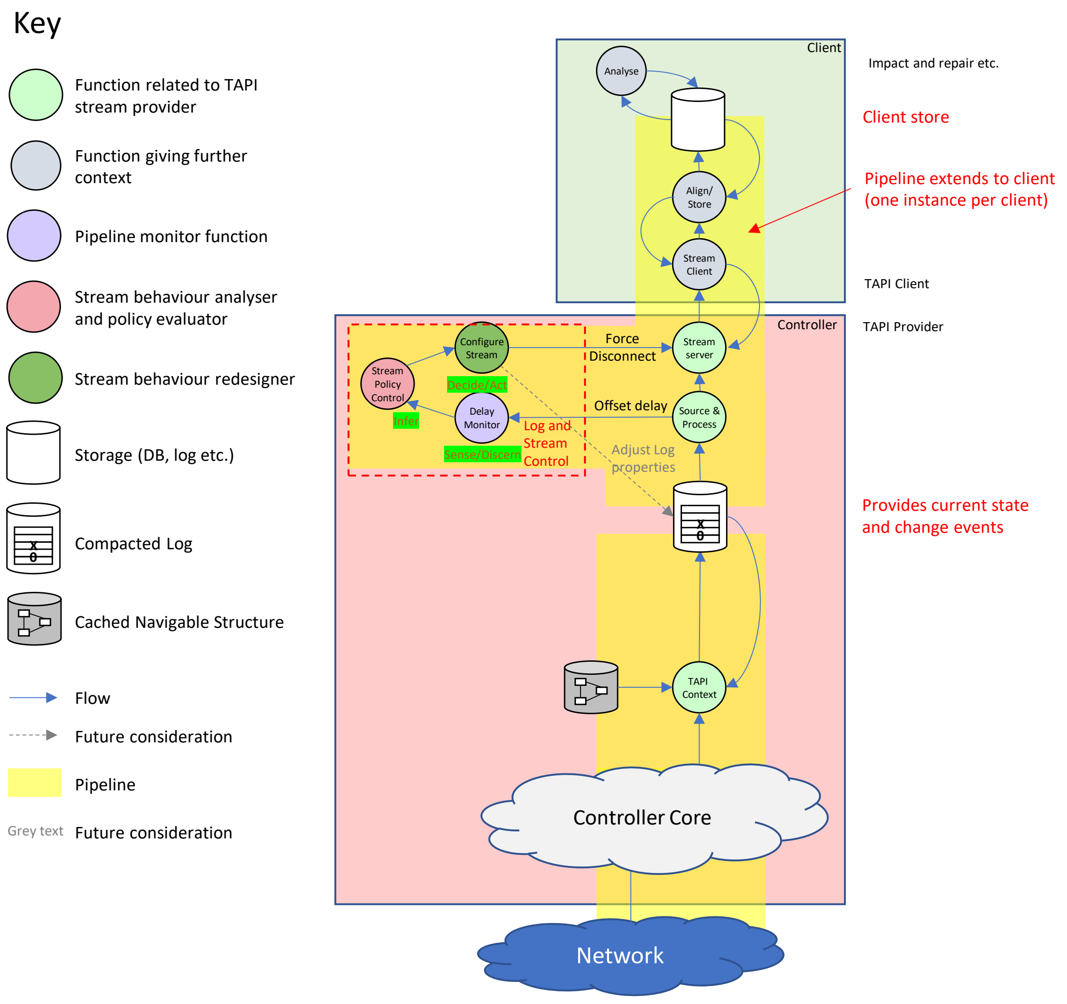


Figure 20 Stylized view of example controller offering full compaction.

The key features highlighted in the diagram are:

- Compacted log providing current state, recent TOMBSTONE records and recent changes.
  - The compacted log records whole entity instances on change
  - See Appendix – Considering compacted logs
- Pipeline providing guaranteed delivery (at least once) whilst connection in place along with provider-initiated connection force drop control when realignment is necessary.

    - To align the client does not need to do anything other than connect to the appropriate stream and receive from the stream.
    - It is assumed that the client will extend the integrity of the stream to include the process of storage (as shown) so as to enable the client to maintain alignment with current state (eventual consistency) and to build and maintain history.
- A stream monitor (depicted as a control loop on the pipeline as an aspect of control of control) that ensures achievement of eventual consistency.
    - Delay Monitor senses the age of the records being streamed (delay)
    - Stream Policy Control infers whether the current delay is acceptable. This depends upon what state the stream is in (e.g., start-up) etc. It provides input to Configure Stream.
    - Configure Stream decides what action to take and then takes that action. This may include forcing a disconnect to cause a client-reconnect and subsequent realignment and/or adjustment of Log properties.

Note that the assumption is that other pipelines are in place (not shown here) throughout the overall flow from controlled device through Controller Core to the client to ensure no relevant loss of information from the controlled device and to ensure that the solution is always in a state of eventual consistency with current network state.

3.19.2 Emulated compaction

This approach uses the same pipeline mechanism but feeds the pipeline from a composite store. This does not offer the full compacted log capability. The behavior is as if the tombstone-retention and the compaction delay are set to the same value (simplistically, the end of the log, but strictly any point in the log).

On client connection the provider would:

- Start the “time-truncated” log to collect all changes that occur from the point of connection[55]
    - These changes would be recorded as whole entity snapshots or as change-only entity snapshots depending upon the.LOG_RECORD_STRATEGY
- Start to stream “current state”.
    - Current state should include time of occurrence of last change.
    - To achieve this the provider may need to page through a large store of data such that the state is skewed over some, potentially long, period of time.

Monitored System (e.g., Network) state may change during streaming of current state such that a change statement (whole entity snapshot) is appended to the truncated log. It is quite likely that the change log will have logged both changes that are ahead of the “current state” being provided as well as changes that have already been covered as part of the “current state”.

Once the provider has sent “current state” (which will include all states that have not changed since connection of the client), it would then begin to stream from the truncated log. As noted, the truncated log may include some states that have already been streamed. As the client is necessarily designed to be able to deal with repeated statements this will not be an issue.

If the connection to the client is dropped, the client will reconnect providing the token of the last record it fully processed. The provider can code into the token any information that helps it determine what to stream next.

[55] It should be feasible for a single truncated log to be used for multiple clients; however, the current state will need to be dealt with on a per client basis.

Clearly, the client must be able to retrieve current state in a shorter period than the log truncation time so that changes are not lost. If the log wraps during streaming of current state, the provider will have to restart the alignment (by dropping the connection and by ignoring the client token).

It would not be unreasonable for the provider to shorten the truncated log once alignment has been achieved.

The provider is expected to add a TOMBSTONE record for every DELETE record. The emulated compacted log could avoid using the specific TOMBSTONE record as there is no log size benefit of retaining only a compressed record (as offered by the TOMBSTONE record).

It should be possible for the provider to use one single truncated log for all clients, however on initial connection of a client it will be necessary for the provider to construct a specific current state snapshot to feed the stream to that client.

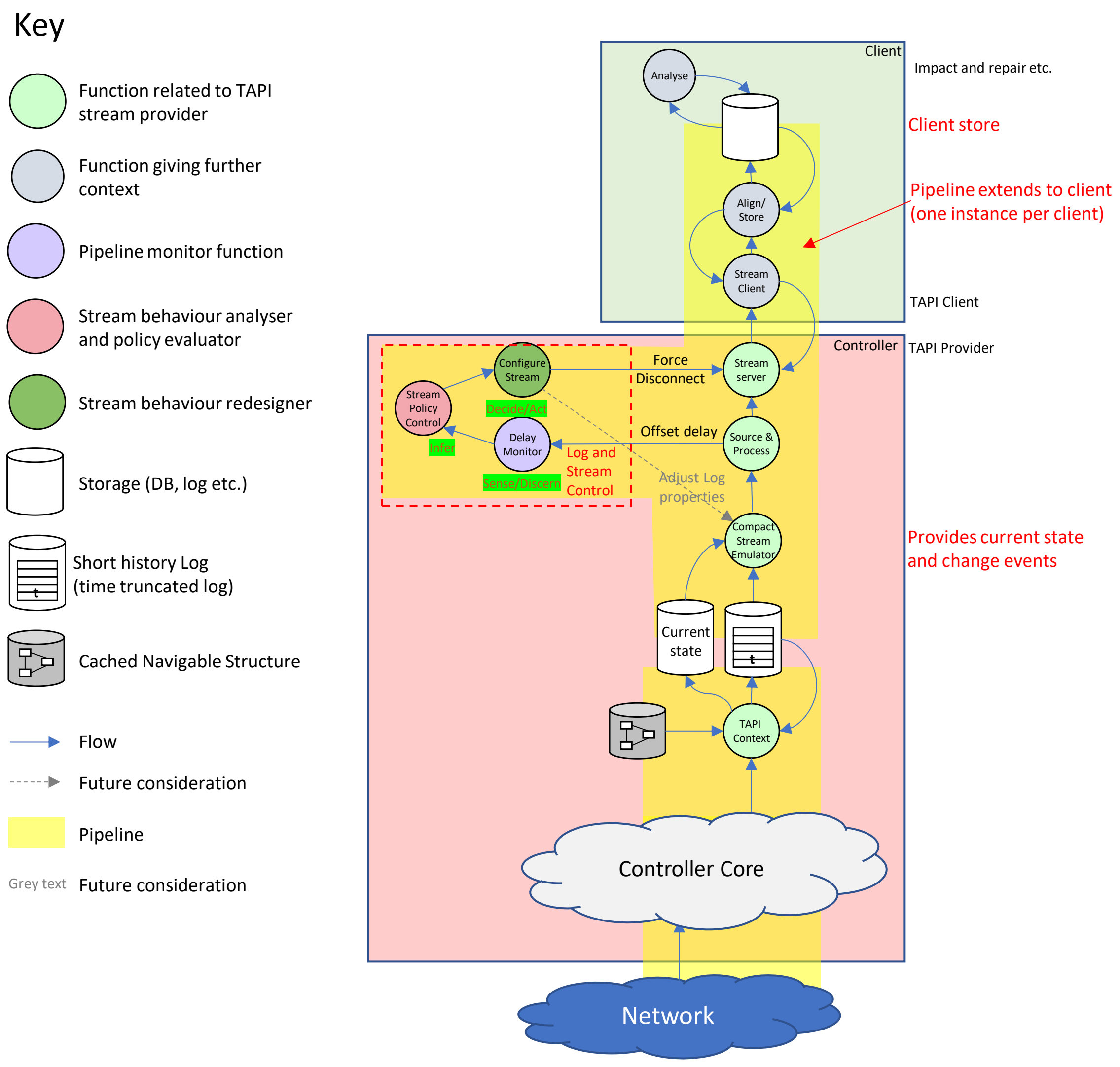


Figure 21 Stylized view of example controller offering emulated compaction

The behavior of the provider feeding the stream is very similar to a traditional behavior (essentially the same as send current state in response to get then notify of change, where the log is CHANGE_ONLY. Where the log is WHOLE_ENTITY a notifications of change will include the whole entity).

In this realization the provider side does not carry out active compaction and hence does not provide the elegant, degraded performance of the full compacted log approach when the client is under pressure and significantly delayed.

3.19.3 Comparing the full compacted log and the emulated compacted log

From the client perspective the emulated compacted log appears to be a compacted log with the compaction delay equal to the tombstone-retention.

The emulated compacted log is also more likely to have a tombstone-retention determined by record count as opposed to time.

4 Using the compacted log approach for alarm reporting

This section provides both informative and normative statements on the alarm solution.

4.1 Specific alarm characteristics - raising/clearing an alarm

A controlled device raises an alarm once specific criteria have been met and clears the alarm once other specific criteria have been met. The criteria could involve, for example:

- Counting traffic related events where the count is within some window (often sliding and sometimes complex)
    - The alarm will be raised when the count exceeds some threshold and will be cleared when it drops below some threshold.
        - The two thresholds provide hysteresis that brings some stability to the alarm.
    - The windows are often very short and can cause extreme intermittency under particular network scenarios.
- Measuring an analogue property with several options
    - The alarm may be raised when the measurement exceeds some threshold and be cleared when it drops below another threshold with hysteresis.
    - The alarm may be raised when the measurement drops below some threshold and be cleared when it exceeds another threshold with hysteresis.

The counts can be of the occurrence of other threshold crossing and hence the alarm definition at origin may be very complex.

4.2 Key Features of an alarm solution (example usage)

The following sections work through the key features of an alarm solution as an example of usage of streaming.

The alarm streaming solution has a particular DELETE/TOMBSTONE behavior to provide the best performance. This is highlighted in 4.7 Alarm TOMBSTONE record behavior on page 67.

4.3 Log strategy

This section identifies the key characteristics of the alarm log and stream and summarizes the implications.

1. The TAPI alarm stream shall be fed from a compacted log.
2. There should be a dedicated connection through which only alarm records are propagated.
3. The log compaction delay shall be set to allow for normal operation with a client to enable the system to deal with temporary communications failure with no loss of fidelity.
    - When compaction is applied fidelity will be lost, although only rapidly changing and fleeting alarms will be lost.
    - In any solution, the client may be a little behind or may suffer a short communication disruption without significant impact on operational quality.
    - The proposed compaction delay setting is 10 minutes[56] for a system with reasonably well engineered platform capacity and communications.

[56] Assuming that an acceptable system will be engineered to not be any more than 10 minutes behind under bad-day conditions. Note that this parameter can be tuned to suit system engineering and also desire to achieve full fidelity. A longer compaction delay will cause there to be more recent history in the initial alignment and hence may slow initial alignment. A similar behavior occurs for a traditional notification queue. It is possible to dynamically tune the stream to reduce this impact.

        - Clearly having an alarm system that is greater than 10 minutes behind the provider will degrade location performance, however experience may show that this time is too short.
    - The solution may allow for this property to be adjustable in the running system, through a mechanism suitable for expert access.
        - Dynamic compaction delay might be beneficial in cases where local control can be applied, and an unusually long comms down is being experienced.
            - Under these circumstances the compaction delay could be increased such that the clients lose no fidelity (although they are clearly behind whilst the comms is down)
    - Note that:
        - ideally, during the normal operation, the client would be at most a few minutes behind the current append time.
        - the volume of information within the compaction delay time depends upon the rate of change of things in the monitored environment.
        - in a sophisticated solution it will be possible to allocate more resources to the client application, when it is under pressure, to enable it to process faster.
4. The log tombstone-retention shall be set to allow for reasonable communication failure/recovery where there may be significant failures and to allow for client reboot.
    - The proposed tombstone-retention setting is 4 hours for a system with reasonably well engineered platform capacity and communications.
        - This solution may allow for this property to be adjustable in the running system, through a mechanism suitable for expert access.
        - The tombstone-retention will always be greater than or equal to the compaction delay.
5. The normal record (not a TOMBSTONE record) retention shall be infinite.
    - This property is not adjustable as it is fundamental to the correct operation of the mechanism
6. If compaction operates on a segment by segment[57] basis such that the segment with the most recent alarms is never compacted (see section 8.1.3 Log segments on page 126), then the segment size shall be such as to not hold significantly more records than would occur during the compaction delay time when operating under normal conditions (ideally a segment would hold significantly less records)[58].

Also see section 3.16 System engineering on page 48.

4.4 Alarm behavior

The following highlights key design decisions (in the context of justifying/explanatory information) and should be read in conjunction with the background provided in section 8.4 Appendix – Detectors, detected conditions and alarms on page 137:

1. An alarm detector shall have an ACTIVE and a CLEAR state

    - Explanatory Information:
        - An active is considered as more important than the clear (hence the states are asymmetric in importance)
        - Most detectors in the network will be clear under normal circumstances (hence the normal state can be considered as clear and the off-normal state as active)
    - The alarms shall be reported as ACTIVE or CLEAR.

[57] The log (on disc) saves records in a live area, a segment, that has a defined size. Once the size is reached a new segment is started for saving new records and the old segment is moved to a historic state. This old segment is available for compaction.
[58] The max-allowed-segment-roll-delay parameter forces a partially filled segment to roll over if the delay is exceeded. The alarm stream may become quiet and compaction behavior may benefit from this parameter setting.

        - An ACTIVE will be followed at some point by a CLEAR (the basic state machine is simple with only Clear → Active and Active → Clear transitions)
    - An alarm CLEAR shall be represented by a DELETE record which shall be followed immediately by a TOMBSTONE record.
        - A TOMBSTONE record alone shall be considered as equivalent to a CLEAR
        - The TOMBSTONE record causes the alarm to be removed from the log. This then has a similar characteristic to a traditional alarm reporting mechanism where there is a store of currently active alarms[59].

2. If an entity upon which the detector is lifecycle dependent is deleted and just prior to the deletion the alarm was active, then a TOMBSTONE record shall be appended to the log for the alarm (and hence its detector).
    - The controlled system may report an alarm clear event to remove the alarm and hence the normal DELETE/TOMBSTONE record pair will be appended to the log
    - If the alarm was not active, then there shall be no TOMBSTONE record appended as there is nothing in the log to remove.
3. The output of a particular detector may be processed and stabilized such that it gains the state INTERMITTENT, indicating that the detector is intermittently rapidly cycling through active and clear states.
    - The alarms will be reported as ACTIVE, INTERMMITTENT or CLEAR
    - All transitions are legal (i.e., Clear → Active, Clear → Intermittent, Active → Clear, Active → Intermittent, Intermittent → Active, Intermittent → Clear)
    - The trigger for moving to/from Intermittent state will be defined by policy. Ideally the policy is configurable[60].

Note: It would be reasonable to also propagate other processed network property types through the same stream as alarms if the network property has similar characteristics to an alarm. For example, operational state is asymmetric in importance with a normal state and off-normal state where the normal state could be considered as equivalent to a clear.

4.5 Condition detector and alarm structure

The condition-detector and related alarm structures are described in detail in this section.

There have been advancements made in this area from TAPI 2.1.3. The condition-detector (see section 4.5.1 condition-detector from tapi-streaming.yang on page 57) augments the log-record-body as was the case for TAPI 2.1.3. However, the use of this structure has been reduced and a new augment from tapi-fm.yang, detected-condition (see 4.5.2 detected-condition from tapi-fm.yang on page 60) is now to be used to convey alarm and pm details (as it is also now used for tapi-notification). The TAPI 2.1.3 augment of alarm-condition-detector-detail (see section 4.5.3 alarm-condition-detector-detail from tapi-streaming.yang on page 66) is still available, but this has been deprecated and is not recommended.

A TAPI 2.4 conformant solution can either use the approach defined in TAPI 2.1.3 (in tapi-streaming.yang, see section 4.5.3) or use the new approach define here using that structures specific in tapi-fm.yang (see section 4.5.2).

[59] As noted earlier, almost all detectors in the network are in clear state. Reporting all detector states would be extremely inefficient as detectors that are not active are of little interest.
[60] Configuration of the policy is outside the scope of this specification.

4.5.1 condition-detector from tapi-streaming.yang

```
grouping condition-detector {
    leaf condition-native-name {
        type string;
        config false;
        description "The name used for the Condition by the source of the information.";
    }
    leaf measured-entity-uuid {
        type tapi-common:uuid;
        config false;
        description "The uuid of the TAPI entity that represents the entity measured at source.
            If the TAPI entity cannot be identified as it cannot be mapped, then this property can be omitted.
            If the TAPI entity is a local class, then this is the UUID of the GlobalClass parent of the entity of which this is part.
            CONDITION: Mandatory where there is a standard TAPI entity (normally the case).";
    }
    leaf measured-entity-native-id {
        type string;
        config false;
        description "The identifier (invariant over the life) of the instance of the measured entity at the source.";
    }
    leaf measured-entity-device-native-name {
        type string;
        config false;
        description "The name of the device (as used by the device) that includes the measured entity.
            CONDITION: Mandatory where the device name is necessary to interpret the detector native id.";
    }
    leaf condition-normalized-name {
        type string;
        config false;
        description "It is often the case that there is a Condition Name that is commonly used or even standardized that has not been used by the source of the condition.
            If this is the case, then that common/standard name is provided in via this property.
            CONDITION: Mandatory where the condition has a normalized name.";
    }
    leaf measured-entity-class {
        type tapi-common:object-type;
        config false;
        description "The TAPI class of the measured entity.
            If the class cannot be identified as it cannot be mapped, then this property can be omitted.
            CONDITION: Mandatory where the measured entity class is known.";
    }
    leaf detector-uuid {
        type tapi-common:uuid;
        config false;
        description "The uuid of the TAPI entity that represents the detector.
            If the TAPI entity cannot be identified as it cannot be mapped, then this property can be omitted.
            Where the detector is not modelled independently, but instead is a part of the measured entity such that it is identified by a 'local id' built from the UUID of the measured entity and
            the condition name, then this property may be omitted.
            CONDITION: Mandatory where the detector has a normalized form with a uuid.";
    }
    leaf detector-native-id {
        type string;
        config false;
        description "The identifier (invariant over the life) of the instance of the detector at the source (e.g. a device).
            The string reported in this field must include the:
            - device identifier
            - one or more resource identifiers including that of the measured entity
            It need not include the condition name.";
    }
    leaf condition-detector-type {
        type condition-detector-type;
        config false;
        description "Identifies the type of detector.
            This drives the conditional augmentation.
            Some types of detector may not need specific augmentation.";
    }
    leaf-list measured-entity-local-id {
        type string;
        config false;
        description "Where the measured entity is a local class and hence does not have a UUID the local ID is provided in conjunction with the parents ID.
            The parent may also be a local class in which case its ID is a a local ID along with its parent ID.
            There will be a parent which is a global class which then supplies a UUID.
            The ID of the entity that is being measured is the combination of the UUID and the ordered list of local IDs.
            The local ID may not be provided where:
            - the report about a global class
            - the report is relying on the detectorNativeId.
            CONDITION: Mandatory where the measured entity is a local class and hence needs local id as well as parent uuid.";
    }
    description "ConditionDetector represents any monitoring component that assesses properties of something and determines from those properties what conditions are associated with
        the thing.
        For example, a thing might be 'too hot' or might be 'unreliable'.
        The monitor may a multi-state output.
        The ConditionDetector lifecycle depends upon the lifecycle of the thing it is monitoring (this is a general OAM model consideration).
        The entityKey in the AppendLogRecordHeader for a ConditionDetector record is the nativeDetector Id which may be derived from other ids (most robustly, nativeOwningEntityName
        (to which the detector is associated) +
        natveConditionName).";
}
```

Figure 22 YANG: condition-detector

For this structure a solution shall provide the following support:

- Mandatory:
    - measured-entity-native-id
    - detector-native-id
    - condition-detector-type
- Conditional Mandatory:
    - measured-entity-device-native-name
    - condition-native-name[61]:
        - This is a Mandatory attribute if the TAPI 2.1.3 approach (augmenting with alarm-condition-detector-detail) is being used.
        - This is optional if the detected-condition augment is being used as it duplicates the mandatory property detected-condition-native-name of that augment
    - condition-normalized-name:
        - This is a Mandatory attribute if the TAPI 2.1.3 approach (augmenting with alarm-condition-detector-detail) is being used and if there is a normalized name available for the condition
        - This should not be used if detected-condition augment is being used as it duplicates the mandatory property detected-condition-native-name of that augment
    - measured-entity-class[62]
    - measured-entity-uuid
    - measured-entity-local-id
    - detector-uuid[63]

[61] It is vital that the alarms stated on TAPI reflect what would be seen at the device regardless of the mappings etc. such that the operator can reference the vendor definitions etc.
[62] Some devices report alarms related to entities external to the device. In these cases, there is no TAPI entity to report against.
[63] The detector may have a distinct UUID or may simply have a UUID constructed from that of the entity for which detection is being performed (e.g., a CEP) along with the detector name.

The following table provide details on the types used in the structure above.

```
  typedef condition-detector-type {
      type identityref {
          base CONDITION_DETECTOR_TYPE;
      }
      description "The type of condition detector.
          The type relates to the characteristics of the detection and reporting strategies.
          This drives the conditional augment.";
  }
….

  identity CONDITION_DETECTOR_TYPE {
      description "none";
  }
  identity CONDITION_DETECTOR_TYPE_ALARM_DETECTOR {
      base CONDITION_DETECTOR_TYPE;
      description "A type of detector used for reporting problems.
          The underlying raw detector is two state from the perspective of the monitored condition.
          The detector is asymmetric in nature.
          One state indicates that there is a problem and the other state indicates that there is no problem.";
  }
  identity CONDITION_DETECTOR_TYPE_EVENT_DETECTOR {
      base CONDITION_DETECTOR_TYPE;
      description "A type of detector used for reporting events.";
  }
  identity CONDITION_DETECTOR_TYPE_PM_THRESHOLD_DETECTOR {
      base CONDITION_DETECTOR_TYPE;
      description "A type of detector used for reporting threshold crossing events related to performance monitoring.";
  }
```

Figure 23 YANG: condition-detector

4.5.2 detected-condition from tapi-fm.yang

TAPI 2.4 provides alarm and performance monitoring structures that are common for both tapi-notification and tapi-streaming ensuring that the two use a common approach.

The detected-condition replaces the alarm-condition-detector from TAPI 2.1.3.

```
grouping detected-condition {
    leaf detected-condition-name {
        type tapi-common:detected;
        description "The name of the Condition, e.g. an alarm probable cause or the PM metric name which threshold crossing alert refers to.
            ITU-T probable cause of the failure (detected fault).
            G.806:
            - fault: A fault is the inability of a function to perform a required action. This does not include an inability due to preventive maintenance, lack of external
              resources or planned actions.
            - fault cause: A single disturbance or fault may lead to the detection of multiple defects.
            - defect: The density of anomalies has reached a level where the ability to perform a required function has been interrupted.
            Defects are used as input for performance monitoring, the control of consequent actions and for the determination of fault causes.
            A fault cause is the result of a correlation process which is intended to identify the defect that is representative of the disturbance or fault that is causing
            the problem.
            - failure: The fault cause persisted long enough to consider the ability of an item to perform a required function to be terminated. The item may be
              considered as failed; a fault has now been detected.
            - alarm: A human-observable indication that draws attention to a failure (detected fault) usually giving an indication of the severity of the fault.";
    }
    leaf detected-condition-native-name {
        type string;
        description "The name used for the Condition by the source of the information.";
    }
    leaf detected-condition-native-info {
        type string;
        description "Additional info of the Condition provided by the source of the information.";
    }
    leaf detected-condition-qualifier {
        type string;
        description "Further information necessary to precisely/uniquely/unambiguously identify the Condition Detector.
            For Equipment and Processing Alarm Category, e.g. the local id of the ActualNonFieldReplaceableModule which identifies exact alarm source.
            For Environment Alarm Category, e.g. on the same Device instance may appear more Environmental alarm notifications with same Alarn Name.
            For Connectivity Alarm Category in case that same CEP instance includes e.g. both OTS and OMS monitoring layers.";
    }
    leaf oam-job {
        type tapi-common:uuid;
        description "Reference to the OamJob instance for which the Condition detection has been configured, e.g. configuration of PM metrics and threshold
            values and/or of the (alarm) Conditions.
            The reference is defined as simple UUID because TapiFm does not import TapiOam.
            MEF 35.1: Identification of the PM Session for which the TCA Function was configured.";
    }
    container pm-metric-info {
        uses pm-metric-info;
        description "The PM metric information.";
    }
    container detector-info {
        uses detector-info;
        description "The detector info for alarm and TCA.";
    }
    container simple-detector {
        uses simple-detector;
        description "The simple detector state.";
    }
    description "A record of the state of a Detector where that Detector has two underling states that are of asymmetric importance.
        For example, an alarm or a threshold crossing alert detected on a given resource.
        A Condition Detector represents any monitoring component that assesses properties of something and determines from those properties what conditions
        are associated with the thing.
        For example, a thing might be 'too hot' or might be 'unreliable'.";
}
```

Figure 24 YANG: detected-condition

For this structure a solution shall provide the following support:

- Mandatory:
  - detected-condition-name
    - Note that the enumerated string “ALARM_NAME_NATIVE” may be used where there is no standard name available.
  - detected-condition-native-name[64]
- Conditional Mandatory:
  - detected-condition-native-info: Mandatory where there is additional information to convey.
  - detected-condition-qualifier: Mandatory where the condition requires further qualification (as for the examples provided in the YANG description)
  - oam-job: Mandatory where there is an associated oam-job
  - pm-metric-info: Mandatory where the information relates to a pm-metric
    - Note that performance monitoring threshold crossing alerts are not covered in detail in this document.
  - detector-info: Mandatory where detector-category is to be provided along with other legacy properties.
  - simple-detector: Mandatory where the detector has state (e.g., for an alarm)

The following tables provide details on the types used in the structure above.

[64] It is vital that the alarms stated on TAPI reflect what would be seen at the device regardless of the mappings etc. such that the operator can reference the vendor definitions etc.

```
  grouping pm-metric-info {
    container threshold-observed-value {
      uses tapi-common:pm-parameter-value;
      description "The observed value of PM metric to which TCA refers to.";
    }
    container threshold-configured-value {
      uses tapi-common:pm-parameter-value;
      description "The configured threshold value of PM metric to which TCA refers to.";
    }
    container granularity-period {
      uses tapi-common:time-period;
      description "The granularity period or measurement interval time.
        This parameter may be necessary when the reference to the OAM Job is not included, e.g. in case the OAM job is not visible at the
        management interface.";
    }
    description "Information associated to a Threshold Crossing Alert.";
  }
…
  grouping pm-parameter-value {
    leaf pm-parameter-value {
      type decimal64 {
        fraction-digits 7;
      }
      description "The PM Parameter value. The type Real allows the representation of e.g. either gauges or counters.";
    }
    leaf pm-parameter-unit {
      type string;
      description "The PM Parameter unit.";
    }
    description "PM metric value.";
  }
…
  grouping time-period {
    leaf value {
      type uint64;
      description "The specific value of the time period.";
    }
    leaf unit {
      type time-unit;
      description "The unit of measurement of the time period.";
    }
    description "Period of time.";
  }
  typedef time-unit {
    type enumeration {
      enum YEARS {
        description "none";
      }
      enum MONTHS {
        description "none";
      }
      enum DAYS {
        description "none";
      }
      enum HOURS {
        description "none";
      }
      enum MINUTES {
        description "none";
      }
      enum SECONDS {
        description "none";
      }
      enum MILLISECONDS {
        description "none";
      }
      enum MICROSECONDS {
        description "none";
      }
      enum NANOSECONDS {
        description "none";
      }
      enum PICOSECONDS {
        description "none";
      }
    }
    description "Units of measurement of the time.";
  }
```

Figure 25 YANG: pm-metric

For this structure a solution shall provide the following support:

- Conditional Mandatory:
    - pm-metric-info: Mandatory where the specific value is available and relevant.
    - threshold-configured-value: Mandatory where the threshold is configurable.
    - granularity-period: Mandatory where there is no oam-job related and the period is not obvious from the property name.

```
    grouping detector-info {
        leaf perceived-severity {
            type perceived-severity-type;
            description "The severity of the detected Condition.";
        }
        leaf service-affecting {
            type service-affecting;
            description "The impact on the service.";
        }
        leaf is-acknowledge {
            type boolean;
            description "Information on operator acknowledgement.";
        }
        leaf detector-category {
            type detector-category;
            description "The Detector (alarm) category, based on ITU-T X.733.";
        }
        description "(Legacy) information associated to a Condition (alarm).";
    }
…
    typedef perceived-severity-type {
        type enumeration {
            enum CRITICAL {
                description "ITU-T G.7710/X.733/M.3100: Indication for a service-affecting condition. Immediate corrective action is required.";
            }
            enum MAJOR {
                description "ITU-T G.7710/X.733/M.3100: Indication for a service-affecting condition. Urgent corrective action is required.";
            }
            enum MINOR {
                description "ITU-T G.7710/X.733/M.3100: Indication for a non-service-affecting condition. Corrective action should be taken in order
                   to prevent more serious fault.";
            }
            enum WARNING {
                description "ITU-T G.7710/X.733/M.3100: Indication for a potential or impending service-affecting fault. Further diagnosis should be
                   made.";
            }
            enum CLEARED {
                description "Included only for some possible backward compatibility purpose. It should not be used to assign a severity to a failure.
                    ITU-T G.7710: The severities 'cleared' and 'indeterminate' defined by [ITU-T X.733] are not included in Table 2, as it is assumed
                    that these are not to be used to assign a failure.";
            }
        }
…
    typedef service-affecting {
        type enumeration {
            enum SERVICE_AFFECTING {
                description "The service is affected by the detected Condition.";
            }
            enum NOT_SERVICE_AFFECTING {
                description "The service is not affected by the detected Condition.";
            }
            enum UNKNOWN {
                description "The impact on the service is unknown.";
            }
        }
        description "The possible impact on the service.";
    }
…
    typedef detector-category {
        type identityref {
            base DETECTOR_CATEGORY;
        }
        description "The Detector (alarm) category, based on ITU-T X.733.";
    }
…
    identity DETECTOR_CATEGORY {
        description "none";
    }
    identity DETECTOR_CATEGORY_EQUIPMENT {
        base DETECTOR_CATEGORY;
        description "none";
    }
    identity DETECTOR_CATEGORY_ENVIRONMENT {
        base DETECTOR_CATEGORY;
        description "none";
    }
    identity DETECTOR_CATEGORY_CONNECTIVITY {
        base DETECTOR_CATEGORY;
        description "none";
    }
    identity DETECTOR_CATEGORY_PROCESSING {
        base DETECTOR_CATEGORY;
        description "none";
    }
    identity DETECTOR_CATEGORY_SECURITY {
        base DETECTOR_CATEGORY;
        description "none";
    }
    identity DETECTOR_CATEGORY_UNDEFINED {
        base DETECTOR_CATEGORY;
        description "none";
    }
```

Figure 26 YANG: detector-info

For this structure a solution shall provide the following support:

- Conditional Mandatory:
    - perceived-severity: Mandatory where a client system requires this
        - CLEARED should not be used. A cleared alarm is represented by a DELETE/TOMBSTONE record pair.
        - Perceived-severity provides a simplistic valuation of the alarm. The relevance of this property is expected to reduce over time.
    - service-affecting: Mandatory where service impact is known. Default assumed to be UNKNOWN.
        - service-affecting provides a simplistic valuation of the alarm. The actual impact is determined by detailed assessment of resource topology and usage. The relevance of this property is expected to reduce over time.
    - is-acknowledged: Mandatory where there is acknowledgement information related to the alarm and this is required by the client.
        - Is-acknowledged relates to traditional per alarm manual operation. The relevance of this property is expected to reduce over time.
    - detector-category: Mandatory where category is required by the client.
        - detector-category is a relatively blunt categorization of an alarm where the alarm and its source dictate category (this property is derivable). The relevance of this property is expected to reduce over time.

```
    grouping simple-detector {
        leaf simple-detector-state {
            type simple-detector-state;
            description "The (simple) state of the Detector.
                The Detector state accounts for the time characteristics of the detected Condition.";
        }
        description "Information regarding the (simple) state of the Detector.";
    }
…
    typedef simple-detector-state {
        type identityref {
            base SIMPLE_DETECTOR_STATE;
        }
        description "The states of the detector.";
    }
…
    identity SIMPLE_DETECTOR_STATE {
        description "none";
    }
    identity SIMPLE_DETECTOR_STATE_ACTIVE {
        base SIMPLE_DETECTOR_STATE;
        description "The detector is indicating the operation of the monitored entity is not within acceptable bounds with respect to the specific
            condition measured.
            If INTERMITTENT is supported there may be a requirement for persisted unacceptable operation after a problem occurs before
            ACTIVE is declared. An alternative may be to declare INTERMITTENT.
            Where INTERMITTENT is supported, ACTIVE indicates the stable presence of a problem.";
    }
    identity SIMPLE_DETECTOR_STATE_CLEAR {
        base SIMPLE_DETECTOR_STATE;
        description "The detector is indicating the operation of the monitored entity is within acceptable bounds with respect to the specific
            condition measured.";
    }
    identity SIMPLE_DETECTOR_STATE_INTERMITTENT {
        base SIMPLE_DETECTOR_STATE;
        description "The detector is indicating the operation of the monitored entity is intermittently not within acceptable bounds with respect to
            the specific condition measured.
            INTERMITTENT support is optional. Where it is supported there may be a requirement for persisted unacceptable operation after a
            problem occurs before ACTIVE or INTERMITTENT is declared.";
    }
    identity SIMPLE_DETECTOR_STATE_FLEETING {
        base SIMPLE_DETECTOR_STATE;
        description "Event has a very short life (Active-Clear), hence is notified/streamed after its occurrence.";
    }
    identity SIMPLE_DETECTOR_STATE_ACTIVE_NO_EXPLICIT_CLEAR {
        base SIMPLE_DETECTOR_STATE;
        description "Same as Active, but an explicit transition to Clear is not foreseen.
            This e.g. applies to PM metrics which can only increase (counters), hence the 'clear' criteria is conventionally the end of a
            measurement period.";
    }
```

Figure 27 YANG: simple-detector

For this structure a solution shall provide the following support:

- Mandatory:
    - simple-detector-state: Where state depends upon underlying system capability
        - For alarms
            - ACTIVE and CLEAR (may be used in the DELETE record) are mandatory
            - INTERMITTENT and FLEETING depend upon underlying alarm processing and hence are conditional mandatory
        - For PM thresholds
            - ACTIVE and CLEAR (in DELETE record) used where there is an explicit clear
            - ACTIVE_NO_EXPLICIT_CLEAR used where there is no explicit clear.

Note that simple-detector provides a broader range of options than that provided by detector-state in TAPI 2.1.3.

4.5.3 alarm-condition-detector-detail from tapi-streaming.yang

In TAPI 2.1.3 alarm reporting was supported by an augmentation of detected-condition with alarm-condition-detector detail. This approach has been deprecated in TAPI 2.4 in favor of the approach detailed in tapi-fm in section 4.5.2 detected-condition from tapi-fm.yang on page 60. The TAPI 2.1.3 approach described in this section is still available for use but is not recommended.

The condition-detector can be augmented with properties related to alarms as follows:

```
grouping alarm-condition-detector-detail {
        leaf alarm-detector-state {
            type alarm-detector-state;
            config false;
        }
        container legacy-properties {
            config false;
            uses legacy-properties;
        }
}

…

grouping legacy-properties {
        leaf perceived-severity {
            type perceived-severity;
        }
        leaf service-affect {
            type service-affect;
        }
        leaf is-acknowledged {
            type boolean;
        }
        leaf-list additional-alarm-info {
            type string;
        }
    }
```

Figure 28 YANG: alarm-detector and legacy-properties (descriptions omitted)

For this structure, a solution shall provide the following support:

- Mandatory:
  - alarm-detector-state: indicates whether the alarm is ACTIVE, INTERMITTENT or CLEAR. This is the essential state of the alarm.
- Optional (see following text):
  - perceived-severity
  - service-affect
  - is-acknowledged

Many solutions and operational practices continue to use the historic schemes (see section 8.4.8 Traditional alarm reporting and legacy-properties on page 137). On that basis, the schemes are supported but relegated to optional. At this point in the evolution of control solutions legacy-properties are probably mandatory, however, it is anticipated that as control solutions advance the legacy-properties will become irrelevant.

The legacy-properties are:

- perceived-severity: A network device will provide an indication of importance for each alarm. This property indicates the importance. In some cases, the severity may change through the life of an active alarm.
- service-affect: Some network devices will indicate, from its very narrow viewpoint, whether service has been affected.
- is-acknowledged: Network devices offer a capability to acknowledge alarms (to stop the bells ringing). Often an EMS will offer a similar capability. This property reflects the current acknowledge state.
- additional-alarm-info: Often, alarms raised by network devices have additional information. This property can be used to convey this.

Although originating from network device control, the above properties may be supported by other device types.

4.6 Alarm Identifier and location

This section further clarifies the rationale for the choices of mandatory and optional identifiers and provides further requirements.

- The identifier of the alarm is essentially the identifier of the detector (detector-native-id). This shall be based upon native device values to ensure consistency over time and across resilient systems etc.
    - The detector is at a functional location in the solution and detects a particular condition at that location. It shall be identified by these two key properties. i.e., functional location and condition.
    - The detector is long lived and may emit many active and clear events through its life.
- The alarm will normally be reported via TAPI against a valid TAPI entity and hence the overall location of the detector will include the identifier of the TAPI entity.
    - The report shall also include information in the location from the device in device terminology to enable the relating of information acquired directly at the device with information acquired at a higher system.
    - The TAPI model allows for alarms without a full TAPI resource id, although this should be a rare case.
- Where the detector relates to a thing that is not fully modeled in TAPI, e.g., a power supply, then:
    - The alarm shall be reported against a containing TAPI entity.
    - The identifier of that detector shall include a meaningful index that include interpretable sub-structuring to describe the position of the detector (again based upon device values)

4.7 Alarm TOMBSTONE record behavior

As noted earlier, the alarm DELETE record (representing the alarm clear) shall be followed immediately by a TOMBSTONE record. As also noted, the deletion of an alarm detector, where the alarm was active prior to the deletion, shall cause at least the logging of a TOMBSTONE record. Allowing for compaction delay:

1. The DELETE/TOMBSTONE record pair (representing the alarm clear) shall cause the compaction process to remove the corresponding active alarm.
2. The TOMBSTONE record shall also cause the compaction process to remove the DELETE record (that immediately precedes it)
3. The TOMBSTONE record shall eventually be removed as a result of tombstone-retention duration being reached.

4.8 Time

Whilst the timestamp for the alarm is not a provider responsibility (other than for provider generated alarms, e.g., disc full), there is an expectation of correct behavior.

A versatile approximate-data-and-time structure has been used for the event-time-stamp to allow representation of inaccuracies.

The time stamp for an alarm shall be as follows:

- The timestamp from the ultimate source, representing the time of occurrence of the event that triggered the raising or clearing of the alarm, shall be preserved through the entire passage of the alarm information through the source device, and any chain of controllers, and presented in the appropriate field in the TAPI log-record-body.
    - Where the source does not provide a time stamp, a timestamp shall be added by a provider as the alarm progresses through the solution. This timestamp shall be marked with a spread value BEFORE
- In general, it is extremely important to accurately capture the leading edge of a problem. Hence, for each detector, the time of first occurrence of an active alarm after a "long" period of clear is the most significant time.
    - The time of clearing can be less precise.
- The log record also has a log-append-time-stamp.
- Ideally, the time of occurrence of an alarm should be the time of the entry into the confirmed assessment as to whether an alarm has occurred (and not the time of exit from that assessment).
    - Likewise, the time of clearing of the alarm should be the time of entry into the confirmed assessment as to whether an alarm has cleared.

4.9 Detected Condition normalization

The alarm detected condition shall be presented:

- Minimally: In native form, i.e., as provided by the Device
- Additionally: In normalized form, i.e., complying with the list of standard alarm/TCA types in the normative [RIA AT]

An alternative is to provide translation metadata to enable normalization from the native form at the client. This metadata can be provided separately from the alarm stream and related to each detector with a relevant mapping. There is not standard expression for translation metadata.

4.10 Meaningful detection (device or any other source)

Under certain circumstances a detected condition may:

- Become meaningless, e.g., when a remote function is disabled, in which case the a TOMBSTONE record for alarm shall be appended to the log.
    - This may require function disable action to be taken on the local system.
- Have inverted meaning, e.g., when signal should not be present on a port, in which case, if there is a "signal not present" alarm active, then a TOMBSTONE record for that alarm should be appended to the log and if there is a signal present a "signal present" alarm should be raised.
    - This may result from some local action on the entities supporting signal flow

- Essentially, the "signal not present" condition detection becomes meaningless, and a "signal present" condition detection becomes meaningful.

5 Reporting Change Only

Whilst the WHOLE_ENTITY method provides an easy and consistent approach to reporting creation and change, where the whole entity is always reported, it is also potentially inefficient. Some entities are very large and if there is a property in a very large entity, perhaps a Boolean, that changes often, then the whole large entity must be reported to convey the Boolean change.

It can be argued that each potentially fast-changing property (e.g., operational-state) should be extracted from the entity that it applies to and reported separately referencing the entity (using the same approach as is used for alarms), this would lead to significant changes to the model and as, essentially, any property could change rapidly this may require a complete refactoring of the model.

To maintain the model whilst reducing the change reporting overhead an approach of reporting only change can be used. This does benefit from some adjustments to details of the model, but none to the tree structure and positioning of properties.

Reporting a change fundamentally requires only the skeleton of the structure of the tree that leads to the property (leaf) that has changed along with the new state of the property. Clearly, this is somewhat simplified as some changes are complex. But regardless, the change is localized to a small part of a large instance and only that part need be conveyed. Each changed property would appear at the appropriate place in the mainly empty tree structure. The complexities of change to ordered list structures and removal of items from a list will be dealt with later in this section.

It is important that the reporting of change is conformant with the YANG model. Clearly, to be conformant, all mandatory properties must be reported, even if they have not changed. Ideally, from this perspective, there would be no properties that are mandatory in the YANG model (other than perhaps the entity id and list key which are fundamental to model structure (structural properties)). On the surface it appears that some properties are mandatory under all network use case circumstances, however, closer inspection brings the realization that no (non-structural) properties are mandatory under all circumstances. For example, it is always possible that under some circumstances the property value is not available[65].

Considering the above, it is likely that each property in the TAPI YANG model will become optional and that conditions for its inclusion will be enhanced in TR-547 to ensure that it is clear under which conditions the property is “mandatory”. Full adjustment to a “YANG optional TR-547 conditional mandatory” approach may take several releases.

The following sections describe the application of CHANGE_ONLY streaming in the above context.

5.1 Behaviors and Capabilities

Whilst the TAPI model has allowed for reporting of CHANGE_ONLY, the approach to streaming only changes, as opposed to WHOLE_ENTITY, has not been described in previous versions of this document. TAPI 2.4.0 offered some improvements to some structures of the streaming model that provide greater clarity when reporting the availability of the CHANGE_ONLY streaming method.

The CHANGE_ONLY method takes advantage of conditional mandatory properties in parts of TAPI 2.4 (as conditional mandatory essentially allows the property to be omitted if it has not changed) and where necessary assumes an implicit conditional mandatory, even where properties are currently declared as mandatory in TR-547. The necessary additional condition is that in the CHANGE_ONLY stream, a property is only mandatorily present when it has changes since the last report of the property. A property may be mandatory in a GET under the prevailing circumstances, but it may still be omitted from the stream record as it has not changed.

[65] This is a complex area which include the consideration of partially visible properties and weightings etc. This is covered in work being initiated in IETF under [IETF-MOBO].

The CHANGE_ONLY report should be YANG model compliant so where there is an element, branch of a tree etc. that is mandatory in YANG, it should be included in the stream data. As noted, it is the expectation that all properties will become optional in the YANG and conditional (or optional) in TR-547. At this point in the development, it is allowed for a non-mandatory property that has not changed to be included in the report if this is considered in some way beneficial. Clearly, parent address, entity ids and list keys are mandatory.

The following section briefly describes the CHANGE_ONLY streaming approach.

5.2 CHANGE_ONLY Streaming

Clearly, for a statement of change to be interpreted by the client it is necessary that it has a complete view of the state before the change. The effect of the compacted log behavior needs to be considered carefully. As discussed, there are essentially three areas in the compacted log:

- Non-compacted: where all recent records (including the most recent) are present including potentially many records about the same entity
- Compacted: where only the most recent full record of state of an entity is guaranteed to be present, but all deletes, represented by TOMBSTONE records will also be present so maintaining alignment can be assured.
- Older than tombstone-retention: where only the most recent records of state of an entity are guaranteed to be present and there are no records of delete (i.e., no TOMBSTONE records)

The following discussion assumes that the client is well behaved and has progressed correctly from the oldest record through the alignment process to get to a particular point in the log/stream. Where the client is receiving records from the

- Non-compacted area of the log
    - The client is aligned with the state of the system at the time of the most recently processed record
    - All changes that have taken place since that point in time are recorded in the log from that point up to the latest change
    - A CHANGE_ONLY record would provide enough detail to allow the client to maintain that alignment
- Compacted area of the log
    - The client is aligned with the state of the system at the time of the most recently processed record
    - Some changes that have taken place since that point in time have been compacted away such that the next record may represent several changes
    - The compaction process does not update any records, it simply deletes “redundant” records
    - A WHOLE_ENTITY record needs to be sent to the client (if the log contained CHANGE_ONLY records, changes would be lost)
- Area where records are older than -retention
    - The client will need to realign as usual and will need all records to be WHOLE_ENTITY

On that basis, to achieve CHANGE_ONLY streaming, the provider must log the whole entity, to support the WHOLE_ENTITY reporting when required, with some identification of the parts that have changed, to support CHANGE_ONLY reporting when possible.

The provider is responsible for making the choice of which record type to send:

- **CHANGE** (change only)
    - Sent when reading from the non-compacted part of the log, where the client can be assumed to have an up-to-date view of the entity and a change is to be reported

- **UPDATE** (whole entity)
  - Used where the provider is distinguishing between the case where the entity existed before and the case where the entity did not exist
  - Sent when reading from the compacted part of the log, where the provider knows the entity existed and wants to report a change but cannot use CHANGE as the client may not be aligned with the state before the change
  - The record includes all details of the entity
  - Considering compaction, the client may interpret an UPDATE as a CREATE when the CREATE has been compacted
    - As a simplification the provider may send a CREATE_UPDATE
- **CREATE** (whole entity)
  - Used where the provider is distinguishing between the case where the entity existed before and the case where the entity did not exist
  - Sent where provider knows the entity has not existed before or where the provider was not previously aware of the entity
  - Sent regardless of where in the log the client is receiving from
  - The record includes all details of the entity
- **CREATE_UPDATE** (whole entity)
  - Used where the provider is not distinguishing between the case where the entity existed before and the case where the entity did not exist
  - Sent when reading from the compacted part of the log,
  - The record includes all details of the entity
- **DELETE**
  - Always sent as a DELETE record
  - The record includes relevant ids and may include other entity details
- **TOMBSTONE**
  - Logged along with a DELETE record
  - Always sent as a TOMBSTONE record

The figure below summarizes the behavior required at the provider.

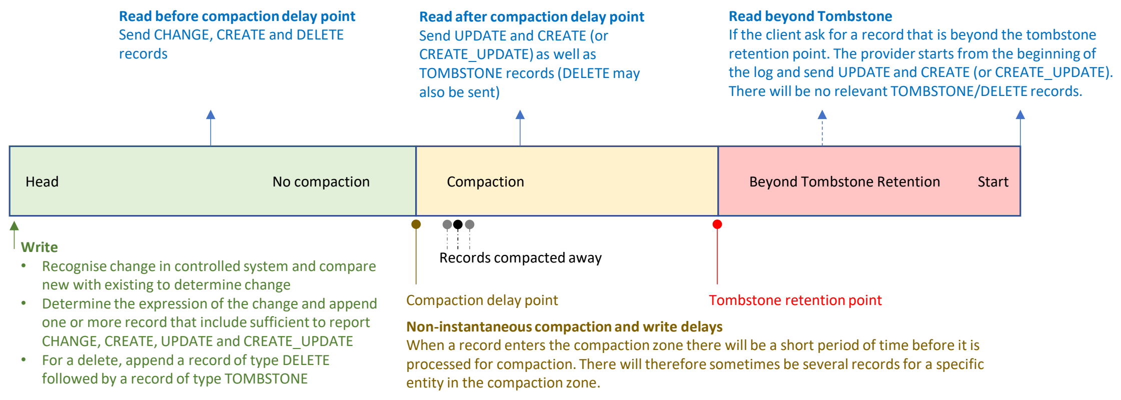

Read before compaction delay point
Send CHANGE, CREATE and DELETE records
Read after compaction delay point
Send UPDATE and CREATE (or CREATE_UPDATE) as well as TOMBSTONE records (DELETE may also be sent)
Read beyond Tombstone
If the client ask for a record that is beyond the tombstone retention point. The provider starts from the beginning of the log and send UPDATE and CREATE (or CREATE_UPDATE). There will be no relevant TOMBSTONE/DELETE records.
Head
No compaction
Compaction
Beyond Tombstone Retention
Start
Write
Recognise change in controlled system and compare new with existing to determine change
Determine the expression of the change and append one or more record that include sufficient to report CHANGE, CREATE, UPDATE and CREATE_UPDATE
For a delete, append a record of type DELETE followed by a record of type TOMBSTONE
Records compacted away
Compaction delay point
Tombstone retention point
Non-instantaneous compaction and write delays
When a record enters the compaction zone there will be a short period of time before it is processed for compaction. There will therefore sometimes be several records for a specific entity in the compaction zone.

Figure 29 Delta stream compacted log behavior

Similar behavior applies to the emulated compacted log stream solution and a stream solution with zero length compaction zone.

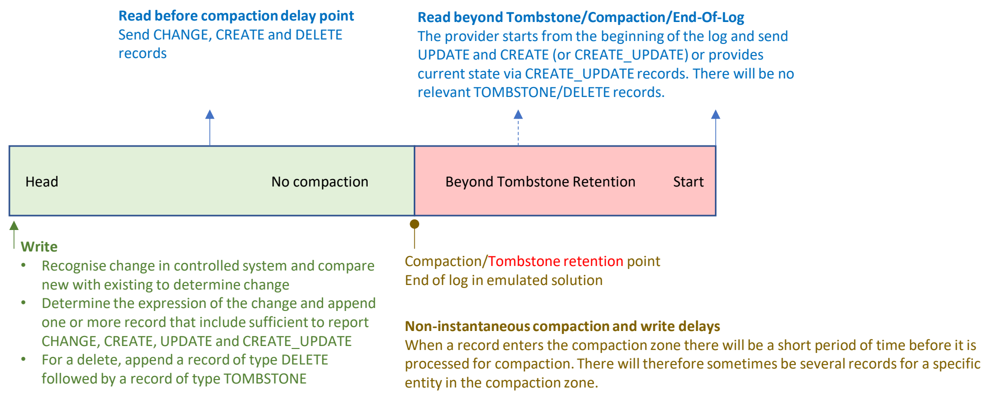

Read before compaction delay point
Send CHANGE, CREATE and DELETE records
Read beyond Tombstone/Compaction/End-Of-Log
The provider starts from the beginning of the log and send UPDATE and CREATE (or CREATE_UPDATE) or provides current state via CREATE_UPDATE records. There will be no relevant TOMBSTONE/DELETE records.
Head
No compaction
Beyond Tombstone Retention
Start
Write
Recognise change in controlled system and compare new with existing to determine change
Determine the expression of the change and append one or more record that include sufficient to report CHANGE, CREATE, UPDATE and CREATE_UPDATE
For a delete, append a record of type DELETE followed by a record of type TOMBSTONE
Compaction/Tombstone retention point
End of log in emulated solution
Non-instantaneous compaction and write delays
When a record enters the compaction zone there will be a short period of time before it is processed for compaction. There will therefore sometimes be several records for a specific entity in the compaction zone.

Figure 30 Delta stream compacted log emulation behavior

5.3 Client view of sequence number

The log record has a sequence number. Compaction causes the sequence number of adjacent records to not to be contiguously sequential. For CHANGE_ONLY, assuming transactional writes, a client in the “No compaction” zone can expect sequence number increments of 1.

5.4 Client processing considerations

When the client receives a record for an entity it should check for the presence of that entity in its repository. If the entity is present and the record is:

- **CHANGE** (change only)
  - If the entity is present in the client’s repository, the client should simply update with the properties provided (some of which may not have changed)
  - If the entity is not present in the client’s repository, the client should initiate a realignment by initiating a request with no token.
- **UPDATE** (whole entity)
  - If the entity is present in the client’s repository, the client can simply replace the old entity values with the new entity values.
  - If the entity is not present in the client’s repository, the client should create the entity.
- **CREATE** (whole entity)
  - If the entity is present in the client’s repository, the client can simply replace the old entity values with the new entity values. The client may record this as an unexpected create.
  - If the entity is not present in the client’s repository, the client should create the entity.
- **CREATE_UPDATE** (whole entity)
  - If the entity is present in the client’s repository, the client can simply replace the old entity values with the new entity values.
  - If the entity is not present in the client’s repository, the client should create the entity.
- **DELETE / TOMBSTONE**
  - If the entity is present in the client’s repository, the corresponding entity should be deleted from the client repository.
  - If the entity is not present in the client’s repository, no action should be taken.

On arrival of a record related to a particular entity, the client may want to determine the specific change in that entity by comparison of the new record with their existing record for that entity.

The client can be sure that once the first CHANGE notification has been received, the provider is streaming from the non-compacted head. However, as the stream flow may be delayed at any moment, and hence the stream may slip back into the compaction zone. On that basis the client should deal with any mix of CHANGE and UPDATE messages.

5.5 Dealing with changes in lists

The CHANGE record can deal with a complex change by reporting an entire sub-tree. For example, if:

- an item is inserted in or removed from a list, the entire list may be sent
- a structure has parts rearranged, the entire structure that the change is in may be sent

Note that this may be such that a whole substructure is sent even though only one property in that substructure has changed.

Beyond this release a more sophisticated approach may be possible (see [RFC6902]) to deal with:

- Add: specific value in specific field in list
- Remove: specific field in list
- Insert: add field with value in particular place in list between two existing list entries
- Append: add field with value to the end of the list
- Replace: overwrite existing value of specific field with new value
- Move: take field with value from one place in the list and place elsewhere in the list
- Swap: Take two fields with values and swap their positions
- Copy: take a field with value and apply it to an existing list using Add/Insert/Append

5.6 Future opportunities for stream tuning

- No readers
    - compaction-delay is zero
    - tombstone-retention zero
- First reader arrives
    - compaction-delay, e.g., 30 minutes
    - tombstone-retention, e.g., 4 hours
- Last reader was connected longer than tombstone retention ago
    - Return to “no readers” state
- Reader dropped for a period close to tombstone retention time
    - Can extend tombstone-retention if requested

6 Streaming of measurement data for performance monitoring

6.1 Reporting measurement data

Analysis of network characteristics enables a control solution to identify issues and take necessary action to maximize the support for services provided. Measurement of a subset of network characteristics of a particular type, where counting occurrences or examining levels, is considered as performance measurement

6.1.1 Volume of data

In a network there can be vast number of enabled performance measurements (many per service etc.) where the measurements are sampled on a regular rapid basis and hence there is a massive quantity of data to be collected ongoing. The data can be used by a range of applications some of which require near real time visibility.

> **Requirement:** The controller, monitoring the performance measures, shall provide performance data in an efficient fashion with little delay to a client where performance data, both recent and historic, will be analyzed.

The approach in TAPI 2.4.1 to reporting PMs relies on a traditional method of getting current and history data in the context of an oam-job. Notification of changes is currently not fully defined (as there are many unresolved behaviors, for example one property in current data is an incrementing timer. Clearly, change to this must not be notified). In addition, the current data is known at the counters in the device but clearly, it is not reasonable to have ongoing knowledge of current data at the controller!

The response to the GET method in TAPI 2.4.1 is encoded in Restconf JSON and is verbose in that invariant fields are present. The current method cannot deal well with suspect and partial results.

There are several distinct measurement property characteristics:

- A value that tends to be mainly constant but that may increase or decrease over time (e.g., power)
- A value that relates to a counter that can remain constant but that normally increment, potentially rapidly, over time (e.g., a packet counter)
- A value that changes relatively often initially, but that gradually reduces in rate of change (e.g., for property of a particular contained characteristic, an average, a max, a min)
- Etc.

It may be required to acquire

1. Completed measurements where there is a defined measurement period and where at the end of the period the values from the current data are captured, potentially reported, and potentially maintained in history data (e.g., an error counter)
2. Periodic views of the values of continuously changing measures (e.g., power)
3. Current values at some point in an ongoing measurement (which is maintained in current data)
4. Values at a specific time or between two specific times where the timing is not aligned with the normal measurement period (where there may be a requirement to calculate average, peak etc. in the period).
5. A measurement that takes a long time to process and hence will be significantly delayed.

Note that this document focusses on (1) and (2) above. Items (3), (4) and (5) are for further study and relate to the snapshot stream method touched on elsewhere.

> **Requirement:** The solution shall support the reporting a stream of "periodic completed measurements" (where there is a defined measurement period over which a count is accumulated)

and “periodic views” (where there is a sampling of an ongoing value). Note that the type of measurement will be sufficient to allow interpretation of the report (and it is not necessary to indicate whether the report is a completed measurement or a view).

Considering the need for near real time visibility, a notification/streaming method would appear more appropriate than a GET method as the stream of data can be initiated by the provider[66] at the moment the data is available.

6.1.2 Responsibility for measurement data

A performance measurement is defined to be taken over a specific period. In the TAPI 2.4.1 model, the current-data provides a view of the ongoing measurement, and the history-data provides a view of previously completed measurement periods. At the completion of the measurement period, the current-data values, i.e., the completed measurement, are moved to history-data and, in many cases, current data is reset so as to start accumulating the measurement for the new period. It is the completed measurement that is required to be streamed with urgency to enable the near-real-time applications.

Assumption: The client is responsible for maintaining the history for all performance measurements in the network and for carrying out analysis on that performance data.

Requirement: The controller, monitoring the performance measures, shall provide the client system with performance data related to all enabled performance monitor points on functions expressed via the TAPI model in the context.

It is assumed that the client system maintains a record of all streamed “completed” measurements. As for the tapi-streaming solution for compacted logs, the performance data streaming solution allows the client to recover from temporary comms loss by retaining a short log of completed measurements.

The performance data streaming solution does not stream:

- current-data, as that is continually changing and is normally not directly available to the control system (as it is maintained by the device and is only available on demand).
- history-data, as the controller, monitoring the performance measures streams and maintains a log of recent completed measurements and it is assumed to be the client responsible to keep the long-term view of history.

6.1.3 Efficient streaming

Considering the vast number of measurements, it is important to maximize the information density in the data.

Requirement: The solution shall:

- Not send invariant fields
- Define fields such that they are smallest footprint (i.e., leading to the shortest message) when the most common report is to be made
- Using an efficient enumerated binary coding method
- Allowing multiple measures that share properties (e.g., time of capture) to be reported in a block with the shared property only stated once

The method defined here takes advantage of techniques related to the above list to achieve a high information density.

It is assumed that any measurement is made in the context of an oam-job.

[66] The word provider is being used as a shorthand for stream provider throughout this document.

Note that there are further opportunities for improvements of information density that are not covered in this release such as suppressing values that can predict:

- Zero
- Same value as last time
- Within expected range
- Following the expected pattern

6.1.4 Streaming of measurement data

TAPI streaming has been defined to support a variety of different reporting formats and encoding strategies. The defined form in previous releases of this document was based on a compacted log. The compacted log strategy is not appropriate for streaming of performance data as each record is unique and independent, so there is no rationale for compaction (other than age, which is then most efficiently implemented using simple truncation). See discussed in section 3.8 Streaming approach and log strategy on page 27 for more details.

As noted in the previous section, performance data tends to be measured on some periodic basis and provided in a report of absolute value, where interpretation of that absolute value depends upon the:

- Time the detector was sampled (i.e., the time the measurement was made)
- Specific measurement method
- Location of the detector (function, termination etc. and hence the specific flow being measured)
- Units of the measurement
- Time period over which the measurement was made
- Job which initiated and is coordinating the measurement

It may also depend upon knowledge of the:

- Previous value (where the counter is not reset when sampled)
- Reporting strategy
- Completeness of measure

The considerations above are covered by requirements in the sections below.

6.1.5 Dealing with Comms failures

Considering the importance of some performance data it is vital that the solution allows a client to recover from a short loss of comms without loss of performance data.

> **Requirement:** For a client that has subscribed for performance data and that is receiving that data successfully, the solution shall allow that client to reconnect after a communications failure such that no performance data is lost. Note that some data may be duplicated.

The provider will therefore need to support a short buffer of recent performance data. This is not a full history. The approach to providing this short buffer is an implementation choice.

Note that the definition of “short” will depend partly on the robustness of the communications environment and the likely down time. Ability to retain the past hour of performance data reports would seem sufficient for many applications.

The gNMI specification allows for extensions and has a defined extension that provides a mechanism to subscribe to “history” [GNMI HIST]. This mechanism allows for a start and end time. Only the start time will be used here. The start time will be interpreted as the time the record was created/logged.

6.1.6 Data Sources

Performance data may be recorded against a CEP, a MIP or a MEP. The identifiers of the CEP, MIP and MEP are UUIDs. The source of the PM data will provide an id that is not the UUID of the CEP/MIP/MEP. Adding the UUID of the TAPI entity (such as CEP, MIP, MEP) requires the provider system to have to look up the entity using the id used by the source. This is a relatively expensive operation especially considering the volume.

> **Requirement:** The solution shall enable a provider to convey interpretable performance data without the need to look up the TAPI entity UUID.

The solution to this requirement is to record the id that will be used by the source of the date in the corresponding TAPI entity in one of the name fields. The decoding of these ids can be carried out by the client when it chooses to analyze the data.

The property “abstract-resource-ref” supports this strategy (see also section 8.6 Appendix – Measurement data – Referencing resources on page 140).

6.1.7 Time

The most significant time is the moment the measure was sampled. It is also important, for some measures, to understand the period over which the measure was taken.

> **Requirement :** The solution shall make available to the client information on the time the measurement was sampled and the interval from the previous sample

> **Requirement:** The solution shall make available the time when the measurement was started if that is different from the time that can be calculated using the time the measurement was sampled and the interval from the previous sample.

6.1.8 Problems with a measurement

It is possible for a measurement device to not correctly record a value:

- The device is reset during a measurement interval so only part of the interval is measured
- The measurement device operation is degraded so that a measurement is not accurate

> **Requirement:** The solution shall allow reporting of a value for a measurement but shall also allow reporting of issues with that measurement including covering the cases where the measurement is not accurate or missing completely.

6.1.9 Framing a measure

A measurement device may be activated and maybe deactivated. There may be ongoing measurement reporting such that the measurement from the newly activated devices is added to a stream of ongoing measurements and such that measurements from a newly deactivated device are no longer present in the stream. It is possible that reporting of a measurement is significantly delayed.

> **Requirement:** The solution shall provide an opportunity to indicate that a measurement is the temporal first or the temporal final measurement for a particular detector/measure in a stream of ongoing measurements.

> **Requirement:** The solution shall provide an opportunity to indicate that a measurement report is for a measurement taken some while ago (and hence is delayed). This indication will be in addition to the time of the measurement.

6.1.10 Relative measurements

Where measurements are in some way relative to a previous measurement it is necessary for the provider to report an absolute baseline against which a stream of relative measurements can be assessed. NOTE: This is not required in this release but is provided in preparation for future capabilities.

> **Requirement:** The solution shall provide an opportunity for the provider to indicate that a measurement is a baseline. The baseline will be required:
>
> - at the start of a measurement sequence
> - on subscription by the client
> - at some intermediate point when requested by the client

Note that there currently is no subscription method defined to acquire baseline reports. This will be covered in a future release.

6.1.11 Qualifying the measurement

Some measurements require further qualification such as direction of the measurement, the end of the measurement, detailed name of the measurement etc.

> **Requirement:** The solution shall provide support for necessary measurement qualifications to highlight direction when not "obvious" from the measure and highlight point of measurement (e.g., far end).

6.2 The solution

6.2.1 Solution technology

Considering the requirements above, a gNMI stream [GNMI-SPEC] with protobuf encoding [GNMI-PROTO] was chosen as the solution technology. The following subsections describe this approach.

6.2.2 Discover supported and available streams

Where performance data streaming via gNMI/Protobuf is supported, streams will be declared that have supported-stream-type details with:

- stream-type-content: measurement-details
- log-record-strategy: WHOLE_ENTITY
- record-trigger: PERIODIC
- log-storage-strategy: TRUNCATED
- record-retention: This is in terms of minutes and clarifies the truncation statement. The specific setting for record-retention is an implementation choice that depends upon deployment parameters etc.
- allowed-connection-protocol: GNMI
- encoding-format: PROTOBUF
- information-record-strategy: Not specified at this stage and is not required to be used, but can be used in a proprietary mode to allow suppression, value expectation and dither reduction. This will be defined in a future release

An implementation conforming to this specification MUST support the supported-stream-type defined above.

6.2.3 TAPI Model

The stream reports completed measurements. The completed measurement is not represented by a formal class in the tree. The completed measurement is a fleeting transient instance that exists between the capture of current-data and the creation of the history-data. A class representing this structure is essentially positioned under the oam-job that controls the measurement, but it is not retrievable as it is essentially created and deleted instantaneously. It would also not fit the normal streaming/notification pattern as reporting deletion would be redundant.

In the streaming model, the entity is called measurement-details and its path includes the related oam-job[67] such that it has essentially been placed in the tree under oam-job. The streaming of measurement-details can be considered as the reporting of both the creation and deletion of the instance in one report.

In the defined performance data streaming model, described below, it is possible to collect many measurements in one measurement-details structure. The current structure allows for the reporting of multiple measures which all belong to the same oam-job[68]. Where measures that belong to several jobs are required to be streamed in a single record, a new oam-job instance would be required to do the collection, i.e., there can be no more than one oam-job defined that details the collection to be streamed in one record.

Once an appropriate oam-job is defined for the maximal collection, the measure of the job may be reported separately if necessary/desired.

The solution defined here allows for reporting against entities other than oam-job as well as reporting with no path detail (i.e., considered directly under the context)[69]. These options apply where the references in the report are sufficient to uniquely identify the measurement within the defined path.

6.2.4 Transfer technologies

Whilst there are several technologies that are oriented towards efficient transfer of large volumes of information where that information has the characteristics of performance measurements. In the photonic network environment, the most prevalent technology is the Protobuf encoded gNMI Streaming.

On that basis a TAPI stream that takes advantage of gNMI and Protobuf has been defined. That definition is explained in the following sections.

6.3 The gNMI Stream

The gNMI Stream [GNMI] supports a subscription strategy that allows the client to select which entities are streamed. The measurement data is assumed to normally be positioned directly under the oam-job. As a consequence, the subscription selection opportunity allows filtering on a per job basis. In the TAPI application it is expected that minimal use will be made of the subscription selection opportunities as the context defines what the client is interested in and hence all enabled performance measures are of interest.

The gNMI Stream allows for extension with data not defined in the gNMI models. The TAPI solution takes advantage of this extension opportunity.

[67] Assuming that the measurement is essentially controlled by an oam job. The path simply needs the oam job uuid and need not include the uuid of the context as that provides no relevant information for path resolution, there being only one context.
[68] If the path includes the oam job, see following.
[69] It is understood that it is necessary for some tooling to have an explicit path in a stream record. If this is the case, the context with its uuid should be sufficient.

6.3.1 Available streams

To inform the client of the opportunity to receive information streamed through a gNMI stream the TAPI provider uses the standard method described in section 3.7 Supported and available streams on page16.

The connection-address should be provided in the format grpc://<host-fqdn>:<port>[70].

6.3.2 YANG and Protobuf

The following section focusses on the core YANG and Protobuf structure. The key class in the PM report is qualified-measurement-common. The UML model (see section 8.2.2 The model of gNMI protobuf on page 131) defines the protobuf enumerations.

6.3.2.1 The YANG schema

This section highlights the key structures in YANG schema in the form of a YANG tree. The explanation of each property is provided in the documentation of the YANG (see section 8.8 Appendix – Measurement YANG on page 143**Error! Reference source not found.**).

[70] The specific client behaviour will depend upon the libraries etc. used by the client.

```
module: tapi-gnmi-streaming
  augment /tapi-streaming:stream-record:
    +--ro subscribe-response
       +--ro notification
          +--ro timestamp?   uint64
          +--ro prefix
          |  +--ro origin?   string
          |  +--ro elem* []
          |  |  +--ro name?   string
          |  |  +--ro key
          |  |     +--ro value-name?   string
          |  |     +--ro value?        string
          |  +--ro target?   string
          +--ro update* []
             +--ro path
             |  +--ro origin?   string
             |  +--ro elem* []
             |  |  +--ro name?   string
             |  |  +--ro key
             |  |     +--ro value-name?   string
             |  |     +--ro value?        string
             |  +--ro target?   string
             +--ro val
                +--ro proto-bytes
                   +--ro measurement-details
                      +--ro time-measurement-was-sampled?       uint64
                      +--ro qualified-measured-value
                      |  +--ro value?                       decimal64
                      |  +--ro value-qualifier?             value-qualifier
                      |  +--ro units?                       string
                      |  +--ro qualified-value-name?         string
                      |  +--ro qualified-location-name?      string
                      |  +--ro qualified-measurement-type?   string
                      +--ro native-resource-id?                 string
                      +--ro sample-qualifier?                   sample-qualifier
                      +--ro relative-position-of-measurement?   relative-position
                      +--ro direction-of-measured-signal?       direction-of-measured-signal
                      +--ro sample-interval?                    uint64
                      +--ro abstract-resource-ref?              string
                      +--ro native-measurement-type?            string
                      +--ro normalized-measurement-type?        normalized-measurement-type
                      +--ro qualified-measurement* [list-index]
                      |  +--ro time-measurement-was-sampled?       uint64
                      |  +--ro qualified-measured-value
                      |  |  +--ro value?                       decimal64
                      |  |  +--ro value-qualifier?             value-qualifier
                      |  |  +--ro units?                       string
                      |  |  +--ro qualified-value-name?         string
                      |  |  +--ro qualified-location-name?      string
                      |  |  +--ro qualified-measurement-type?   string
                      |  +--ro native-resource-id?                 string
                      |  +--ro sample-qualifier?                   sample-qualifier
                      |  +--ro relative-position-of-measurement?   relative-position
                      |  +--ro direction-of-measured-signal?       direction-of-measured-signal
                      |  +--ro sample-interval?                    uint64
                      |  +--ro abstract-resource-ref?              string
                      |  +--ro native-measurement-type?            string
                      |  +--ro normalized-measurement-type?        normalized-measurement-type
                      |  +--ro list-index                          uint64
                      |  +--ro connection-end-point
                      |  |  +--ro topology-uuid?               -> /tapi-common:context/tapi-topology:topology-context/topology/uuid
                      |  |  +--ro node-uuid?                   -> /tapi-common:context/tapi-topology:topology-context/topology/node/uuid
                      |  |  +--ro node-edge-point-uuid?        -> /tapi-common:context/tapi-topology:topology-context/topology/node/owned-node-edge-point/uuid
                      |  |  +--ro connection-end-point-uuid?   -> /tapi-common:context/tapi-topology:topology-context/topology/node/owned-node-edge-point/tapi-
                      |  |                                        connectivity:cep-list/connection-end-point/uuid
                      |  +--ro maintenance-intermediate-point
                      |  |  +--ro meg-uuid?       -> /tapi-common:context/tapi-oam:oam-context/meg/uuid
                      |  |  +--ro mip-local-id?   -> /tapi-common:context/tapi-oam:oam-context/meg/mip/local-id
                      |  +--ro maintenance-end-point
                      |  |  +--ro meg-uuid?       -> /tapi-common:context/tapi-oam:oam-context/meg/uuid
                      |  |  +--ro mep-local-id?   -> /tapi-common:context/tapi-oam:oam-context/meg/mep/local-id
                      |  +--ro measurement-start-time?             uint64
                      |  +--ro qualified-measured-value-set* []
                      |  |  +--ro value?                       decimal64
                      |  |  +--ro value-qualifier?             value-qualifier
                      |  |  +--ro units?                       string
                      |  |  +--ro qualified-value-name?         string
                      |  |  +--ro qualified-location-name?      string
                      |  |  +--ro qualified-measurement-type?   string
                      |  +--ro additional-info* [value-name]
                      |     +--ro value-name    string
                      |     +--ro value?        string
                      +--ro connection-end-point
                      |  +--ro topology-uuid?               -> /tapi-common:context/tapi-topology:topology-context/topology/uuid
                      |  +--ro node-uuid?                   -> /tapi-common:context/tapi-topology:topology-context/topology/node/uuid
                      |  +--ro node-edge-point-uuid?        -> /tapi-common:context/tapi-topology:topology-context/topology/node/owned-node-edge-point/uuid
                      |  +--ro connection-end-point-uuid?   -> /tapi-common:context/tapi-topology:topology-context/topology/node/owned-node-edge-point/tapi-
                      |                                        connectivity:cep-list/connection-end-point/uuid
                      +--ro maintenance-intermediate-point
                      |  +--ro meg-uuid?       -> /tapi-common:context/tapi-oam:oam-context/meg/uuid
                      |  +--ro mip-local-id?   -> /tapi-common:context/tapi-oam:oam-context/meg/mip/local-id
                      +--ro maintenance-end-point
                      |  +--ro meg-uuid?       -> /tapi-common:context/tapi-oam:oam-context/meg/uuid
                      |  +--ro mep-local-id?   -> /tapi-common:context/tapi-oam:oam-context/meg/mep/local-id
                      +--ro measurement-start-time?             uint64
                      +--ro qualified-measured-value-set* []
                      |  +--ro value?                       decimal64
                      |  +--ro value-qualifier?             value-qualifier
                      |  +--ro units?                       string
                      |  +--ro qualified-value-name?         string
                      |  +--ro qualified-location-name?      string
                      |  +--ro qualified-measurement-type?   string
                      +--ro additional-info* [value-name]
                         +--ro value-name    string
                         +--ro value?        string
```

Figure 31 YANG tree

6.3.2.2 The protobuf structure

The protobuf structure that is used to enhance the standard protobuf stream in place of proto-bytes as shown in the YANG above is available at [TAPI PROTO].

6.3.3 Contents of path

As discussed above, the path is expected to normally identify the oam-job that controls the collection. The path can have no content[71], essentially placing the measurement in the context if the measurement has sufficient content to be uniquely identifiable.

6.3.4 Subscription

The gNMI specification [GNMI-SPEC] provides a full description of the subscription approaches available and the associated behavior.

The TAPI solution only utilizes a small subset of the specification and only supports one stream per port in this release. The following list explains what is used and what not in the SubscribeRequest:

- `subscribe`: provides a single entry `SubscriptionList`
  - `prefix:` The path is everything in the context hence only the target needs to be specified
    - `origin:` omitted (disambiguates the path)
    - `elem:` omitted (no path details necessary as the stream is not restricted)
    - `target:` “tapi” (no formal usage at this point. This may be omitted)
  - `subscription`: only a single entry is required in the list
    - `path: set to “{}”.` Path is just the context and hence the root element, so the path is essentially `null.`
    - `mode`: TARGET_DEFINED. This allows the provider to choose the best reporting strategy, i.e., whether the report is on change, sample etc., depending upon the specific measure characteristics
    - `sample_interval`: omitted (gNMI specifies that the value zero means provider choice, however as the field is optional in protobuf it would appear reasonable to omit the field so as to indicate provider choice).
    - `suppress_redundant`: omitted (Value suppression is not supported currently and will be specified on a per counter basis in future. Hence this is provider choice based upon other properties).
    - `heartbeat_interval`: omitted[72]
  - `qos`: omitted (the field is not used and hence should not be present in the subscription. If present it should be ignored)
  - `mode:` STREAM
  - `allow_aggregation:` omitted (the field is not used and hence should not be present in the subscription. If present it should be ignored).
  - `use_models`: omitted (not stated as this means all models are considered)
  - `updates_only`: true (same value suppression, when supported, will use another mechanism)
- `extension`:
  - `History`: This is defined in [GNMI-HIST]
    - `range:` Used to allow for lossless recovery from short comms failures. Uses `message TimeRange:`

[71] It is understood that it is necessary for some tooling to have an explicit path in a stream record. If this is the case, the context with its uuid should be sufficient.
[72] To be validated through PoCs and deployments.

        - `start:` the time from which the stream continuation is required. If the start time has not yet elapsed, the server may either return an UNIMPLEMENTED error or start from the next available performance report. This is to be interpreted as the time the record was created/logged (not the time the measure was made/collected).
        - `end:` omitted (gNMI specifies that max int means no end, however as the field is optional in protobuf it would appear reasonable to omit the field so as to indicate no end).
    - `snapshot_time:` Not used.

Following is an example subscription message (which will be encoded in protobuf):

```
{
    "subscribe": {
        "prefix": {
            "target": "tapi"
        },
        "subscription": [
            {
                "mode": "SAMPLE",
                "path": {},
                "sample_interval": 0
            }
        ],
        "updatesOnly": true
    }
}
```

Note that it is anticipated that when suppression mechanisms are supported such that a baseline needs to be provided, that “`updates_only`” will be false and there will be some initial updates that provide the baseline (as essentially current state).

- “a client creating a subscription receives an initial set of updates, terminated by a message indicating that initial synchronisation has completed, and then receives subsequent updates indicating changes to the initial state of those elements.”
- Initial: As all performance data records are independent (as same value suppression is not yet specified), there is no initial state.
- Subsequent: Each performance record is considered as new and hence corresponds to a creation where all values are considered as having new values

6.3.5 Stream record response

The response will be conformant to the model as discussed earlier. Following are examples of stream records (which will be encoded in protobuf – see section 8.7 Appendix – Protobuf encoding sketch on page 141).

6.3.5.1 A simple complete measurement

```
{
  "timestamp": "1693240502242882097", // 2023-08-28 16:35:02.242
  "prefix": {
    "target": "tapi" //No formal usage at this point. This is expected to evolve with deployment experience
  },
  "update": [
    {
      "path": {
        "elem": [ //The path will normally be the job that "controls" the measurement. It can be another entity
          {
            "name": "oam-job",
            "key": {
              "uuid": "560b49b5-670d-3d36-baca-31f560b2c4c6"
            }
          }
        ]
      },
      "val": {
        "protoBytes":{
          "measurementDetails": {
            "timeMeasurementWasSampled": "1693240037000", //2023-08-28 16:27:17.000
            "qualifiedMeasuredValue": {
              "value": -5.2,
              "units": "dBm", //Inherent in the measurement and hence need not be present
              "qualifiedValueName" : "MIN" //Present if only normalizedMeasurementType present
            },
            "nativeResourceId": "OPT-8-7-13", //Need not be present if the resource referenced by
                                                        //abstractResourceRef carries this detail
            "relativePositionOfMeasurement": "NEAR_END", //The default and hence need not be present.
            "directionOfMeasuredSignal": "RECEIVE", //Inherent in the measurement type and hence need not be present
            "sampleInterval": "900000000000", //Is in the containing job and hence need not be present
            "abstractResourceRef": "NE1-OPT-8-7-13", //Used in place of CEP/MEP/MIP reference. Treated as opaque.
            "nativeMeasurementType": "RX_OPTICAL_POWER_MIN", //Mandatory if normalized type in metadata
            "normalizedMeasurmentType": "OPT_TOTAL_PWR_INPUT" //Native type can be in metadata
          }
        }
      }
    }
  ]
}
```

6.3.5.2 A suspect measurement

This example shows a measurement that the system considers as suspect. The example also shows a minimal use of fields. The prefix is omitted as the measurement target is considered to require no further refinement.

```
{
  "timestamp": "1693240502242882097", // 2023-08-28 16:35:02.242
  "update": [
    {
      "path": {
        "elem": [
          {
            "name": "oam-job",
            "key": {
              "uuid": "560b49b5-670d-3d36-baca-31f560b2c4c6"
            }
          }
        ]
      },
      "val":{
        "protoBytes": {
          "measurementDetails": {
            "timeMeasurementWasSampled": "1693240037000", //2023-08-28 16:27:17.000
            "qualifiedMeasuredValue": {
              "value": -99,
              "valueQualifier" : "SUSPECT", //Suspect as other measures have indicated that the function is operating
              "qualifiedValueName" : "MAX" //Present as only normalizedMeasurementType present
            },
            "sampleInterval": "700000000000", //Is different from what is stated in the job
            "abstractResourceRef": "NE1-OPT-8-7-13", //Used in place of CEP/MEP/MIP reference. Treated as opaque.
            "normalizedMeasurmentType": "OPT_TOTAL_PWR_INPUT" //Standard enum. Native type can be in metadata
          }
        }
      }
    }
  ]
}
```

6.3.6 Using abstract-resource-ref

The approaches to referencing the resource to which the measurement being reported applies are covered in section 8.6 Appendix – Measurement data – Referencing resources on page 140. As explained in that description, where the abstract-resource-ref is being used, the resource being referenced will include one or more abstract-resource-refs.

Any resource could be being measured. To allow for this the general-purpose name attribute is used (as it appears in all entities).

An entity representing a resource that is to have measurement reported against it shall use the standard name attribute and include ABSTRACT_RESOURCE_REF as in the table below.

name	**List of {value-name: value} MUST include "value-name": "ABSTRACT_RESOURCE_REF" "value": string**	**RO**	**C**	• Provided by *tapi-server* • Must be present when the resource represented by the entity is to have measurements reported related to it. • Where there are several refs to report, the string can be a comma separated list of refs. • Where the set of abstract-resource-refs exceeds the string size limit one or more additional named entries can be included where the value-name can be differentiated as follows "ABTRACT_RESOURCE_REF_1" etc. This scheme can be used wherever multiple abstract-resource-refs need to be provided (instead of the comma separated approach).

6.3.7 Use of normalized-measurement-type

Normalization of measurement types is vital for correct interpretation of the measurements. The normalized-measurement-type property provides a method for normalization.

However, this may place a performance burden on the originator. Considering the importance of throughput an alternative method that simply allows the provider to convey a non-normalized measurement record and to also provide the client with normalization metadata (the normalization rules are invariant) is being considered.

In this alternative method the normalization mapping would be carried by the specification for the entity type that is originating the measurement. As this specification mechanism is not yet available, it may be necessary to use the normalized-measurement-type property.

7 Use Cases

7.1 Use Case General Considerations

7.1.1 TAPI Context

Initial TAPI deployments using TAPI 2.4 are applied to solutions where a provider is feeding a client from a single context that is taking a view of the entire controlled network for all layers.

7.1.2 Underlying behavior

It is assumed that at start up the provider will form the necessary full context and will initialize various streams. It is also assumed that the provider will recover from any internal problems to ensure that the streams allow the client to achieve eventual consistency.

There are a number of critical behaviors assumed from the underlying system (essentially use cases for other components):

- TAPI is fed from a reliable source that has necessary notifications and access to current state etc. (e.g., has alarm notifications).
  - I.e., the underlying system through the provider to the network device is reliable (appropriate pipelines, etc.) so that the provider cannot lose eventual consistency with the Devices.
  - The solution may use compacted stream in which case the compaction delay and tombstone-retention are compatible with TAPI needs.
- The notifications are well behaved both at the network device level and within the provider, e.g., for alarms such that:
  - An alarm will have a defined active and a defined clear.
  - Only legal transitions (clear to active and active to clear etc.) are represented.
- If the resource is deconfigured[73]/deleted a TOMBSTONE record will be logged for each dependent resource (e.g., alarm detector) related to the resource that was indicating active just prior to the deconfiguration/deletion.
- If a circuit pack is configured the states of any dependent resources will be reported appropriately, e.g., if an alarm detector indicates active an active alarm record will be logged.
- If a circuit pack is deconfigured any dependent resource will be tombstoned, e.g., any dependent alarm detectors that are active will have Tombstones logged.

7.1.3 Model conformance

Where a compacted log stream is used for gaining and maintaining alignment (see ST-0.3), the entities should comply with TAPI YANG and with the “Relevant parameters” definitions in the specific use case set out in [LF TR-547] as explained in the following table. This table is for guidance only. Where the table does not reflect [LF TR-547], clearly, [LF TR-547] takes precedence. All properties and structures usage specified in [LF TR-547] should be supported.

Note that some entity types:

- vary in parameters depending upon the layer-protocol (as described in [LF TR-547] use cases).
- have additional NVP properties defined in [LF TR-547].

[73] Most, but not all deconfiguration actions will result in removal of entities, hence some deconfiguration actions will not cause tombstones.

Table 3: Parameters per entity type

Entity Type (class/data-type)	Required Parameters stated in TR-547 Use Case/Section	Comments
context	Use Case 0a	
service-interface-point	Use Case 0a	Depends upon layer protocol choice.
profile		
topology	Use Case 0b	
node	Use Case 0b	
node-edge-point	Use Case 0b, 0d	Depends upon layer protocol choice.
node-rule-group	Use Case 0b	
rule	Use Case 0b	
inter-rule-group		
link	Use Case 0b	
physical-route	Use Case 0c.1	
physical-route-element	Use Case 0c.1	
connectivity-service	Use Case 1.0, 1a, 3a, 3b, 3c, 3d, 3e, 3f, 5b, 5c, 6a, 6b, 7a, 7b, 8, 9, 12c and Section 6.4	
connectivity-service-end-point	Use Case 1.0, 1a, 1c, 1e, 1f, 1g, 2a, 2b, 2c, 5b, 17b	Depends upon layer protocol choice.
connection	Use Case 1.0, 5b	Including switch control and switch
connection-end-point	Use Case 1.0, 1c, 17b, 17e	Depends upon layer protocol choice.
route	Use Case 1.0	
switch-control	Use Case 5b	
switch	Use Case 5b	
equipment	Use Case 4b	
holder	Use Case 4b	
device	Use Case 4b	
physical-span	Use Case 4b	
path-computation-context	Use Case 12a	
path-computation-service	Use Case 12a	
path-service-end-point	Use Case 12a	
topology-constraint	Use Case 12a	
routing-constraint	Use Case 12a	
objective-function	Use Case 12a	

Entity Type (class/data-type)	Required Parameters stated in TR-547 Use Case/Section	Comments
optimization-constraint	Use Case 12a	
oam-service	Use Case 17a, 17e	
oam-service-point	Use Case 17a, 17e	
meg	Use Case 17a	
mep	Use Case 17a	
mip	Use Case 17a	
current-data	Use Case 17a	
history-data	Use Case 17a	
otu-fec-performance-data	Use Case 17a	
odu-error-performance-data	Use Case 17a	
pm-threshold-data	Use Case 17a	
oam-job	Use Case 17a, 17d	
oam-profile	Section 6.8.2 and Use Case 17a, 17c	
pm-parameter	Section 6.8.2 and Use Case 17b	
threshold-config	Section 6.8.2 and Use Case 17b	
connectivity-oam-job	Use Case 17b	
connectivity-oam-service	Use Case 17b	
connectivity-oam-service-point	Use Case 17b	
pm-data	Use Case 17c	

7.1.4 Use Case Overview

The use cases are divided into 7 groups. Use Case group:

- ST-0.n deal with streaming infrastructure
- ST-1.n deal with building and operation a stream on a provider
- ST-2.n deal with strategies for the client to maintain alignment with the provider
- ST-3.n detail the approach for the client to maintain alignment with resources and with alarms
- ST-4.n deal with whole context considerations
- ST-5.n deal with streaming through the lifecycle of a service
- ST-6.n deal with Performance Monitoring Streaming using gNMI and protobuf encoding

7.2 Streaming infrastructure use cases

The following use cases are described briefly here and then illustrated in the sequence diagram (see Figure 32 and Figure 33). The use cases assume that Websockets over TCP is the chosen connection/protocol method.

The following uses cases are initiated roughly in the order set out below. The interdependence between the use cases is illustrated by the sequence diagram. The use cases in this section are normative.

7.2.1 Use Case ST-0.1: Get Auth Token

Number	ST-0.1
Name	Get Auth Token
Process/Area	Authorization and Authentication
Brief description	This use case describes how the client acquires the Auth Token.
Preconditions	The client knows the provider address and where to get the Auth Token
Type	Acquiring information
Description and workflow	The client acquires the Auth Token, for example, using method described in 7.11.2. See sequence diagram “Prepare” phase (Figure 32 and Figure 33).

7.2.2 Use Case ST-0.2: Discover supported and available streams, then select available streams

This use case deals with retrieval of data conformant with 3.7.

Number	ST-0.2
Name	Discover supported and available streams, then select available streams
Process/Area	Streaming infrastructure
Brief description	This use case describes how the client acquires information on supported and available streams.
Preconditions	UC ST-0.1 has run successfully
Type	Acquiring information
Description and workflow	The client gets all supported-stream-type structures from the context. The client examines record-content for streams that support entities or combinations of entities of interest. For each stream type that supports the appropriate entity/entity combination, the client examines: 1. log-storage-strategy: to identify a stream that has the right characteristics (e.g., COMPACTED) 2. log-record-strategy: to identify a stream that has the right record characteristics (e.g., WHOLE_ENTITY or CHANGE_ONLY) 3. record-trigger: to identify a stream that has the right record generation characteristic (e.g., ON_CHANGE) 4. Other log parameters to tune its operation to suit timer settings etc. The client examines available-streams to find running streams that reference the stream-types identified in supported-stream-type-ref. For each available-stream that is of a relevant stream-type, the client examines: 1. stream-state: to determine if the stream is operating. 2. connection-protocol: to identify a stream that offers a compatible protocol for connection (e.g., WEBSOCKETS) 3. connection-address: to determine where to connect

7.2.3 Use Case ST-0.3: Connect to Stream and align - new client

This use case deals with use of the stream which is explained further in 3.9 with content as in 3.10 and behavior as described in 3.11, 3.12, 3.13, 3.14, 3.16 and 3.17.

Number	ST-0.3
Name	Connect to Stream and align - new client
Process/Area	Streaming Infrastructure
Brief description	This use case describes the connection of a client to a stream and the client acquiring stream records.
Preconditions	UC ST-0.1 has run successfully. The client has gained knowledge of the available streams via some mechanism (such as UC ST-0.2)
Type	Gaining and maintaining alignment
Description and workflow	The client uses the results from UC ST-0.1: 1. Use the provided endpoint address and method to connect. • The client will connect with the null token[74] to cause the provider to stream from the oldest record in the stream (offset zero) 2. On connection both the client and provider stream processes are started, and the necessary communication is setup between client and provider. • As a result, the pipeline is started. • The provider reads the appropriate log from offset zero filling the pipeline and responding to ongoing demand. 3. The client demands from the provider and buffers as appropriate. 4. The client stores in a repository (e.g., Log, database etc.) 5. The client maintains a record of last commit (see also ST-2.1) Through the above activities the client works through the stream from initial record towards the most recent record 6. Passing the tombstone-retention "point" • It must take less than the tombstone-retention period to reach the tombstone-retention "point" in the log. • If it takes longer than the tombstone-retention the connection will be dropped as Tombstones for records already read may have been missed. 7. Passing the compaction "point" • From this point onward the client will be getting a full view of changes. These will be either recorded a changes where the strategy is CHANGE_ONLY or whole entities where the strategy us WHOLE_ENTITY 8. Getting close to the head of the stream • The client is well aligned with little lag. Assuming that the client is engineered to match the provider stream rate, the client should achieve a steady state and continue to be reading records close to the head of the stream, i.e., close to the most recent record logged by the provider. See sequence diagram “Connect” and “Streaming” phases where the stream starts from “offset 0” (Figure 32 and Figure 33). As changes occur in the network etc. records of the changes are appended to the appropriate logs and then streamed to the client.

[74] The token referred to here is the token from the log-record-header (see 3.10 Record content on page 23)

7.2.4 Use Case ST-0.4: Client maintains idle connection

Number	ST-0.4
Name	Client maintains idle connection
Process/Area	Streaming Infrastructure
Brief description	This use case describes how the client will maintain an idle connection
Preconditions	UC ST-0.3 has run successfully
Type	Maintaining Connection
Description and workflow	1. The client uses protocol specific mechanisms to ensure that an idle connection is maintained open. • E.g., the client periodically sends a uni-directional "Pong" frame on the connection (see [RFC6455-PONG]) in order to keep an idle WebSocket connection open. • No response expected from the server. • The server should expect a pong frame with appropriate timing. As shown in Figure 32.

7.2.5 Use Case ST-0.5: Provider delivers event storm (or slow client) – bad day

The client takes advantage of log behavior as described in 3.17.

Number	ST-0.5
Name	Provider delivers event storm (or slow client) – bad day scenario
Process/Area	Streaming Infrastructure
Brief description	The provider(s) record events at a higher rate than the client can handle
Preconditions	UC ST-0.3 has run successfully
Type	Gaining and maintaining alignment
Description and workflow	Note that the use case is written as a general description. A relatively common network example related to alarms can be used. This example includes major intermittent failure in the network, for example, several micro-bends with active mechanical vibration interference, overloaded client with reduce compute power available. 1. The pipeline continues but the client is absorbing events at a rate slower than the production and hence is slipping back down the log. 2. Eventually the client will slip back beyond compaction point. 3. If the problem resulted from excessive intermittent network activity the client will then benefit from compaction as much of the intermittent noise will be eliminated by compaction • The client will lose fidelity and will be sub-Nyquist sampling[75] so may completely lose some repeated fleeting events but regardless of the scheme used, the client would not be able to maintain full alignment with history anyway because it has hit an engineering limit. 4. The client may hover in the compaction zone until its performance improves (via some mechanism for compute power enhancement) or the network stabilizes. • Note that the intention in future is to support sophisticated backpressure interaction controlling intelligent filtering in the devices and network[76]. This would be coordinated by the provider as client load problems are detected. See sequence diagram “Streaming” phase (Figure 32 and Figure 33).

[75] See [NYQUIST]
[76] Note that this will be client induced server behavior. This assumes a single client scenario for TAPI… there is a single alarm system etc. (perhaps resilient). It can work for multiple clients so long as there is a reasonable balance of engineering and some priority scheme etc.

7.2.6 Use Case ST-0.6: Provider delivers extreme event storm (or very slow client) – very bad day

The client takes advantage of log behavior as described in 3.17.

Number	ST-0.6
Name	Provider delivers extreme event storm (or very slow client) – very bad day scenario
Process/Area	Streaming Infrastructure
Brief description	The provider(s) record events at an extremely high rate, much higher than the client can handle
Preconditions	UC ST-0.3 has run successfully
Type	Gaining and maintaining alignment
Description and workflow	Note that the use case is written in general. A relatively common network example related to alarms can be used. This example includes extreme intermittent failure in the network, for example, timing faults and micro-bends with active mechanical vibration interference, and a massively overloaded client with reduce compute power available. **Streaming continues:** 1. The pipeline continues as before, but the client rapidly slips back past the compaction point toward the tombstone-retention point. • If the client passes tombstone-retention, then there is a possibility of loss of eventual consistency as deletes will be lost. **Failure occurs:** 2. On passing the tombstone-retention point the provider forces a disconnect by dropping the connection • The client and the provider kill the pipeline. **Realign: See UC ST-0.9** See sequence diagram “Streaming” and “Drop Connection” phases (Figure 32 and Figure 33).

7.2.7 Use Case ST-0.7: Short loss of communications

The client takes advantage of log behavior as described in 3.17.

Number	ST-0.7
Name	Short loss of communications
Process/Area	Streaming Infrastructure
Brief description	The communications between the client and provider fails briefly (less than tombstone-retention time) then recovers
Preconditions	UC ST-0.3 has run successfully
Type	Gaining and maintaining alignment
Description and workflow	1. The client is consuming the stream with some delay. 2. On loss of comms the client and the provider kill the pipeline. 3. The log continues to progress on the provider side. 4. The client attempts to reconnect with the token of the most recently processed record. • This will be successful assuming the comms has recovered and the time that the client has been disconnected from the provider is no longer than the tombstone-retention time. 5. The stream is filled by the provider from record with the next valid token after the token provided. • In a short comms loss case, this token will be for the next record logged by the provider. • The record with the provided token will still exist in the log. • In a longer comms loss, compaction may have taken place such that the next valid token is for a record that was not logged adjacently to the record with the token provided. • The record with the provided token may no longer exist in the log as a result of compaction. 6. Assuming that the client was near the most recent record of the log there should be no loss of fidelity (and clearly eventual consistency will be maintained) • If the client was in the compaction zone, then there will be some loss of fidelity (see UC ST-0.6) • If the client was close to tombstone-retention, then the short comms loss may have the behavior of a long comms loss (see UC ST- 0.8) See sequence diagram “Streaming”, “Loss”, “Connect” (with specific token) and “Streaming” phases (Figure 32 and Figure 33).

7.2.8 Use Case ST-0.8: Long loss of communications requiring realignment

The client takes advantage of log behavior as described in 3.17.

Number	ST-0.8
Name	Long loss of communications
Process/Area	Streaming Infrastructure
Brief description	The communications between the client and provider fails then recovers
Preconditions	UC ST-0.3 has run successfully
Type	Gaining and maintaining alignment
Description and workflow	1. The client is consuming the stream with some delay. 2. On loss of comms the client and the provider kill the pipeline. 3. The log continues to progress on the provider side. 4. The client/comms is down for longer than the tombstone-retention. 5. The client attempts to reconnect with the token of the record that it previously successfully processed. 6. The provider recognizes that this is outside tombstone-retention and streams from the oldest record etc. • The client could take action to revert to the oldest record by not providing the token, however, it will certainly take longer to align on average than relying on the provider knowledge regarding whether alignment is required or not. 7. See UC ST-0.9 See sequence diagram “Streaming”, “Loss”, “Connect” (with specific token but forced to offset = 0) and “Streaming” phases (Figure 32 and Figure 33).

7.2.9 Use Case ST-0.9: Client requires realignment

Number	ST-0.9
Name	Client requires realignment
Process/Area	Streaming Infrastructure
Brief description	For some reason, the provider determines that the client needs to be aligned/realigned or the client chooses to realign
Preconditions	UC ST-0.3 has run successfully
Type	Gaining and maintaining alignment
Description and workflow	1. The client reconnects with the previous token. • Or connects with no token. 2. On connection both the client and provider stream processes are started, and the necessary connection is made between client and provider. • As a result, the pipeline is started. 3. The provider recognizes the token is outside tombstone-retention and restarts the stream from the oldest record. 4. The provider informs the client, via the stream, that it is back at oldest record (offset zero) 5. The client consumes the stream to regain alignment. • If the provider is logging records at a significantly higher rate than the client can handle, the client will stay beyond the tombstone-retention and will get forced to realign. • The client may utilize some efficient realignment strategies (see ST-2.2) • The client is expected to use some form of “garbage collection” mechanism (such as “mark and sweep”) to ensure that instances that are stored by the client but that are no longer present in the provider stream are eliminated. Assuming that the reason for the stream restart have gone, then the client will regain alignment (eventual consistency) and will return to the state achieved in UC ST-0.3 See sequence diagram “Connect” (with specific token but forced to offset = 0) and “Streaming” phases (Figure 32 and Figure 33).

7.3 Building and operating a stream on a provider

The following use cases describe how the provider should populate the stream. The details of storage and internal mechanism sketched are for informative example, however, the data in the log is essentially normative.

The use cases in this section are normative, except use case ST-1.3.

7.3.1 Use Case ST-1.1: Provider initializes and operates a stream

Number	ST-1.1
Name	Provider initializes and operates a stream
Process/Area	Streaming operation
Brief description	The provider initializes the streams from internal sources of current state and change
Preconditions	The provider is running, the streaming infrastructure is running and there is a defined stream to populate. Note: As the activities associated with achieving the preconditions will vary significantly from solution to solution and are internal detail, no use cases have been provided for this area.
Type	Preparing and maintaining a stream

Description and workflow	1. The stream type is added to the supported-stream-type structure if not already listed. • The data shall comply with the structure set out in section 3.7.1 2. The provider creates a stream for a specific information source. 3. The stream is added to the available-stream structure with a stream-state of “ALIGNING”. • The data shall comply with the structure set out in section 3.7.2 • A client may connect to the stream from this moment on • In this state no records will be streamed to a connected client 4. The provider populates the stream from internal stores of current state and change. • The data shall comply with the structure set out in section 3.10 • [example] If the provider uses compacted-log-based streams internally, this may simply be connecting to the appropriate internal streams from the oldest record and performing necessary transformations to form the TAPI stream content. • [example] If the provider, internally, uses a current state repository with a truncated stream of changes, then it will populate the stream with appropriately transformed current state followed by recent changes. ◦ It is possible that the current state repository does not record event time. If this is the case, the event-time-stamp should take advantage of the approx-date-and-time structure. The event-time-stamp should be the same as the log-append-time-stamp with the spread set to “BEFORE”. • Assuming the log has settings as per the example set out in 7.2.2 Use Case ST-0.2: Discover supported and available streams, then select available streams on page 92 (COMPACTED, WHOLE_ENTITY or CHANGE_ONLY and ON_CHANGE), for current state and for change notifications the provider appends a record of the whole entity instance using the appropriate structure for the class with: ◦ record-type set to CREATE_UPDATE etc. ◦ record-content set to the appropriate object-class-identifier. ◦ entity-key set to a value appropriate for identifying instances of the object class which may be a relative address for a local class, a uuid for a global class, or any value that is unique/invariant for the entity instance in the stream such that a CREATE_UPDATE, DELETE, TOMBSTONE etc. events all have the same entity-key. • For delete notifications the provider appends a DELETE record for the delete of the entity followed by a TOMBSTONE record for the entity-key of the DELETE record 5. Once stream population is sufficient, the provider changes the stream-state to “ACTIVE”. • The provider may be capable of making the stream “ACTIVE” prior to complete population of the stream. • In this state records will be streamed to any connected clients • Records for new create/change/delete events will be appended to the log as defined in step (4) above. 6. When the stream-state is “ACTIVE”, the provider may set the stream-state to “PAUSED” at any point. • In the “PAUSED” state no records will be streamed to connected clients. If the stream had changed to “PAUSED” from “ACTIVE”, a client may continue to receive records from the comms buffers for some time. 7. When the stream-state is “PAUSED”, the provider may set the stream-state to “ACTIVE” at any point. • At the transition from “PAUSED” to “ACTIVE” records will be streamed from the next record after the one streamed just before the stream entered the “PAUSED” state.

7.3.2 Use Case ST-1.2: Provider recovers a stream after internal loss

Number	ST-1.2
Name	Provider recovers a stream after potential loss of data
Process/Area	Streaming operation
Brief description	The provider suffers some loss of data from a source feeding a stream
Preconditions	The stream is in the available-stream list.
Type	Preparing and maintaining a stream
Description and workflow	1. The provider detects a potential loss of data from some area of the solution (the “area of concern”) that provides data to a stream. • This may be an internal component, perhaps where the provider solution is distributed, or an external feed. 2. The provider may set the stream-state to “PAUSED” or “ALIGNING”. • In “PAUSED” or ”ALIGNING” state no records will be streamed to connected clients. If the stream had changed to “PAUSED”/”ALIGNING” from “ACTIVE”, a client may continue to receive records from the comms buffers for some time. 3. The provider will ensure that the stream is populated with the correct state for the “area of concern”. To do this it may fully rebuild the stream or may run an internal audit on the stream that ensures that the stream is populated with the correct state for the area of concern. If any cases of misalignment are detected the provider may set the stream-state to “ALIGNING”. The provider may realign the stream using various methods, for example: • For records of an entity that are found in the log feeding the stream where that entity is no longer at the source and most recent record is not of record-type of TOMBSTONE the provider will append a TOMBSTONE record for the entity • For an entity that is found at the source but where there are no records in the log that is feeding the stream or there are only DELETE or TOMBSTONE records the Provider will append the details of the current state of the entity (taking advantage of approx.-date-and-time as necessary) • The provider will validate as appropriate that the latest record in the log for each entity that exist in the area of concern is aligned with the current state of that entity. If the entity in the log is not aligned the provider will append a record that reflects the current state (taking advantage of approx-date-and-time as necessary). 4. Once stream population is sufficient, the Controller changes the stream-state to “ACTIVE”. • The stream may have remained “ACTIVE” throughout the process. • In this state records will be streamed to any connected clients. • Records for new create/change/delete events will be appended to the log. 5. As a result of the above process the client should be returned to a state of “eventually consistent” with the state of presented context with no need to take any specific action.

7.3.3 Use Case ST-1.3: Provider recovers a stream after an upgrade

Note that this use case is informative.

Number	ST-1.3
Name	Provider recovers a stream after an upgrade
Process/Area	Streaming operation
Brief description	The provider is upgraded
Preconditions	The stream is in the available-stream list.
Type	Preparing and maintaining a stream
Description and workflow	There are two distinct sub-cases: 1. The stream log persists through the upgrade: An approach similar to UC ST-1.2 could be taken where the stream log is updated with changes that occurred during the upgrade where an approach similar to UC ST-2.2 (“mark and sweep”) is used to clean up the stream. Again, approx-date-and-time could be used. 2. The stream log is lost during the upgrade: An approach similar to UC ST-1.1 could be taken. When the client connects with the previous token UC ST-0.3 will be run.

7.4 Client maintains alignment – Example strategies and approaches

This section provides further clarification of earlier use cases. The use cases in this section are informative.

7.4.1 Use Case ST-2.1: Client aligns with a stream

The client prepares for realignment during the initial alignment phase. This use case embellishes ST-0.3 and is provided as an example of a client solution.

Number	ST-2.1
Name	Client aligns with a stream
Process/Area	Streaming Infrastructure
Brief description	The client connects to provider and aligns
Preconditions	UC ST-0.1 has run successfully. The client has gained knowledge of the available streams via some mechanism (such as UC ST-0.2)
Type	Gaining and maintaining alignment
Description and workflow	1. Client prepares to connect to a stream for the first time by • Setting current-stream-alignment-attempt-counter to 1 for the stream (stream-id) • Setting the alignment-attempt-start-time to current time. 2. Client connects to the stream and receives records ("simple update") as per UC ST-0.3. 3. Client identifies the entity instance to add/update/delete using the entity-key (and potentially other data) and attempts to locate the entity using the entity key as well as parent-address where provided (see 3.10.3). • If record-type = CREATE_UPDATE, UPDATE, CREATE 1. If entity exists, then update the entity and set stream-alignment-attempt-counter in the stored entity to current-stream-alignment-attempt-counter (stream id) 1. Update can be a simple overwrite, but client may want to mark changes or notify changes. 2. If entity instance does not exist, then create entity instance (with the alignment attempt counter value etc.) • If record-type = CHANGE 1. If entity exists, then update the entity with the changed values and set stream-alignment-attempt-counter in the stored entity to current-stream-alignment-attempt-counter (stream id) 2. {This should never occur} If entity instance does not exist, then there has been an error in alignment and alignment must be restarted. • If record-type = DELETE or TOMBSTONE (may be two records, both should be processed, may be able to make this efficient by expecting the TOMBSTONE record if there is a DELETE record) 1. If entity exists, then delete the entity. 2. If entity instance does not exist, then take no action. 4. Note: • The parent-address provides the positioning in the tree for the entity. • The UUID uniquely identifies an entity and should be sufficient to locate the entity. • In such a solution the parent-address is still necessary in a CREATE_UPDATE, UPDATE, CREATE and CHANGE to provide the association to the grouping entities (such as Node grouping NEPs). • The entities identified in the parent address may not yet be present due to asynchronous arrival of stream information. Where this is the case, it is expected that a skeleton tree would be constructed to be subsequently filled in as the entities arrive. 5. Each time a record is successfully processed, the client records the token in the token-from-latest-record-successfully-process-for-stream (stream-id) and the log-append-time-stamp for this alignment attempt • This assumes the client processes the stream records in the sequence received, if a more complex process is used, a more complex recording of tokens will be required. See also UC ST-4.2.

7.4.2 Use Case ST-2.2: Client realigns

Number	ST-2.2
Name	Client realigns
Process/Area	Streaming Infrastructure
Brief description	The client realigns after reconnecting to the stream.
Preconditions	UC ST-2.1 has run successfully
Type	Gaining and maintaining alignment

Description and workflow	1. Connection is dropped (triggered by client, by provider or by comms failure). 2. Client reconnects to the stream. This use case assumes that there is a resync triggered: • Provider driven resync (and very long comms failure) 1. Client provides the value (token) from token-from-latest-record-successfully-process-for-stream (stream-id)) in the connection request. • Client driven resync. 1. Client provides no token. 3. Provider streams from first record (offset 0). 4. Client detects realignment in first record received from the stream. • Client increments current-stream-alignment-attempt-counter for the stream (stream-id) and stores, for this alignment attempt, the alignment-attempt-start-time (set to current time), the token-from-latest-record-successfully-process-for-stream (stream-id) and the log-append-time-stamp from the first record just received 5. Client identifies the entity instance to update using the entity-key and parent-address (along with potentially other data) and attempts to locate the entity ("alignment update") • If record-type = CREATE_UPDATE, UPDATE, CREATE 1. If entity exists and has: 1. Newer log-append-time-stamp in the currently stored entity compared to the newly received record, then ignore record just received 2. Same log-append-time-stamp (and token) in the currently stored entity and the new record, then store the newly received entity values and set stream-alignment-attempt-counter in the stored entity to current-stream-alignment-attempt-counter (stream-id) 3. Older log-append-time-stamp in the currently stored entity compared to the new record, then update (overwrite) the entity and set stream-alignment-attempt-counter in the stored entity to current-stream-alignment-attempt-counter (stream id) 2. If entity instance does not exist, then create entity instance as usual (with the alignment attempt counter value etc.) • If record-type = CHANGE 1. If entity exists and has: 1. Newer log-append-time-stamp in the currently stored entity compared to the newly received record, then ignore record just received 2. Same log-append-time-stamp (and token) in the currently stored entity and the new record, then update the entity with the newly received change values and set stream-alignment-attempt-counter in the stored entity to current-stream-alignment-attempt-counter (stream-id) 3. Older log-append-time-stamp in the currently stored entity compared to the new record, then update the entity with the recorded changes and set stream-alignment-attempt-counter in the stored entity to current-stream-alignment-attempt-counter (stream id) 2. {This should never occur} If entity instance does not exist, then there has been an error in realignment and realignment must be restarted.

	• If record-type = DELETE or TOMBSTONE (may be two records, both should be processed, may be able to make this efficient by expecting the TOMBSTONE record if there is a DELETE record) 1. If entity exists and has 1. Newer log-append-time-stamp in the currently stored entity compared to the newly received record, then ignore record just received 2. Same log-append-time-stamp in the currently stored entity and the new record, then delete the entity 3. Older log-append-time-stamp in the currently stored entity compared to the new record, then delete the entity 2. If entity instance does not exist, then take no action 6. Client receives further records. 7. Alignment has progressed sufficiently: • If log-append-time-stamp from the received record is (both of) 1. More recent than the log-append-time-stamp-from-latest-record-successfully-process-for-stream (stream-id) for the previous alignment attempt (i.e., the client has passed the time of the last record successfully received before the realignment) 2. Less than the compaction delay from current time (compaction will not remove old records due to records that are newer than the compaction delay) • Then the client can sweep through the repository removing any entities still marked with previous alignment attempt counter value • Stream processing can continue uninterrupted as the sweep takes place (the client can revert to “simple update” processing as the sweep starts) 8. Once the sweep is complete the client has achieved “eventual consistency” with the provider

7.4.3 Use Case ST-2.3: Client performs a stream audit

Number	ST-2.3
Name	Client performs a stream audit
Process/Area	Streaming Infrastructure
Brief description	The client considers that there may be an alignment issue and decides to audit the stream.
Preconditions	UC ST-0.3 has run successfully
Type	Gaining and maintaining alignment
Description and workflow	Notes: 1. Whilst maintaining an existing connection to a stream, the client makes a second connection as described in UC ST-0.3 (1-3). 2. The client runs the second stream as if realigning and marks valid current state 3. When it passes the time of a current state, the entity is marked as suspect. Once sufficient progression has been achieved, the client will remove proved bogus records 4. When thing appears in the stream that is not in the repository, it is retained. Once sufficient progression has been achieved, the client will add the entity to the repository (avoiding race conditions with the main stream receiver).

7.5 Gaining and maintaining Alignment with individual network resources

A vast majority of the TAPI entities can be dealt with in the same way. The provider can offer streams that each include one or more classes. A stream will send records of all instances of all of the classes identified in its definition.

The use cases in this section are informative.

7.5.1 Use Case ST-3.1: Client maintains alignment with all instances of a class (e.g., Node) in a context

This use case considers a COMPACTED log solution.

Number	ST-3.1
Name	Client maintains alignment with all instances of a class (e.g., NODE) in a context
Process/Area	Streaming Operation
Brief description	The client discovers the availability of a stream for a class, connects to the stream aligns and maintains alignment ongoing.
Preconditions	Client knows which classes it needs to monitor and has rum UC ST-0.1
Type	Gaining and maintaining alignment
Description and workflow	The client runs: 1. UC ST-0.2: From this the client has the address and connection method for an appropriate COMPACTED stream for the class(es) of interest (in this example NODE). 2. UC ST-0.3: From this the client achieves “eventual consistency” with the state of the class(es) of interest (e.g., NODE) including dealing with ongoing change. 3. UC ST-0.4: To ensure that the connection remains open. It is possible, during operation, that conditions in the network are such that any one of the following may occur. 4. UC ST-0.5: Where the client may lose some information fidelity but will maintain “eventual consistency” without any special action. 5. UC ST-0.6: Where the client and provider may need to deal with UC ST-0.9 6. UC ST-0.7: Where the client will simply suffer a short delay in receipt of information but will lose no fidelity or integrity. 7. UC ST-0.8: Where the client and provider may need to deal with UC ST-0.9 8. UC ST-0.9: Where the client will clean up its repository as it aligns with the provider

7.5.2 Use Case ST-3.2: Client maintains alignment with all alarms in the context

This use case deals with alarms that are processed, logged and streamed as described in sections 4.1, 4.2, 4.3 and 4.4. The alarm records abide by the model as described in 4.5 (further explained in 4.6). The alarm clear will be reported as described in 4.7. It is assumed that the event-time-stamp will be as described in 4.8 and that the optional normalization will perform as described in 4.9. The provider is also expected to deal with ensuring meaningful detection as described in 4.10.

Number	ST-3.2
Name	Client maintains alignment with all alarms in the context
Process/Area	Streaming Operation
Brief description	The client discovers the availability of a stream for Alarms, connects to the stream, aligns and maintains alignment ongoing.
Preconditions	Client knows which classes it needs to monitor and has run UC ST-0.1
Type	Gaining and maintaining alignment
Description and workflow	This is essentially UC ST-3.1 but where the specific class is CONDITION_DETECTOR 1. The client and provider cover UC ST-3.1.

7.6 Dealing with the whole context of resources

This section provides an overview of how the client would be expected to deal with all resources in a context. The use cases in this section are informative.

The associated normative document [RIA SS] provides a view of stream events expected (Mandatory) and possible (Conditional) for particular identified network scenarios.

7.6.1 Use Case ST-4.1: Client maintains alignment with all resources in the context

Number	ST-4.1
Name	Client maintains alignment with all resources in the context
Process/Area	Solution Operation
Brief description	The client listens to a number of streams and assembles a view of the network
Preconditions	Client is running
Type	A Client building and maintaining the context
Description and workflow	1. The client runs UC ST-0.1 2. The client runs UC ST-2.x for all relevant classes and information • The client starts various streams in a sequence compatible with its alignment strategy 1. The streams guarantee order within any instance of a class but do not guarantee any ordering between instances of classes 2. For example, the client may decide to align nodes first • As entity information is received the client resolves references on-the-fly. 1. Where references do not resolve immediately, due to differential propagation delay etc., the client uses an appropriate method to defer the reference mapping until the relevant entity is received 2. As noted in UC ST-2.1 the tree referenced by the parent-address may not yet be fully present and hence a skeleton of the tree will need to be constructed and subsequently populated. • The client builds a view of interrelated network resources

7.6.2 Use Case ST-4.2: In a resilient solution the Controller the client is connected to becomes unavailable

Number	ST-4.2
Name	In a resilient solution the Controller the client is connected to becomes unavailable
Process/Area	Solution Operation
Brief description	The client detects the Controller to which it was connected is no longer available and connects to an alternative Controller instance for the streams of interest
Preconditions	Client is running UC ST-4.1
Type	A Client building and maintaining the context
Description and workflow	1. The client runs UC ST-0.1 for the new Controller instance 2. The client runs UC ST-2.1 for the new Controller instance

7.7 Connectivity Service Lifecycle

The use cases in this section are informative.

7.7.1 Use Case ST-5.1: Provide streams network changes to the client

Number	ST-5.1
Name	Provider streams network changes to the client
Process/Area	Solution Operation
Brief description	Changes in the network are detected by the provider and interpreted into changes to entities in the TAPI context.
Preconditions	Client is running UC ST-4.1
Type	Provider is monitoring the network for change
Description and workflow	1. The provider detected a change in network state and makes the appropriate adjustments to the TAPI context to reflect the change (e.g., creation of connections etc.) 2. The provider streams the changed entities via the appropriate streams to the client. 3. The attached client absorbs the stream maintaining eventual consistency with the network state. Note that the maintenance of eventual consistency is dependent upon UC ST-0.n.

7.7.2 Use Case ST-5.2: Client runs a provisioning use case ([LF TR-547] UC1x etc.)

Number	ST-5.2
Name	Client runs a provisioning use case ([LF TR-547] UC1x etc.)
Process/Area	Solution Operation
Brief description	Client provides connectivity-service details, and the provider takes action to satisfy the request.
Preconditions	Client is running UC ST-4.1
Type	A Client adjusting connectivity-services
Description and workflow	1. The client provides the connectivity-service details including routing constraints etc. 2. The provider assesses the request etc. as per [LF TR-547] UC1x etc. 3. The provider runs UC ST-5.1 and streams changes as they occur.

7.7.3 Use Case ST-5.3: Client runs the service deletion use case ([LF TR-547] UC10)

Number	ST-5.3
Name	Client runs the service deletion use case ([LF TR-547] UC10)
Process/Area	Solution Operation
Brief description	Client requests a delete of a connectivity-service
Preconditions	Client is running UC ST-4.1
Type	A Client adjusting connectivity-services
Description and workflow	1. The client requests the delete of a connectivity-service. 2. The provider assesses the request etc. as per ([LF TR-547] UC10) 3. The provider runs UC ST-5.1

7.8 Measurement data streaming

The use cases in this section are informative.

7.8.1 Use Case ST-6.1: Provider initializes and operates a measurement data stream

Number	ST-6.1
Name	Provider initializes and operates a measurement data stream (based on ST-1.1)
Process/Area	Streaming operation
Brief description	The provider initializes the streams from internal sources of measurement data
Preconditions	The provider is running, the streaming infrastructure is running and there is a defined stream to populate. Note: As the activities associated with achieving the preconditions will vary significantly from solution to solution and are internal detail, no use cases have been provided for this area.
Type	Preparing and maintaining a stream
Description and workflow	1. As ST-1.1 step (1). 2. As ST-1.1 step (2). 3. As ST-1.1 step (3). 4. Derived from ST-1.1 step (4) Assuming that measurement data collection is enabled in the controller and the devices controller, as measurement data arrives from underlying system it is evaluated for relevance to the entities in the TAPI context. The provider populates the stream with relevant measurement data conforming to the structure set out in section 6.3 The gNMI Stream on page 80. • The provider will also make available via a stream (as per ST-1.1) the oam-jobs that correspond to the active measures. 5. Derived from ST-1.1 step (5) Once stream population is sufficient, the provider changes the stream-state to “ACTIVE”. • The provider may be capable of making the stream “ACTIVE” prior to complete population of the stream. • In this state records will be streamed to any connected clients • New measurement data will be appended to the log as defined in step (4) above. 6. As ST-1.1 step (6). 7. As ST-1.1 step (7).

7.8.2 Use Case ST-6.2: Client subscribes to a measurement data stream

Number	ST-6.2
Name	Client subscribes to a measurement data stream (related to ST-0.3)
Process/Area	Streaming Infrastructure
Brief description	This use case describes the connection of a client to a stream and the client acquiring stream records.
Preconditions	UC ST-0.1 has run successfully. The client has gained knowledge of the available streams via some mechanism (such as UC ST-0.2)
Type	Gaining and maintaining alignment
Description and workflow	The client uses the results from UC ST-0.1: 1. Use the provided endpoint address and method to connect. • The client will subscribe to the stream using the gNMI subscription method 2. On subscription both the client and provider stream processes are started, and the necessary communication is setup between client and provider. • As a result, the pipeline is started. • The provider reads the appropriate internal measurement data log, filling the pipeline and responding to ongoing demand. 3. The client demands from the provider and buffers as appropriate. 4. The client stores in a repository (e.g., Log, database etc.) 5. The client maintains the time of the last successfully received record enable resubscription after a comms fail and recovery cycle (see also ST-6.4) Through the above activities the client works through the stream from initial record towards the most recent record Assuming that the client is engineered to match the provider stream rate, the client should achieve a steady state and continue to be reading records close to the head of the stream, i.e., close to the most recent record logged by the provider.

7.8.3 Use Case ST-6.3: Provider maintains the stream

Number	ST-6.3
Name	Provider maintains the stream
Process/Area	Streaming operation
Brief description	As the provider is populating the streams from internal sources of measurement data there is a change in the measured data set
Preconditions	The provider is running, the streaming infrastructure is running and there is a defined stream to populate. Note: As the activities associated with achieving the preconditions will vary significantly from solution to solution and are internal detail, no use cases have been provided for this area.
Type	Preparing and maintaining a stream

Description and workflow	1. The provider continues to populate the NBI stream. 2. The following situations may occur: a. First measurement from a new measurement point enabled (optional): i. If the measurement is marked such that it is clear that this is the first sample of an ongoing set of samples then the sample_qualifier should be set to FIRST. ii. If the measurement is not marked in any way, then the sample_qualifier should be omitted even if this is the FIRST measurement iii. Following measurements will not have sample_qualifier present until one of the following occur b. Final measurement from a measurement point that has been disabled (optional): i. If the measurement is marked such that it is clear that this is the final sample of an ongoing set of samples then the sample_qualifier should be set to FINAL. ii. If the measurement is not marked in any way, then the sample_qualifier should be omitted even if this is the final measurement. c. Measurement data report delayed (optional): i. If the measurement data report is either marked as delayed or is clearly delayed based upon the timestamp being significantly historic (e.g., a report of a per-minute measurement where the specific measurement occurred several hours ago where other reports being made at that same time are up to date) and especially where other reports for the same point and measurement that were made after that time have already been reported then the sample_qualifier should be set to DELAYED. ii. If it is not possible to determine this, then the sample_qualifier should be omitted. d. Measurement data appears invalid (optional) i. If the measurement data provided is clearly invalid (e.g., there were more errors in a measured period than there were bits conveyed in that period) then the value_qualifier should be set to INVALID, but the value should still also be provided where it can be (note that the value may not fit in the range of the value field for some reason, in this case the value cannot be provided) ii. If the solution is not aware of valid range, value_qualifier "INVALID" should not be used other than for the cases where the value provided does not fit within the range of the field. e. Measurement data appears suspect (optional) i. If the measurement data provided does not appear reasonable (e.g., there most bits conveyed in that period were reported as in error) then the value_qualifier should be set to SUSPECT, but the value should still also be provided ii. If the solution is not aware of valid range, value_qualifier "SUSPECT" should not be used. f. Measurement data appears partial (optional) i. If the measurement data provided indicates a sample_interval that was significantly less than expected for the measurement then the value_qualifier should be set to PARTIAL (clearly, the value should also be provided) ii. Note that there may be other cases where the value_qualifier "PARTIAL" applies and "PARTIAL" may be used for these cases as the emerge. iii. If the solution has no expectation for sample_interval, then "PARTIAL" should not be use. g. Measurement data is missing (optional) i. If the measurement data should have been provided for a point at a particular time but it was not provided, then the value_qualifier should be set to MISSING (note that this is vital for a case where same value suppression is supported. The identification of MISSING

	takes place in the control process where the suppression is being applied. No suppression cases are supported in TAPI 2.5.0). ii. If the solution is not aware of when measurements should be provided that the value_qualifier "MISSING" should not be used.

7.8.4 Use Case ST-6.4: Short loss of communications

Number	ST-6.4
Name	Short loss of communications (related to ST-0.7)
Process/Area	Streaming Infrastructure
Brief description	The communications between the client and provider fails briefly (less than measurement data retention time) then recovers
Preconditions	UC ST-6.2 has run successfully
Type	Gaining and maintaining alignment
Description and workflow	1. The client is consuming the stream with some delay and is maintaining a record of the time of the latest successfully processed records. 2. On loss of comms the client and the provider kill the subscription. 3. The log continues to progress on the provider side. 4. The comms recovers 5. The client attempts to resubscribe from where it was by providing the time of the last successfully processed record. • This is provided in the start value of the range parameter of the history extension of the subscription, as detailed in section 6.3.4 Subscription on page 83. • This will be successful assuming the comms has recovered and the time that the client has been disconnected from the provider is no longer than the record retention time. 6. The stream is filled by the provider from the first record with time identified as start in the request from the client. • In a short comms loss case, the record will still exist • There will be some duplication of data especially where there were several records provided at the same time. Sending the same data simplifies the processing of "start". • In a longer comms loss, the record with the provided time may no longer exist in the log as a result of truncation. In this case the oldest record will be sent but that will NOT be from the time provided in the "start" parameter. • Note that where the oldest record in the log is at the time requested as start, that record must not be sent as a record at that time may have been lost Note that a long loss of communications simple means that the client has lost data.

7.9 Message Sequence example

The hybrid message sequence diagram, in Figure 32 below, captures all relevant flows for the listed use cases. The diagram includes several special symbols that are highlighted in the key below the diagram. The diagram essentially lays out a sketch of a solution that would support the normative aspects of streaming. This section is informative.

It is assumed that the client has already used Restconf to get supported-streams and to get available-streams so as to have the connection method. In this example the connection method is assumed to be WebSockets.

In the diagram the behavior of the Device Source and TAPI Context are summarized. The diagram only shows presence and fundamental flow for these elements.

Two pipelines, the TAPI Context Pipeline and the Extended Pipeline, are shown in the diagram as yellow background rectangles. The TAPI context pipeline intentionally shows no detail as that is outside the scope of the interface definition.

The diagram shows coupled asynchronous parallel/concurrent repeating activities and independent asynchronous repeating activities each in a dashed box. Where the asynchronous activities are coupled there is a dashed line arrow showing the relationship as a ratio of activity, n:1, which indicates that there are potentially more trigger activities than result activities, or as a 1, which indicates that "eventually" there will be the same number of activities in both asynchronous elements (as a result of a flow through both). Some activities are shown as nested. Where nested there is an indication where there are n repeats of the inner asynchronous activity for each of the outer activities. Buffers/logs are shown to emphasize the asynchronous coupling. The compacted log is shown with a buffer/log symbol annotated with an "X" indicating compaction (the deletion of records in the log) and "0" indicating that the record is for all system time, i.e., for the time the system has been logging, (where compaction will remove duplicates and hence contain the log size).

The logs marked with “r” are limited to “r” records (size or number) and will block when that size is reached applying back pressure to the feed (via “fill”). Hence, in combination, there is a pipeline with backpressure from the client store to the provider log. This includes all sequences and flows in the “Extended Pipeline” shown as a large yellow rectangle.

To the left of the figure (i.e., to the left of the “underlying external comms” (brown vertical bar)) is the client side (which initiates the connections etc.), shown as a stylized example. The client is shown with both a database option and a compacted log option for storage. The critical features are the "Pong" and "Last commit". The majority of the client depiction is to explain last commit. Last commit is used by the client on reconnection to continue where it left off (“token~x”).

The external comms between client and provider is shown as a brown bar and is not considered in any detail. It is assumed that it is reliable (e.g., TCP) and is playing an active part in maintenance of the pipeline.

The client token is opaque so the client has no knowledge of sequence through the token, although there is also exposure of the sequence number, this is primarily intended for stream analysis (note that it may be beneficial as part of normal behavior to validate communication).

The middle of the diagram shows the provider and explains the basic flows related to initial connection, loss of connection and forced connection drop.

To the extreme right of the figure are vertical progression bars that highlight phases of interaction between the client and provider.

The example functions (at the head of each timeline) can best be explained in terms of the phases of interaction in which they participate.

The "Prepare" phase involves an authentication service, here shown within the provider system, which supplies Authentication Token on request to a Web Sockets (WS) client connection control function in the client. The Authentication Token ("Auth Token") provided is used in a connection request in the "Connect phase". Along with a null stream token (absence of a stream token). These exchanges are described in section 7.11 Message approach (WebSocket example) on page 123. This causes the stream to start from the oldest record (offset 0).

In the "Streaming" phase the provider continues to take records from the Compacted Log to the left and feed them into the communications system via appropriate queues that blocks on fill[77]. In the example, the "Source and Process" function collects records from the log and 'publishes' them to "WS Endpoint Stream Server Source Actor" that fills a blocking queue feeding the "WS Endpoint Stream Server". The "Websockets Endpoint Stream Server" feeding the underlying comms system (which is assumed to be reliable and blocking when full). On the client side, the stream client takes from the comms system and places on a queue. This results in there being an effective demand back to the Stream Server (stylized in the figure).

As noted above, the yellow background shows the whole (Extended) Pipeline. The intention is that this pipeline guarantees delivery to the client repository (shown as a compacted log "AND/OR" DB, but any repository relevant to the client is appropriate). The repository to the left of the diagram and associated functions are a simple sketch of the sort of operations that may take place. The key consideration on the left side is the "Last commit" token which records the last successfully stored in the persistent repository. The client benefits from recording this token as without it a full resync on comms fail would be required. The client need not record every token (as suggested by the "n:1" relationship).

The remaining phases, "Loss", "Drop Connection" (two variants) and "Kill" pipeline are all related to failure modes. Assuming the client still exists and requires the stream information, in all failure cases, the client will reconnect using the stream token.

When a token is provided during the "Connect" phase the provider assesses the token to determine which record to start the stream from. If the token relates to a record:

- Newer than the compaction delay, then next record that was appended to the log is streamed.
- Between the compaction delay and the tombstone-retention, then next record in the log, which is not necessary the next record that was appended due to compaction, is streamed
- Older than the compaction delay, then the provider forces a resync by streaming from the oldest record in the stream (Offset 0)

"Log and stream Control" monitors the state of each stream in the provider and drops the client connection when any problem is detected (e.g., it is blocked on a record older than tombstone-retention).

Pong frames are required to maintain the stream when there is no activity, and a frame is required every 30 seconds (default time).

[77] For the pipeline to maintain integrity it is vital that back pressure prevents the queue from overflowing. The "blocking" is essentially through "demand". If there is a queue overflow event, the pipeline must be restarted.

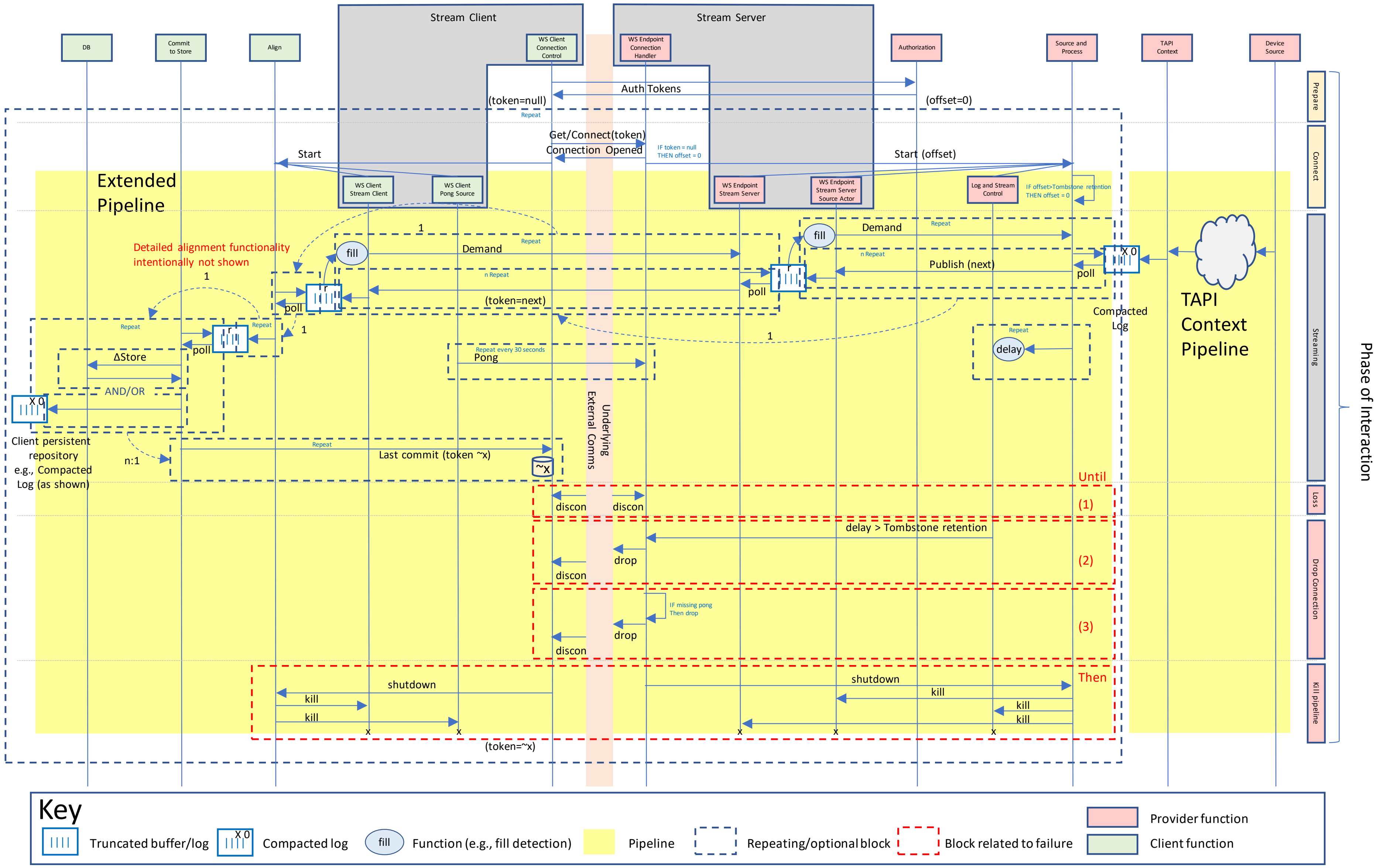


Figure 32 Hybrid Message Sequence Diagram for example implementation corresponding to Use Cases

The provider shall support:

- Authentication as described.
- A pipeline using backpressure from the communications system to ensure reliable delivery from the internal compacted log (full or emulated).
- Start streaming from:
    - The oldest record.
    - The record after the one with the stream token specified by the client in a connect request.
- Forced restart of stream from the oldest record when it detects:
    - A very slow client taking records older than tombstone-retention.
    - A reconnect request with a token for a record older than tombstone-retention.
- Pong frame timeout connection drop behaviour

The following figure shows the use cases and relationship to the hybrid message sequence chart in the figure above. The heading bars are the same as the righthand vertical bars on the previous figure. The flow across the figure assumes a chaining of the use cases in the order they are described in the earlier sections.

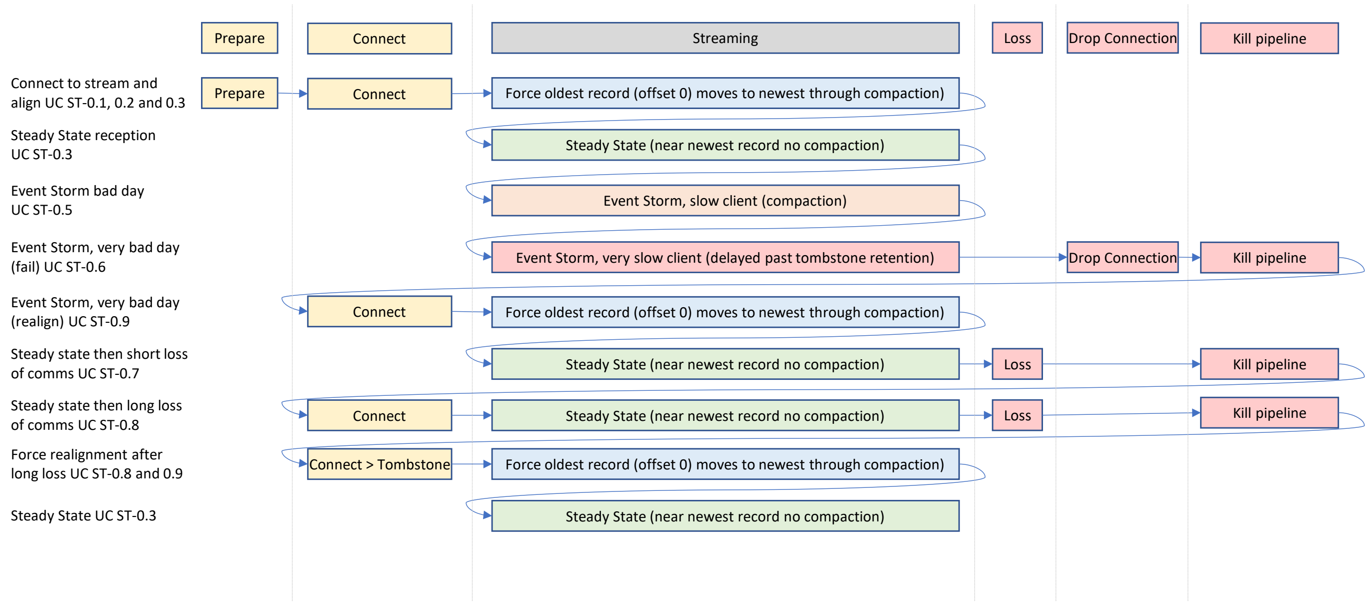

Prepare
Connect
Streaming
Loss
Drop Connection
Kill pipeline
Connect to stream and align UC ST-0.1, 0.2 and 0.3
Prepare
Connect
Force oldest record (offset 0) moves to newest through compaction)
Steady State reception UC ST-0.3
Steady State (near newest record no compaction)
Event Storm bad day UC ST-0.5
Event Storm, slow client (compaction)
Event Storm, very bad day (fail) UC ST-0.6
Event Storm, very slow client (delayed past tombstone retention)
Drop Connection
Kill pipeline
Event Storm, very bad day (realign) UC ST-0.9
Connect
Force oldest record (offset 0) moves to newest through compaction)
Steady state then short loss of comms UC ST-0.7
Steady State (near newest record no compaction)
Loss
Kill pipeline
Steady state then long loss of comms UC ST-0.8
Connect
Steady State (near newest record no compaction)
Loss
Kill pipeline
Force realignment after long loss UC ST-0.8 and 0.9
Connect > Tombstone
Force oldest record (offset 0) moves to newest through compaction)
Steady State UC ST-0.3
Steady State (near newest record no compaction)

Figure 33 Phases of interaction for Use Cases

7.10 Use cases beyond current release

Beyond the current release there will be further improvements including the ability to dynamically adjust the stream behavior and the feeds to the stream as follows:

- Adjustment of compaction delay and tombstone-retention on-the-fly
    - Allows for tuning of stream behavior at initial start-up and during predictable comms failures.
- Building and adjusting the context (creation, expansion, contraction and deletion)
    - Allows for multiple clients with differing needs and security clearances etc.
        - Negotiate Context opportunities based upon policy and client role etc.
        - Build explicit context (including topology, nodes, links etc.)
            - Various interactions to set up intent for nodes and links that themselves need to be realized in the underlying structure.
        - Note that the initial requests are in terms of shared knowledge such as city or building location and generalized termination points/flows with minimal technology detail.
    - In the general solution, with TAPI feeding a client, there could be several alternative contexts that can be provided. Contexts may focus on a single layer or layer grouping or on a region of the network etc. In a more sophisticated solution where there are many clients each with a slice, in these solutions various negotiations would be required to agree and form the context.
    - Note: Currently, the context is defined by a default intent where there was no opportunity for the client to express the context intent over an interface.
- Taking advantage of context adjustment capabilities to increase and decrease the intensity of view of information. This is applied where the intensity could not be handled across the whole context and hence is a focus. This may be where the parameter changes very often.
    - Spotlighting: Allows the client to selectively increase the fidelity of measurements by changing the measurement policy for a specific property and/or by including an instance of a property in the context where that property is usually not monitored
    - Single snapshot: Allows the client to select a property to take a momentary view of via the stream. This may be the capturing of a single counter value where that counter changes very often (e.g., a packet counter) such that streaming of the raw value would be excessive even for a single measure.

There are clearly other potential applications of a streaming solution where there are:

1. Many direct clients using the same context.
    - Here some form of multi-cast stream with a message broker or other multi-cast mechanism may be appropriate.
    - In the cases where there is a broker, it may be appropriate to continue to use the compacted log, but this will only benefit the broker and the provider will not be aware of clients or client performance challenges.
    - There are no apparent specific applications in a telecommunications network context.
2. Many direct clients each with different context
    - This leads to a characteristic very similar to that considered in this document.
    - The distinction is the multiple contexts.
    - It is likely that context build/modify will be necessary to enable this capability.
    - An application example is presentation of a slice to its client where each client has its own slice.
3. Short-lived clients that are attached for only a short period and want a specific context.
    - It is likely, for this case, that there will also be multiple clients.
    - It is possible that the short-lived clients define a short-term context and align with this.

    - A possible case is a controller component dedicated to a particular test activity where the provider presents a context related to the test and when the test is complete the provider deletes the context.
    - Another possible case is a GUI. This does have a short-term cache and may benefit from ongoing updates even if only for a short period.
4. Clients that do not have a cache or store of information acquired from the provider.
    - A solution that does not store information cannot be expected to take significant advantage of a stream that provides information on changes.
    - However, as the stream can be used to gain current state and the context can be set to define the relevant things that current state is required for, then a one-shot compacted log stream could be used to get a temporary snapshot.
    - A human driven CLI is an example, and this does not normally provide asynchronous delivery. In general, TAPI is not oriented towards this case.
        - A human tends to want to make somewhat random, complex and unbounded queries.

Cases 1-3 appear to benefit from the streaming capabilities described in this document, whereas case 4 does not, but it also does not seem to be a relevant TAPI application.

For each consideration in this section, it will be necessary to enhance the expression of capability such that the client can know what opportunities for adjustment are available. It is expected that this will be expressed using machine interpretable specifications.

7.11 Message approach (WebSocket example)

7.11.1 Basic interaction

The messaging is as defined below:

- The general structure for the WebSocket URL is obtained from the connection-address field of the discovered available streams. The implementations MAY support WS and WSS. For example, the URI can be of the form :
“`wss://<host>/tapi/data/context/stream-context/available-stream=<uuid>`”

  where the uuid is acquired through a RESTCONF GET operation of available-stream (s). Using this URL would start the stream from the oldest record.

- Considering the "Connect (token)" from the message sequence diagram, this becomes “`wss://<host>/tapi/data/context/stream-context/available-stream=<uuid>?start_from=<token>`”. Omitting the token causes the provider to start from offset zero (i.e., the oldest record).
- In some cases, it may be relevant to start from the latest (e.g., for non-compacted logs) “`wss://<host>/tapi/data/context/stream-context/available-stream=<uuid>?start_from=latest`”.

7.11.2 Authorization example – WebSockets

The WebSockets specification indicates that the WebSockets server can use any client authentication mechanism available to a generic HTTP server ([RFC6455-AUTH]).

Use of the authorization framework defined in [RFC6750] is recommended.

A valid authentication token for the provider must be supplied by the client via the use of "Authorization: Bearer <token>". This must be supplied in the header of the WebSockets connection request.

The authentication token is obtained from the provider using an appropriate method. This would supply a token for the specific username (i.e., the client).

7.11.3 Connecting to a stream example - WebSockets

This token is then used in the WebSockets handshake:

```
GET /tapi/data/context/stream-context/available-stream=<uuid> HTTP/1.1
Upgrade: websocket
Connection: Upgrade
Host: <host name>
Origin: http://<host name>
Sec-WebSocket-Key: <key>
Sec-WebSocket-Version: 13
Authorization: Bearer <authentication token>
```

The provider will provide a response similar to:

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Sec-WebSocket-Accept: eiUnNdCyox5gJ7eAbD4ZNo2H4xY=
Date: Mon, 07 Sep 2020 11:57:08 GMT
Connection: upgrade
Strict-Transport-Security: max-age=31536000; includeSubDomains
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
Content-Security-Policy: default-src 'self' data: mediastream: blob: filesystem:
'unsafe-inline' 'unsafe-eval'
```

Followed by stream messages.

8 Appendix

8.1 Appendix – Considering compacted logs

The following section considers Kafka as an example implementation of a compacted log and then discusses implications of compaction and some storage strategies.

8.1.1 Essential characteristics of a compacted log

Compaction is described in the Kafka documentation at [KAFKA-COMP]

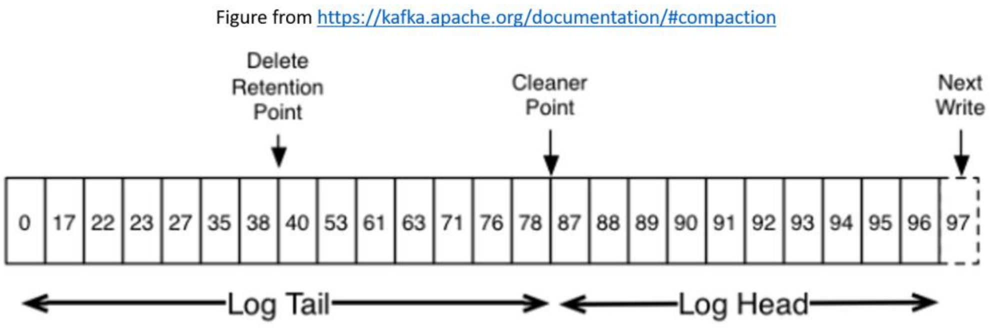


Figure 34 Kafka compaction

With retention (for non-deletes) set to "FOREVER", the log becomes a single source of truth for absolute state (eventual consistency) and change of state (cost effective fidelity).

Client essentially reading next record in sequence:

- To the right of the Cleaner Point ensures full fidelity
- Between cleaner point and delete retention (tombstone-retention) points provides reduced fidelity but still supports eventual consistency
- To the left of (before) the delete retention (tombstone-retention) point potentially violates eventual consistency and requires the client to go back to read record offset zero

8.1.2 Order of events

- Disordering of records
  - In a distributed system, information from the various parts is received with varying delay such that it is likely to be out of order
  - It can be assumed that time of day is well synchronized across the network
  - Event order can be regenerated (within reason) based upon time of event at source
  - Critical ordering that should be preserved through the log and pipeline is that related to each single event source. For example, consider an alarm detector.
    - It is possible that the time granularity at the source is not sufficient to resolve the active-clear sequence when cycling is very rapid as they can both appear to be at the same recorded time.

        - If the detector goes active and then clear, that ordering should be preserved through the system such that time granularity problems are not encountered, so that the view of system state is always eventually consistent with the state of the controlled system
- Multiple receipts of the same record: Idempotency
    - A record received more than once should not have any impact on system behavior

8.1.3 Log segments

A log may be divided into segments (units of storage) where the segments may have a defined maximum size. Records will be stored in an active segment until it becomes full at which point a new segment will be created for new records (a new active segment) and the previously active segment will become inactive. A log of this form will comprise a sequence of inactive segments and one active segment.

8.1.4 Partitions

To improve scaling a stream may be fed from a conceptual log that is formed from a number of separate logs where those separate logs are considered as partitions of the single conceptual log feeding the stream. The partitions have to be such as to maintain integrity of the flow with respect to the order of occurrence for any particular entity instance. Each partition has its own sequence numbering. The resource allocation to each partition and the load on the partitions is likely to be different. There is no particular order in which partitions may feed the stream.

8.1.5 Compaction in a real implementation

Considering compaction delay, in general system load will cause the compaction to sometimes drift such that less compaction occurs than is ideal.

In Kafka, compaction does not operate at a fixed cleaner point as the head (active) segment is not compacted. When the head rolls to become a tail (inactive) segment compaction can happen but may be delayed. The behavior is not fully deterministic as it depends upon segment fill and intermittency occurrence.

8.1.6 The TOMBSTONE record

The TOMBSTONE record is a light-weight record that indicates that the identified entity is no longer in existence. The TOMBSTONE record is used to ensure eventual consistency during a realignment after a long communications failure. When the client receives a TOMBSTONE record, it should remove any record of the entity identified.

8.2 Appendix – UML Model

The YANG model for streaming has been generated using the Eagle tooling for the streaming UML model. The UML diagrams provide a convenient overview of the streaming structure and relevant properties. The key UML diagrams for streaming are provided in this section.

The conversion from UML to YANG accounts for various considerations including the change of formats and notations. For example, in UML camel case is used whereas in YANG uses kebab case (hyphenated lower case). The diagrams should be read with this in mind.

The figure below shows the structure of the streaming model. This structural view may help when reviewing the YANG representation.

8.2.1 UML model for streaming inventory and alarms

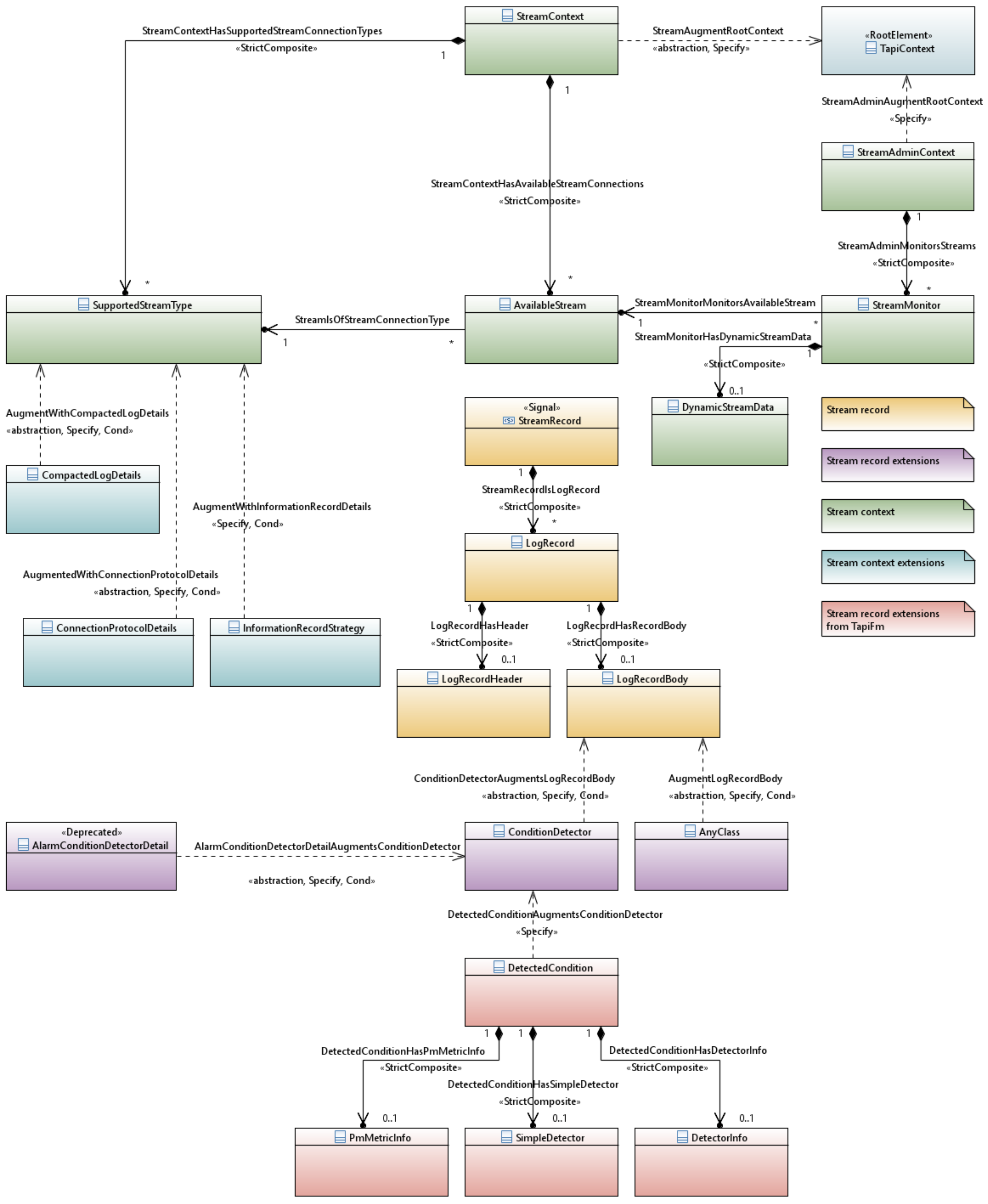
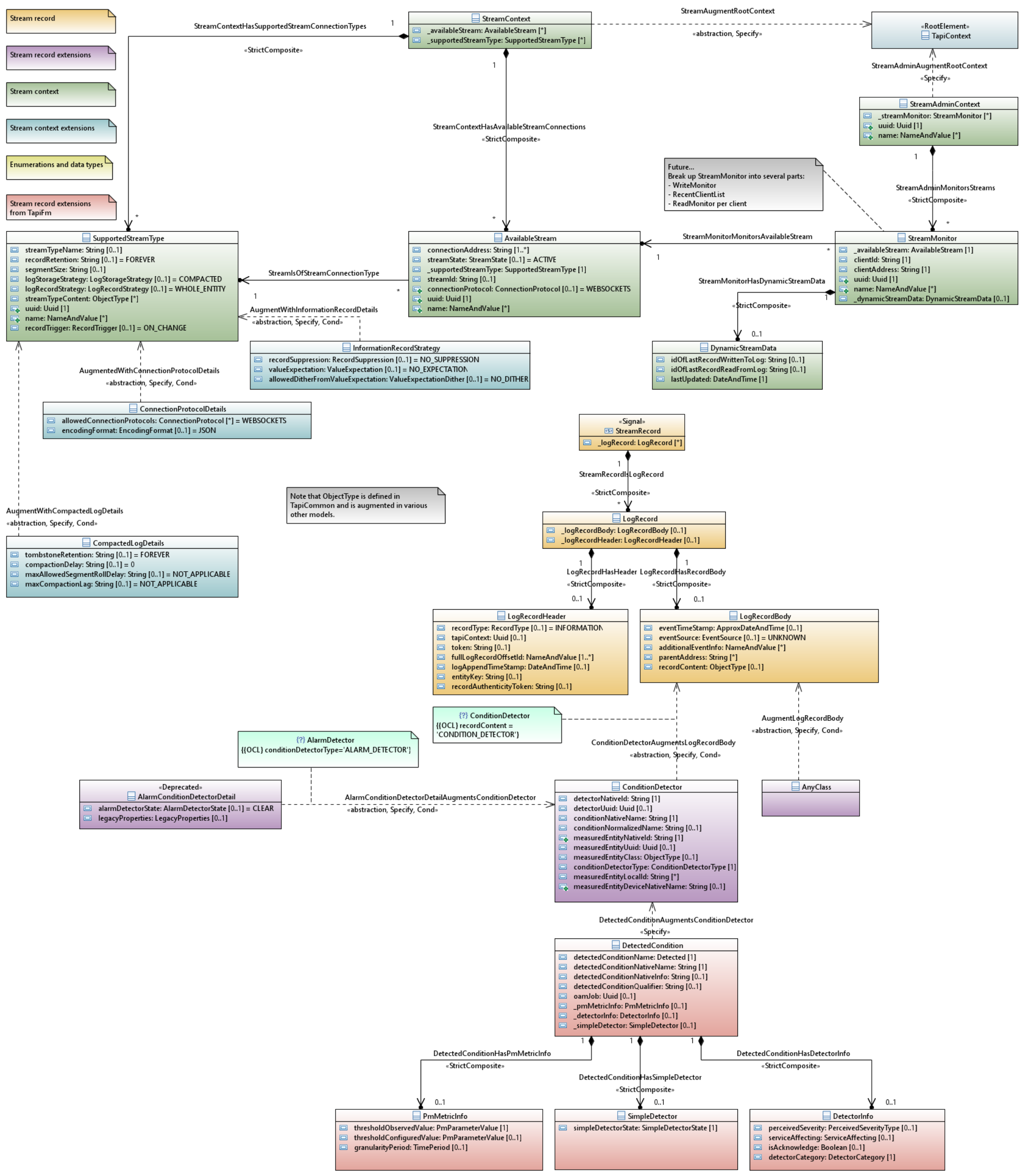

StreamContext
«RootElement» TapiContext
StreamContextHasSupportedStreamConnectionTypes «StrictComposite»
StreamAugmentRootContext «abstraction, Specify»
StreamAdminAugmentRootContext «Specify»
StreamAdminContext
StreamContextHasAvailableStreamConnections «StrictComposite»
StreamAdminMonitorsStreams «StrictComposite»
SupportedStreamType
AvailableStream
StreamMonitor
StreamIsOfStreamConnectionType
StreamMonitorMonitorsAvailableStream
StreamMonitorHasDynamicStreamData «StrictComposite»
DynamicStreamData
«Signal» StreamRecord
AugmentWithCompactedLogDetails «abstraction, Specify, Cond»
CompactedLogDetails
AugmentWithInformationRecordDetails «Specify, Cond»
AugmentedWithConnectionProtocolDetails «abstraction, Specify, Cond»
ConnectionProtocolDetails
InformationRecordStrategy
StreamRecordIsLogRecord «StrictComposite»
LogRecord
LogRecordHasHeader «StrictComposite»
LogRecordHasRecordBody «StrictComposite»
LogRecordHeader
LogRecordBody
ConditionDetectorAugmentsLogRecordBody «abstraction, Specify, Cond»
AugmentLogRecordBody «abstraction, Specify, Cond»
«Deprecated» AlarmConditionDetectorDetail
AlarmConditionDetectorDetailAugmentsConditionDetector «abstraction, Specify, Cond»
ConditionDetector
AnyClass
DetectedConditionAugmentsConditionDetector «Specify»
DetectedCondition
DetectedConditionHasPmMetricInfo «StrictComposite»
DetectedConditionHasSimpleDetector «StrictComposite»
DetectedConditionHasDetectorInfo «StrictComposite»
PmMetricInfo
SimpleDetector
DetectorInfo
Stream record
Stream record extensions
Stream context
Stream context extensions
Stream record extensions from TapiFm

Figure 35 Structure of the streaming model

The figure below shows key content of the classes shown in the structure above.



Stream record
Stream record extensions
Stream context
Stream context extensions
Enumerations and data types
Stream record extensions from TapiFm
StreamContextHasSupportedStreamConnectionTypes
«StrictComposite»
StreamContext
_availableStream: AvailableStream [*]
_supportedStreamType: SupportedStreamType [*]
StreamAugmentRootContext
«abstraction, Specify»
«RootElement»
TapiContext
StreamAdminAugmentRootContext
«Specify»
StreamAdminContext
_streamMonitor: StreamMonitor [*]
uuid: Uuid [1]
name: NameAndValue [*]
StreamContextHasAvailableStreamConnections
«StrictComposite»
Future...
Break up StreamMonitor into several parts:
- WriteMonitor
- RecentClientList
- ReadMonitor per client
StreamAdminMonitorsStreams
«StrictComposite»
SupportedStreamType
streamTypeName: String [0..1]
recordRetention: String [0..1] = FOREVER
segmentSize: String [0..1]
logStorageStrategy: LogStorageStrategy [0..1] = COMPACTED
logRecordStrategy: LogRecordStrategy [0..1] = WHOLE_ENTITY
streamTypeContent: ObjectType [*]
uuid: Uuid [1]
name: NameAndValue [*]
recordTrigger: RecordTrigger [0..1] = ON_CHANGE
StreamIsOfStreamConnectionType
AvailableStream
connectionAddress: String [1..*]
streamState: StreamState [0..1] = ACTIVE
_supportedStreamType: SupportedStreamType [1]
streamId: String [0..1]
connectionProtocol: ConnectionProtocol [0..1] = WEBSOCKETS
uuid: Uuid [1]
name: NameAndValue [*]
StreamMonitorMonitorsAvailableStream
StreamMonitor
_availableStream: AvailableStream [1]
clientId: String [1]
clientAddress: String [1]
uuid: Uuid [1]
name: NameAndValue [*]
_dynamicStreamData: DynamicStreamData [0..1]
StreamMonitorHasDynamicStreamData
«StrictComposite»
AugmentWithInformationRecordDetails
«abstraction, Specify, Cond»
InformationRecordStrategy
recordSuppression: RecordSuppression [0..1] = NO_SUPPRESSION
valueExpectation: ValueExpectation [0..1] = NO_EXPECTATION
allowedDitherFromValueExpectation: ValueExpectationDither [0..1] = NO_DITHER
DynamicStreamData
idOfLastRecordWrittenToLog: String [0..1]
idOfLastRecordReadFromLog: String [0..1]
lastUpdated: DateAndTime [1]
AugmentedWithConnectionProtocolDetails
«abstraction, Specify, Cond»
ConnectionProtocolDetails
allowedConnectionProtocols: ConnectionProtocol [*] = WEBSOCKETS
encodingFormat: EncodingFormat [0..1] = JSON
«Signal»
StreamRecord
_logRecord: LogRecord [*]
StreamRecordHasLogRecord
«StrictComposite»
Note that ObjectType is defined in TapiCommon and is augmented in various other models.
AugmentWithCompactedLogDetails
«abstraction, Specify, Cond»
CompactedLogDetails
tombstoneRetention: String [0..1] = FOREVER
compactionDelay: String [0..1] = 0
maxAllowedSegmentRollDelay: String [0..1] = NOT_APPLICABLE
maxCompactionLag: String [0..1] = NOT_APPLICABLE
LogRecord
_logRecordBody: LogRecordBody [0..1]
_logRecordHeader: LogRecordHeader [0..1]
LogRecordHasHeader
«StrictComposite»
LogRecordHasRecordBody
«StrictComposite»
LogRecordHeader
recordType: RecordType [0..1] = INFORMATION
tapiContext: Uuid [0..1]
token: String [0..1]
fullLogRecordOffsetId: NameAndValue [1..*]
logAppendTimeStamp: DateAndTime [0..1]
entityKey: String [0..1]
recordAuthenticityToken: String [0..1]
LogRecordBody
eventTimeStamp: ApproxDateAndTime [0..1]
eventSource: EventSource [0..1] = UNKNOWN
additionalEventInfo: NameAndValue [*]
parentAddress: String [*]
recordContent: ObjectType [0..1]
{?} ConditionDetector
{{OCL} recordContent = 'CONDITION_DETECTOR'}
{?} AlarmDetector
{{OCL} conditionDetectorType='ALARM_DETECTOR'}
AugmentLogRecordBody
«abstraction, Specify, Cond»
ConditionDetectorAugmentsLogRecordBody
«abstraction, Specify, Cond»
«Deprecated»
AlarmConditionDetectorDetail
alarmDetectorState: AlarmDetectorState [0..1] = CLEAR
legacyProperties: LegacyProperties [0..1]
AlarmConditionDetectorDetailAugmentsConditionDetector
«abstraction, Specify, Cond»
ConditionDetector
detectorNativeId: String [1]
detectorUuid: Uuid [0..1]
conditionNativeName: String [1]
conditionNormalizedName: String [0..1]
measuredEntityNativeId: String [1]
measuredEntityUuid: Uuid [0..1]
measuredEntityClass: ObjectType [0..1]
conditionDetectorType: ConditionDetectorType [1]
measuredEntityLocalId: String [*]
measuredEntityDeviceNativeName: String [0..1]
AnyClass
DetectedConditionAugmentsConditionDetector
«Specify»
DetectedCondition
detectedConditionName: Detected [1]
detectedConditionNativeName: String [1]
detectedConditionNativeInfo: String [0..1]
detectedConditionQualifier: String [0..1]
oamJob: Uuid [0..1]
_pmMetricInfo: PmMetricInfo [0..1]
_detectorInfo: DetectorInfo [0..1]
_simpleDetector: SimpleDetector [0..1]
DetectedConditionHasPmMetricInfo
«StrictComposite»
DetectedConditionHasSimpleDetector
«StrictComposite»
DetectedConditionHasDetectorInfo
«StrictComposite»
PmMetricInfo
thresholdObservedValue: PmParameterValue [1]
thresholdConfiguredValue: PmParameterValue [0..1]
granularityPeriod: TimePeriod [0..1]
SimpleDetector
simpleDetectorState: SimpleDetectorState [1]
DetectorInfo
perceivedSeverity: PerceivedSeverityType [0..1]
serviceAffecting: ServiceAffecting [0..1]
isAcknowledge: Boolean [0..1]
detectorCategory: DetectorCategory [1]

Figure 36 Structure and content of the streaming model

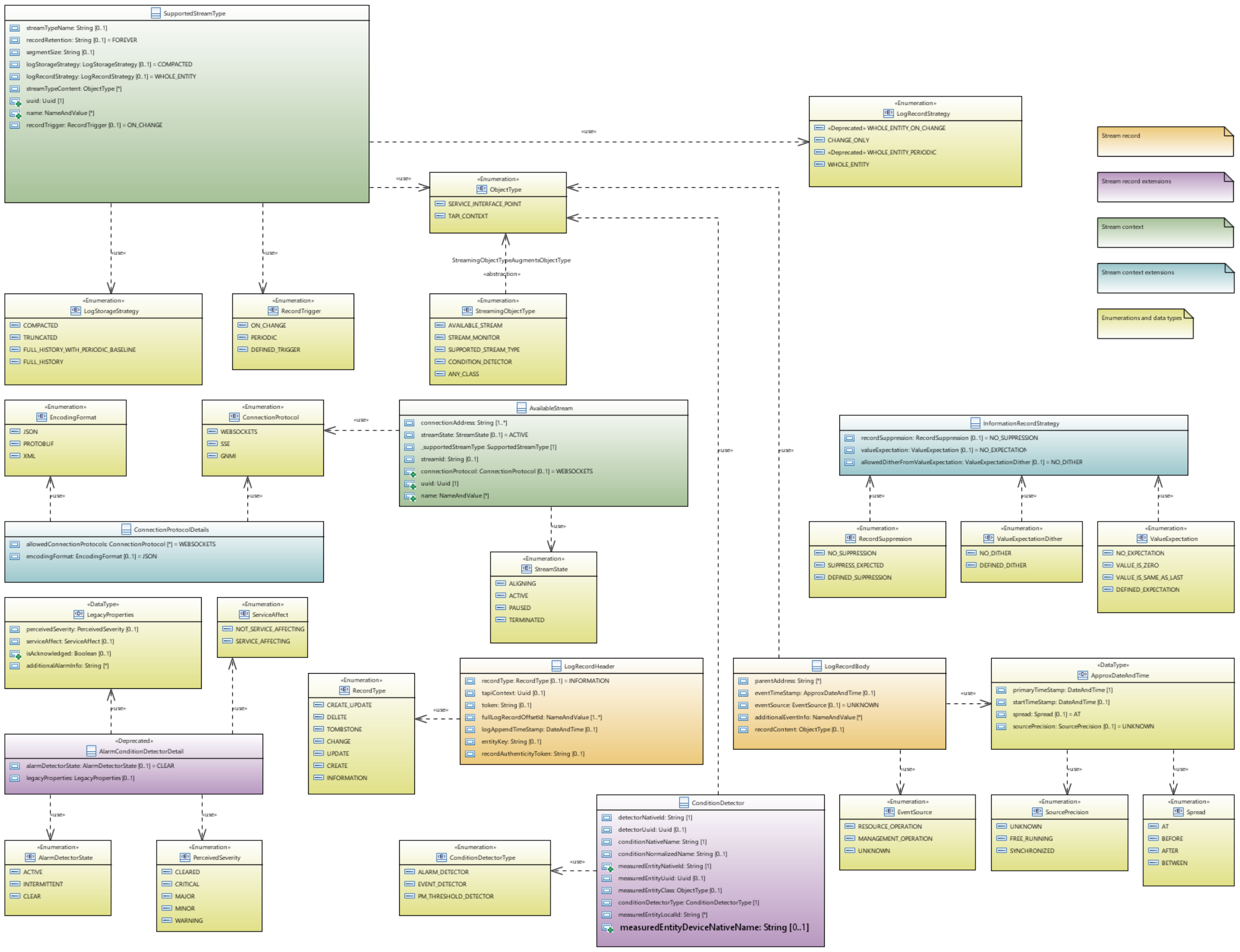

SupportedStreamType
streamTypeName: String [0..1]
recordRetention: String [0..1] = FOREVER
segmentSize: String [0..1]
logStorageStrategy: LogStorageStrategy [0..1] = COMPACTED
logRecordStrategy: LogRecordStrategy [0..1] = WHOLE_ENTITY
streamTypeContent: ObjectType [*]
uuid: Uuid [1]
name: NameAndValue [*]
recordTrigger: RecordTrigger [0..1] = ON_CHANGE
«use»
«Enumeration»
LogRecordStrategy
«Deprecated» WHOLE_ENTITY_ON_CHANGE
CHANGE_ONLY
«Deprecated» WHOLE_ENTITY_PERIODIC
WHOLE_ENTITY
Stream record
Stream record extensions
Stream context
Stream context extensions
Enumerations and data types
«use»
«Enumeration»
ObjectType
SERVICE_INTERFACE_POINT
TAPI_CONTEXT
StreamingObjectTypeAugmentsObjectType
«abstraction»
«use»
«use»
«Enumeration»
LogStorageStrategy
COMPACTED
TRUNCATED
FULL_HISTORY_WITH_PERIODIC_BASELINE
FULL_HISTORY
«Enumeration»
RecordTrigger
ON_CHANGE
PERIODIC
DEFINED_TRIGGER
«Enumeration»
StreamingObjectType
AVAILABLE_STREAM
STREAM_MONITOR
SUPPORTED_STREAM_TYPE
CONDITION_DETECTOR
ANY_CLASS
«Enumeration»
EncodingFormat
JSON
PROTOBUF
XML
«Enumeration»
ConnectionProtocol
WEBSOCKETS
SSE
GNMI
«use»
AvailableStream
connectionAddress: String [1..*]
streamState: StreamState [0..1] = ACTIVE
_supportedStreamType: SupportedStreamType [1]
streamId: String [0..1]
connectionProtocol: ConnectionProtocol [0..1] = WEBSOCKETS
uuid: Uuid [1]
name: NameAndValue [*]
«use»
«use»
InformationRecordStrategy
recordSuppression: RecordSuppression [0..1] = NO_SUPPRESSION
valueExpectation: ValueExpectation [0..1] = NO_EXPECTATION
allowedDitherFromValueExpectation: ValueExpectationDither [0..1] = NO_DITHER
«use»
«use»
«use»
«use»
«use»
ConnectionProtocolDetails
allowedConnectionProtocols: ConnectionProtocol [*] = WEBSOCKETS
encodingFormat: EncodingFormat [0..1] = JSON
«use»
«Enumeration»
StreamState
ALIGNING
ACTIVE
PAUSED
TERMINATED
«Enumeration»
RecordSuppression
NO_SUPPRESSION
SUPPRESS_EXPECTED
DEFINED_SUPPRESSION
«Enumeration»
ValueExpectationDither
NO_DITHER
DEFINED_DITHER
«Enumeration»
ValueExpectation
NO_EXPECTATION
VALUE_IS_ZERO
VALUE_IS_SAME_AS_LAST
DEFINED_EXPECTATION
«DataType»
LegacyProperties
perceivedSeverity: PerceivedSeverity [0..1]
serviceAffect: ServiceAffect [0..1]
isAcknowledged: Boolean [0..1]
additionalAlarmInfo: String [*]
«Enumeration»
ServiceAffect
NOT_SERVICE_AFFECTING
SERVICE_AFFECTING
«Enumeration»
RecordType
CREATE_UPDATE
DELETE
TOMBSTONE
CHANGE
UPDATE
CREATE
INFORMATION
«use»
LogRecordHeader
recordType: RecordType [0..1] = INFORMATION
tapiContext: Uuid [0..1]
token: String [0..1]
fullLogRecordOffsetId: NameAndValue [1..*]
logAppendTimeStamp: DateAndTime [0..1]
entityKey: String [0..1]
recordAuthenticityToken: String [0..1]
LogRecordBody
parentAddress: String [*]
eventTimeStamp: ApproxDateAndTime [0..1]
eventSource: EventSource [0..1] = UNKNOWN
additionalEventInfo: NameAndValue [*]
recordContent: ObjectType [0..1]
«use»
«DataType»
ApproxDateAndTime
primaryTimeStamp: DateAndTime [1]
startTimeStamp: DateAndTime [0..1]
spread: Spread [0..1] = AT
sourcePrecision: SourcePrecision [0..1] = UNKNOWN
«use»
«use»
«Deprecated»
AlarmConditionDetectorDetail
alarmDetectorState: AlarmDetectorState [0..1] = CLEAR
legacyProperties: LegacyProperties [0..1]
«use»
«use»
«use»
«use»
«use»
ConditionDetector
detectorNativeId: String [1]
detectorUuid: Uuid [0..1]
conditionNativeName: String [1]
conditionNormalizedName: String [0..1]
measuredEntityNativeId: String [1]
measuredEntityUuid: Uuid [0..1]
measuredEntityClass: ObjectType [0..1]
conditionDetectorType: ConditionDetectorType [1]
measuredEntityLocalId: String [*]
measuredEntityDeviceNativeName: String [0..1]
«Enumeration»
EventSource
RESOURCE_OPERATION
MANAGEMENT_OPERATION
UNKNOWN
«Enumeration»
SourcePrecision
UNKNOWN
FREE_RUNNING
SYNCHRONIZED
«Enumeration»
Spread
AT
BEFORE
AFTER
BETWEEN
«Enumeration»
AlarmDetectorState
ACTIVE
INTERMITTENT
CLEAR
«Enumeration»
PerceivedSeverity
CLEARED
CRITICAL
MAJOR
MINOR
WARNING
«use»
«Enumeration»
ConditionDetectorType
ALARM_DETECTOR
EVENT_DETECTOR
PM_THRESHOLD_DETECTOR

Figure 37 Datatypes of the streaming model

The figure below shows the UML form of augmentation (using the <<specify>> stereotype). All classes in the model can augment LogRecordBody.

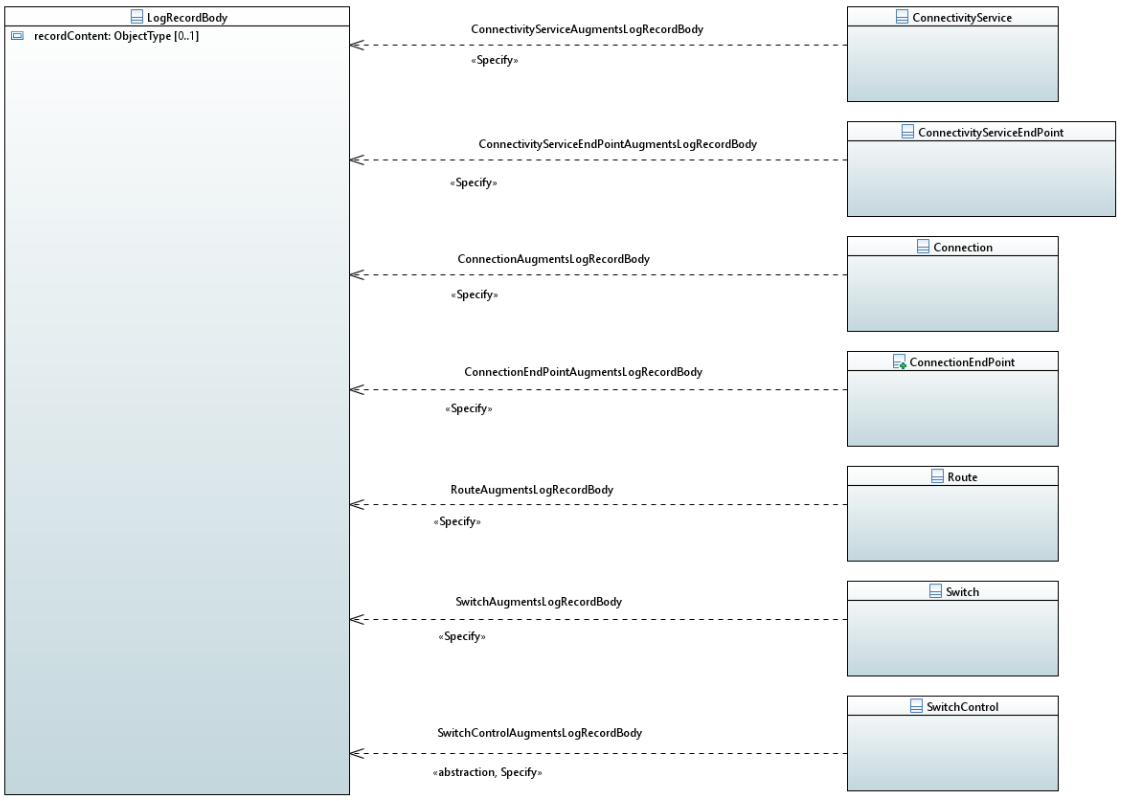

LogRecordBody
recordContent: ObjectType [0..1]
ConnectivityServiceAugmentsLogRecordBody
«Specify»
ConnectivityService
ConnectivityServiceEndPointAugmentsLogRecordBody
«Specify»
ConnectivityServiceEndPoint
ConnectionAugmentsLogRecordBody
«Specify»
Connection
ConnectionEndPointAugmentsLogRecordBody
«Specify»
ConnectionEndPoint
RouteAugmentsLogRecordBody
«Specify»
Route
SwitchAugmentsLogRecordBody
«Specify»
Switch
SwitchControlAugmentsLogRecordBody
«abstraction, Specify»
SwitchControl

Figure 38 Example of Augmentation of the LogRecordBody with some classes from the model

8.2.2 The model of gNMI protobuf

The TAPI solution includes a UML model fragment that drives the YANG and represents the protobuf structure defined in [GNMI-PROTO] use in the gNMI streaming capability for measurement reporting. The structure augments the stream record which is the container for all forms of TAPI streaming and which essentially has no content or structure for gNMI streaming. This is shown in the UML fragment below.

Whilst the gNMI structure, starting from SubscribeResponse to TypedValue is not a TAPI model, it is necessary to drive the appropriate YANG to drive the construction/generation of the protobuf message structures[78]. The gNMI model allows various other data types in structures. Only those that are used are shown. TypedValue has various different definitions for the value in a "one of" structure. Only the proto_bytes value is used (this is enumeration 13 in the protobuf definition). This option enables any defined protobuf structure to be added (augmented) from this point. This is where the TAPI structure is added.

Each property shows the protobuf enumeration.

[78] Protobuf encoding is currently manual.

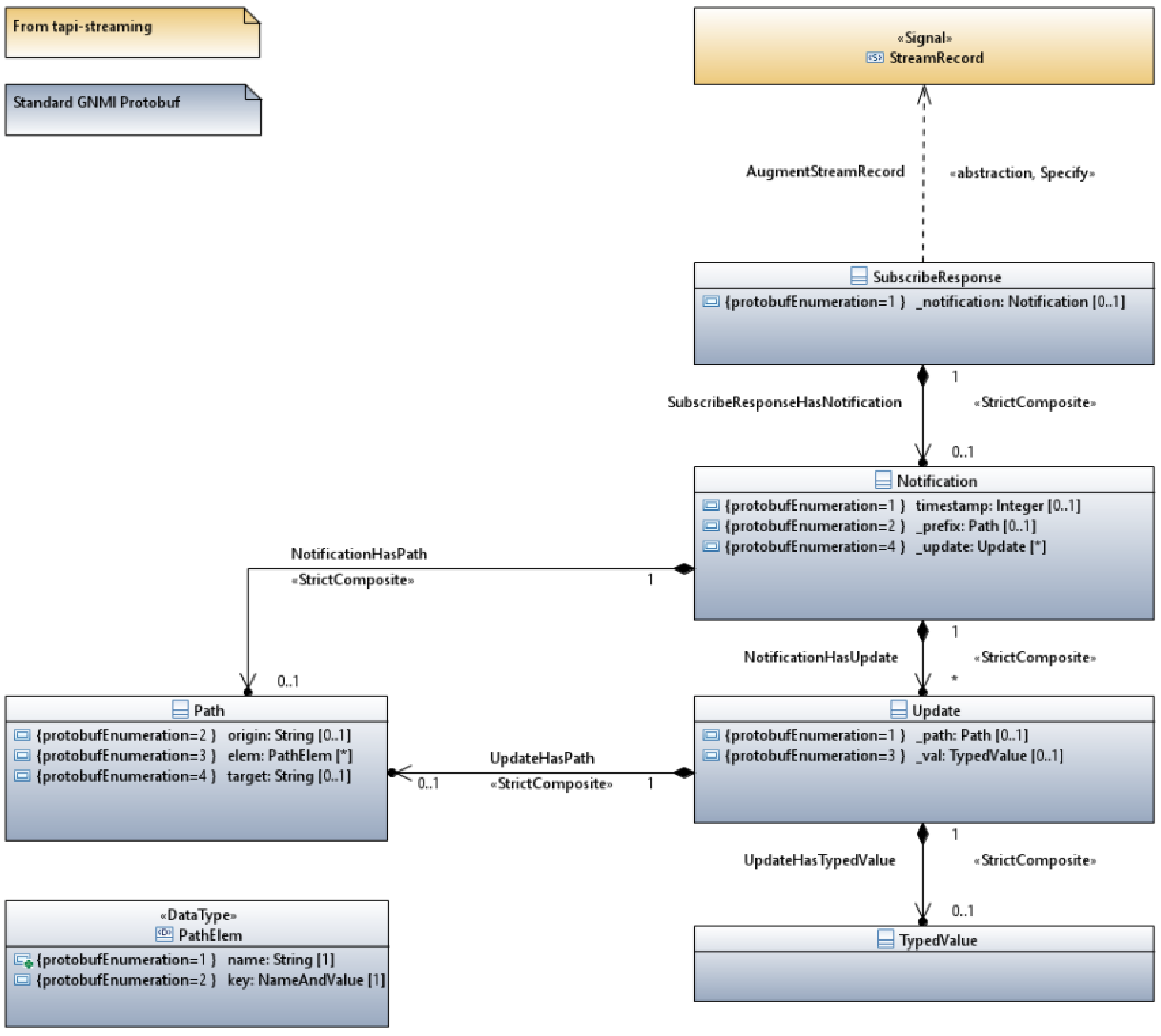

From tapi-streaming
Standard GNMI Protobuf
«Signal» StreamRecord
AugmentStreamRecord
«abstraction, Specify»
SubscribeResponse
{protobufEnumeration=1 } _notification: Notification [0..1]
SubscribeResponseHasNotification
«StrictComposite»
1
0..1
Notification
{protobufEnumeration=1 } timestamp: Integer [0..1]
{protobufEnumeration=2 } _prefix: Path [0..1]
{protobufEnumeration=4 } _update: Update [*]
NotificationHasPath
«StrictComposite»
1
0..1
NotificationHasUpdate
«StrictComposite»
1
*
Path
{protobufEnumeration=2 } origin: String [0..1]
{protobufEnumeration=3 } elem: PathElem [*]
{protobufEnumeration=4 } target: String [0..1]
UpdateHasPath
«StrictComposite»
0..1
1
Update
{protobufEnumeration=1 } _path: Path [0..1]
{protobufEnumeration=3 } _val: TypedValue [0..1]
UpdateHasTypedValue
«StrictComposite»
1
0..1
«DataType» PathElem
{protobufEnumeration=1 } name: String [1]
{protobufEnumeration=2 } key: NameAndValue [1]
TypedValue

Figure 39 Standard gNMI stream structure

The UML model below shows the definition of the structure of protoBytes. The proto_bytes property (protobufEnumeration =13) of the TypedValue has the protobuf structure defined by the remainder of the model in the fragment starting from StreamDetails. This is inserted directly into the TypedValue via the «ExtendedComposite» association. The structure allows for various StreamStructures. Only one structure is defined for MeasurementDetails.

The MeasurementDetails is constructed in two parts. This allows qualified measurements to be inserted at protobufEnumeration 11. This illustrates the approach to adding nested blocks of data in general.

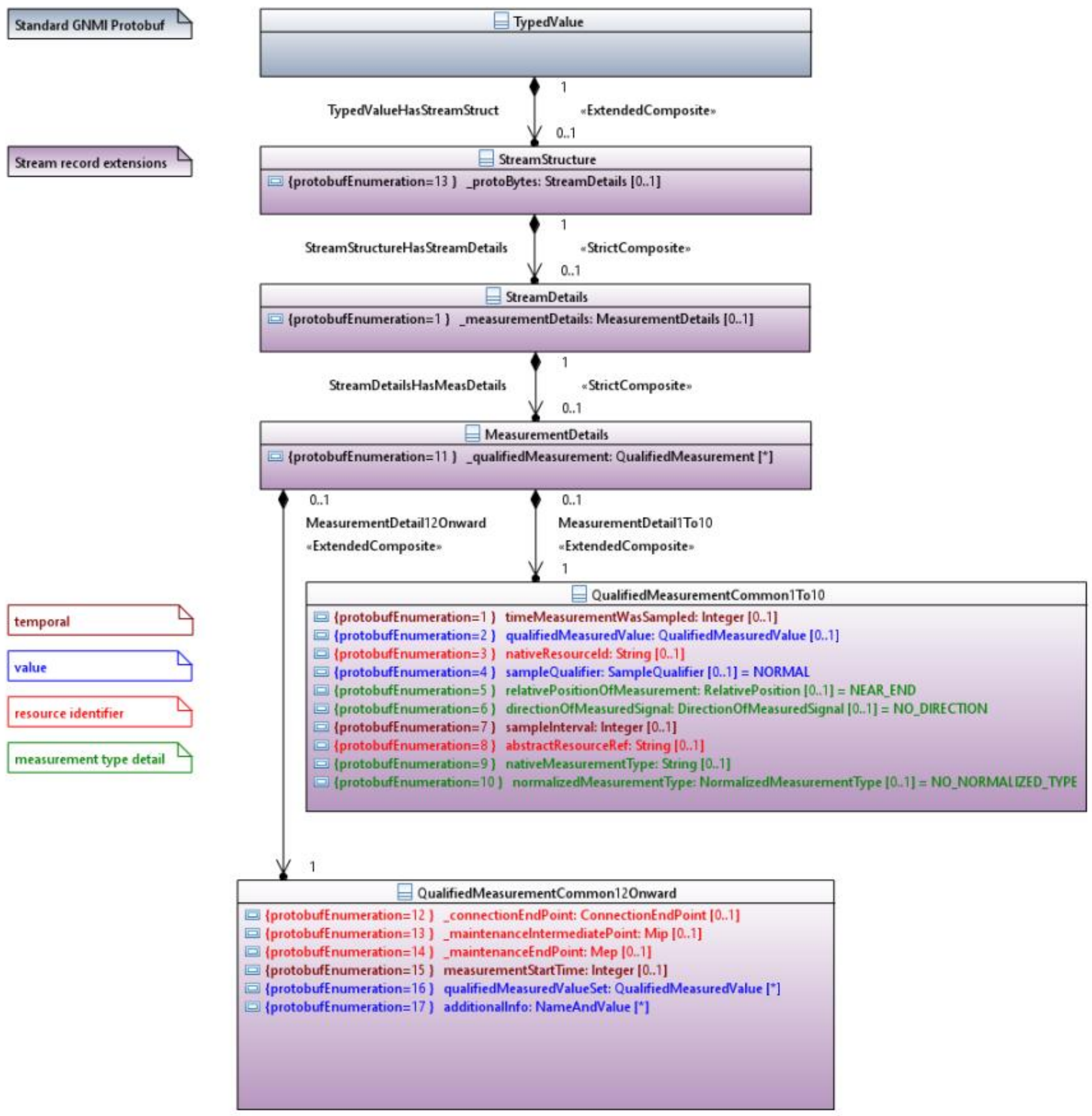

Standard GNMI Protobuf
TypedValue
1
TypedValueHasStreamStruct
«ExtendedComposite»
0..1
Stream record extensions
StreamStructure
{protobufEnumeration=13} _protoBytes: StreamDetails [0..1]
1
StreamStructureHasStreamDetails
«StrictComposite»
0..1
StreamDetails
{protobufEnumeration=1} _measurementDetails: MeasurementDetails [0..1]
1
StreamDetailsHasMeasDetails
«StrictComposite»
0..1
MeasurementDetails
{protobufEnumeration=11} _qualifiedMeasurement: QualifiedMeasurement [*]
0..1
MeasurementDetail12Onward
«ExtendedComposite»
0..1
MeasurementDetail1To10
«ExtendedComposite»
1
QualifiedMeasurementCommon1To10
{protobufEnumeration=1} timeMeasurementWasSampled: Integer [0..1]
{protobufEnumeration=2} qualifiedMeasuredValue: QualifiedMeasuredValue [0..1]
{protobufEnumeration=3} nativeResourceId: String [0..1]
{protobufEnumeration=4} sampleQualifier: SampleQualifier [0..1] = NORMAL
{protobufEnumeration=5} relativePositionOfMeasurement: RelativePosition [0..1] = NEAR_END
{protobufEnumeration=6} directionOfMeasuredSignal: DirectionOfMeasuredSignal [0..1] = NO_DIRECTION
{protobufEnumeration=7} sampleInterval: Integer [0..1]
{protobufEnumeration=8} abstractResourceRef: String [0..1]
{protobufEnumeration=9} nativeMeasurementType: String [0..1]
{protobufEnumeration=10} normalizedMeasurementType: NormalizedMeasurementType [0..1] = NO_NORMALIZED_TYPE
temporal
value
resource identifier
measurement type detail
1
QualifiedMeasurementCommon12Onward
{protobufEnumeration=12} _connectionEndPoint: ConnectionEndPoint [0..1]
{protobufEnumeration=13} _maintenanceIntermediatePoint: Mip [0..1]
{protobufEnumeration=14} _maintenanceEndPoint: Mep [0..1]
{protobufEnumeration=15} measurementStartTime: Integer [0..1]
{protobufEnumeration=16} qualifiedMeasuredValueSet: QualifiedMeasuredValue [*]
{protobufEnumeration=17} additionalInfo: NameAndValue [*]

Figure 40 Basic measurement augment

The model fragment above shows the basic PM structure supporting a single measurement per record where a single valued measurement is conveyed via qualifiedMeasuredValue chunks which are inserted directly into the MeasurementDetails structure via the «ExtendedComposite» association. If a measurement is multi-valued, the multiple values may be conveyed via qualifiedMeasuredValueSet. Each member of the set may be distinguished in the details of QualifiedMeasuredValue.

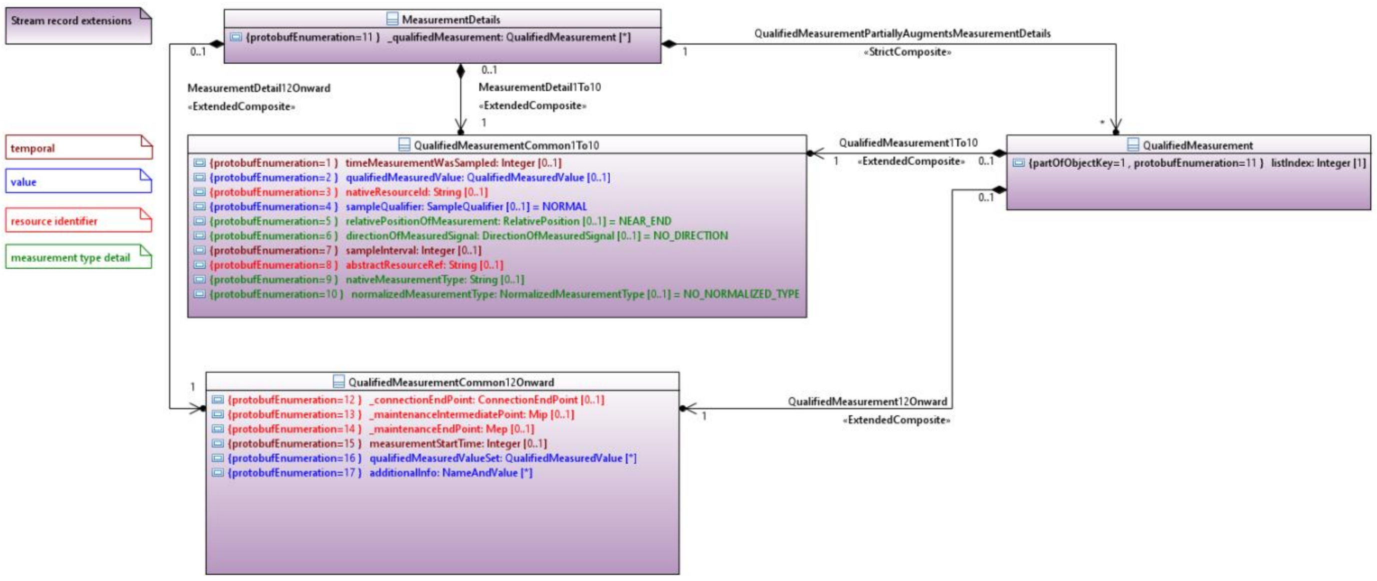

Stream record extensions
MeasurementDetails
{protobufEnumeration=11 } _qualifiedMeasurement: QualifiedMeasurement [*]
QualifiedMeasurementPartiallyAugmentsMeasurementDetails
«StrictComposite»
MeasurementDetail12Onward
«ExtendedComposite»
MeasurementDetail1To10
«ExtendedComposite»
temporal
value
resource identifier
measurement type detail
QualifiedMeasurementCommon1To10
{protobufEnumeration=1 } timeMeasurementWasSampled: Integer [0..1]
{protobufEnumeration=2 } qualifiedMeasuredValue: QualifiedMeasuredValue [0..1]
{protobufEnumeration=3 } nativeResourceId: String [0..1]
{protobufEnumeration=4 } sampleQualifier: SampleQualifier [0..1] = NORMAL
{protobufEnumeration=5 } relativePositionOfMeasurement: RelativePosition [0..1] = NEAR_END
{protobufEnumeration=6 } directionOfMeasuredSignal: DirectionOfMeasuredSignal [0..1] = NO_DIRECTION
{protobufEnumeration=7 } sampleInterval: Integer [0..1]
{protobufEnumeration=8 } abstractResourceRef: String [0..1]
{protobufEnumeration=9 } nativeMeasurementType: String [0..1]
{protobufEnumeration=10 } normalizedMeasurementType: NormalizedMeasurementType [0..1] = NO_NORMALIZED_TYPE
QualifiedMeasurement1To10
«ExtendedComposite»
QualifiedMeasurement
{partOfObjectKey=1 , protobufEnumeration=11 } listIndex: Integer [1]
QualifiedMeasurementCommon12Onward
{protobufEnumeration=12 } _connectionEndPoint: ConnectionEndPoint [0..1]
{protobufEnumeration=13 } _maintenanceIntermediatePoint: Mip [0..1]
{protobufEnumeration=14 } _maintenanceEndPoint: Mep [0..1]
{protobufEnumeration=15 } measurementStartTime: Integer [0..1]
{protobufEnumeration=16 } qualifiedMeasuredValueSet: QualifiedMeasuredValue [*]
{protobufEnumeration=17 } additionalInfo: NameAndValue [*]
QualifiedMeasurement12Onward
«ExtendedComposite»
0..1
1
*

Figure 41 Support for multiple measurements in one record

The model fragment above shows the full PM structure supporting multiple measurements in one record where measurements have some property values in common. Common values are defined in MeasurementDetails directly. Values that differ across the measurements are defined via the list in QualifiedMeasurement.

The model fragment shown below sets out the data types in detail.

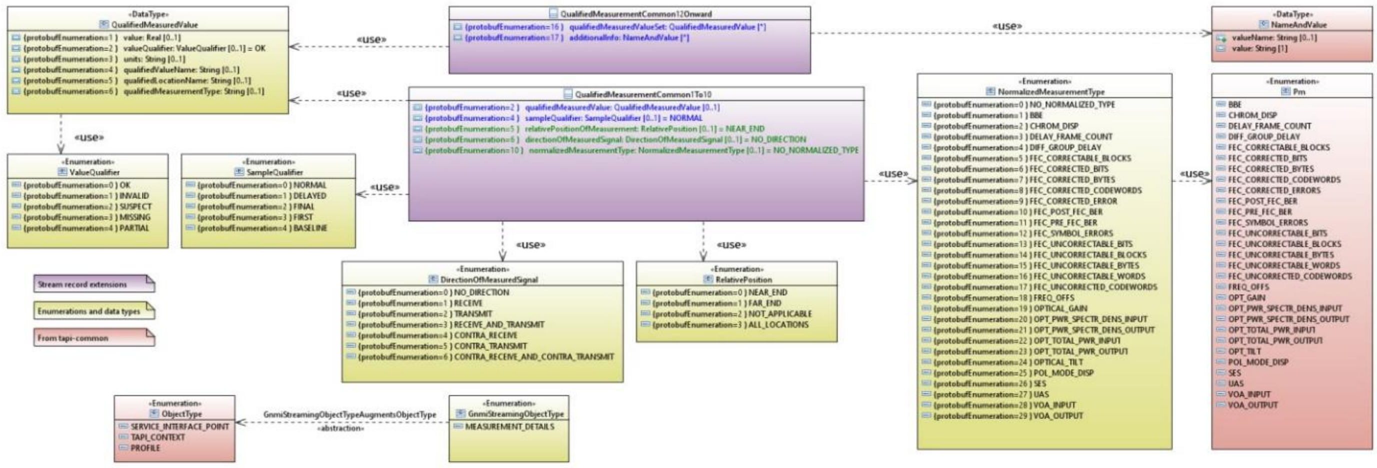

«DataType»
QualifiedMeasuredValue
{protobufEnumeration=1 } value: Real [0..1]
{protobufEnumeration=2 } valueQualifier: ValueQualifier [0..1] = OK
{protobufEnumeration=3 } units: String [0..1]
{protobufEnumeration=4 } qualifiedValueName: String [0..1]
{protobufEnumeration=5 } qualifiedLocationName: String [0..1]
{protobufEnumeration=6 } qualifiedMeasurementType: String [0..1]
QualifiedMeasurementCommon12Onward
{protobufEnumeration=16 } qualifiedMeasuredValueSet: QualifiedMeasuredValue [*]
{protobufEnumeration=17 } additionalInfo: NameAndValue [*]
«use»
«DataType»
NameAndValue
valueName: String [0..1]
value: String [1]
QualifiedMeasurementCommon1To10
{protobufEnumeration=2 } qualifiedMeasuredValue: QualifiedMeasuredValue [0..1]
{protobufEnumeration=4 } sampleQualifier: SampleQualifier [0..1] = NORMAL
{protobufEnumeration=5 } relativePositionOfMeasurement: RelativePosition [0..1] = NEAR_END
{protobufEnumeration=6 } directionOfMeasuredSignal: DirectionOfMeasuredSignal [0..1] = NO_DIRECTION
{protobufEnumeration=10 } normalizedMeasurementType: NormalizedMeasurementType [0..1] = NO_NORMALIZED_TYPE
«Enumeration»
ValueQualifier
{protobufEnumeration=0 } OK
{protobufEnumeration=1 } INVALID
{protobufEnumeration=2 } SUSPECT
{protobufEnumeration=3 } MISSING
{protobufEnumeration=4 } PARTIAL
«Enumeration»
SampleQualifier
{protobufEnumeration=0 } NORMAL
{protobufEnumeration=1 } DELAYED
{protobufEnumeration=2 } FINAL
{protobufEnumeration=3 } FIRST
{protobufEnumeration=4 } BASELINE
Stream record extensions
Enumerations and data types
From tapi-common
«Enumeration»
DirectionOfMeasuredSignal
{protobufEnumeration=0 } NO_DIRECTION
{protobufEnumeration=1 } RECEIVE
{protobufEnumeration=2 } TRANSMIT
{protobufEnumeration=3 } RECEIVE_AND_TRANSMIT
{protobufEnumeration=4 } CONTRA_RECEIVE
{protobufEnumeration=5 } CONTRA_TRANSMIT
{protobufEnumeration=6 } CONTRA_RECEIVE_AND_CONTRA_TRANSMIT
«Enumeration»
RelativePosition
{protobufEnumeration=0 } NEAR_END
{protobufEnumeration=1 } FAR_END
{protobufEnumeration=2 } NOT_APPLICABLE
{protobufEnumeration=3 } ALL_LOCATIONS
«Enumeration»
ObjectType
SERVICE_INTERFACE_POINT
TAPI_CONTEXT
PROFILE
GnmiStreamingObjectTypeAugmentsObjectType
«abstraction»
«Enumeration»
GnmiStreamingObjectType
MEASUREMENT_DETAILS
«Enumeration»
NormalizedMeasurementType
{protobufEnumeration=0 } NO_NORMALIZED_TYPE
{protobufEnumeration=1 } BBE
{protobufEnumeration=2 } CHROM_DISP
{protobufEnumeration=3 } DELAY_FRAME_COUNT
{protobufEnumeration=4 } DIFF_GROUP_DELAY
{protobufEnumeration=5 } FEC_CORRECTABLE_BLOCKS
{protobufEnumeration=6 } FEC_CORRECTED_BITS
{protobufEnumeration=7 } FEC_CORRECTED_BYTES
{protobufEnumeration=8 } FEC_CORRECTED_CODEWORDS
{protobufEnumeration=9 } FEC_CORRECTED_ERROR
{protobufEnumeration=10 } FEC_POST_FEC_BER
{protobufEnumeration=11 } FEC_PRE_FEC_BER
{protobufEnumeration=12 } FEC_SYMBOL_ERRORS
{protobufEnumeration=13 } FEC_UNCORRECTABLE_BITS
{protobufEnumeration=14 } FEC_UNCORRECTABLE_BLOCKS
{protobufEnumeration=15 } FEC_UNCORRECTABLE_BYTES
{protobufEnumeration=16 } FEC_UNCORRECTABLE_WORDS
{protobufEnumeration=17 } FEC_UNCORRECTED_CODEWORDS
{protobufEnumeration=18 } FREQ_OFFS
{protobufEnumeration=19 } OPTICAL_GAIN
{protobufEnumeration=20 } OPT_PWR_SPECTR_DENS_INPUT
{protobufEnumeration=21 } OPT_PWR_SPECTR_DENS_OUTPUT
{protobufEnumeration=22 } OPT_TOTAL_PWR_INPUT
{protobufEnumeration=23 } OPT_TOTAL_PWR_OUTPUT
{protobufEnumeration=24 } OPTICAL_TILT
{protobufEnumeration=25 } POL_MODE_DISP
{protobufEnumeration=26 } SES
{protobufEnumeration=27 } UAS
{protobufEnumeration=28 } VOA_INPUT
{protobufEnumeration=29 } VOA_OUTPUT
«Enumeration»
Pm
BBE
CHROM_DISP
DELAY_FRAME_COUNT
DIFF_GROUP_DELAY
FEC_CORRECTABLE_BLOCKS
FEC_CORRECTED_BITS
FEC_CORRECTED_BYTES
FEC_CORRECTED_CODEWORDS
FEC_CORRECTED_ERRORS
FEC_POST_FEC_BER
FEC_PRE_FEC_BER
FEC_SYMBOL_ERRORS
FEC_UNCORRECTABLE_BITS
FEC_UNCORRECTABLE_BLOCKS
FEC_UNCORRECTABLE_BYTES
FEC_UNCORRECTABLE_WORDS
FEC_UNCORRECTED_CODEWORDS
FREQ_OFFS
OPT_GAIN
OPT_PWR_SPECTR_DENS_INPUT
OPT_PWR_SPECTR_DENS_OUTPUT
OPT_TOTAL_PWR_INPUT
OPT_TOTAL_PWR_OUTPUT
OPT_TILT
POL_MODE_DISP
SES
UAS
VOA_INPUT
VOA_OUTPUT

Figure 42 Data types

The figure below shows QualifiedMeasurementCommon, CurrentData and HistoryData for comparison.

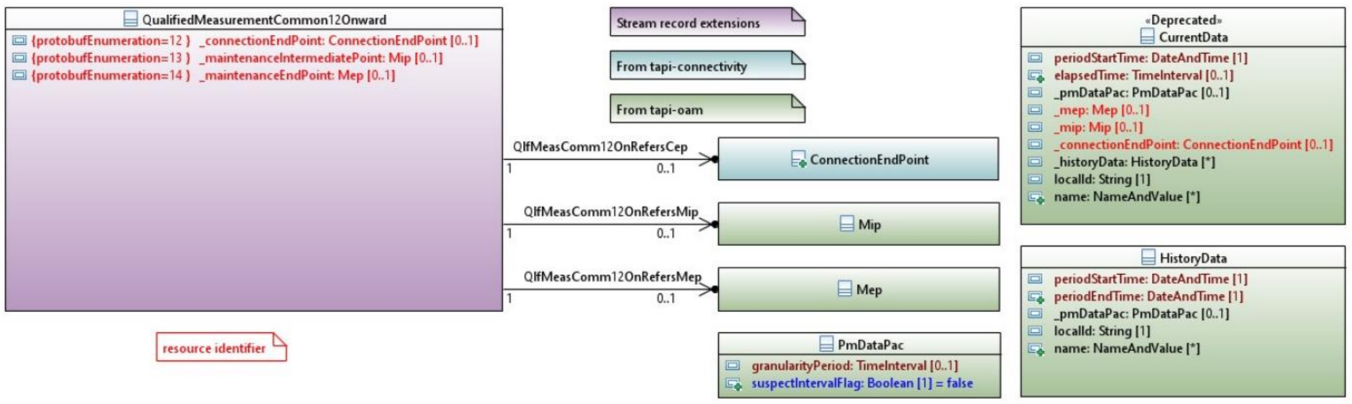

QualifiedMeasurementCommon12Onward
{protobufEnumeration=12 } _connectionEndPoint: ConnectionEndPoint [0..1]
{protobufEnumeration=13 } _maintenanceIntermediatePoint: Mip [0..1]
{protobufEnumeration=14 } _maintenanceEndPoint: Mep [0..1]
Stream record extensions
From tapi-connectivity
From tapi-oam
QlfMeasComm12OnRefersCep
1
0..1
ConnectionEndPoint
QlfMeasComm12OnRefersMip
1
0..1
Mip
QlfMeasComm12OnRefersMep
1
0..1
Mep
resource identifier
PmDataPac
granularityPeriod: TimeInterval [0..1]
suspectIntervalFlag: Boolean [1] = false
«Deprecated»
CurrentData
periodStartTime: DateAndTime [1]
elapsedTime: TimeInterval [0..1]
_pmDataPac: PmDataPac [0..1]
_mep: Mep [0..1]
_mip: Mip [0..1]
_connectionEndPoint: ConnectionEndPoint [0..1]
_historyData: HistoryData [*]
localId: String [1]
name: NameAndValue [*]
HistoryData
periodStartTime: DateAndTime [1]
periodEndTime: DateAndTime [1]
_pmDataPac: PmDataPac [0..1]
localId: String [1]
name: NameAndValue [*]

Figure 43 Related Classes

Where a solution supports streaming of performance data as described in this TR, support for retrieval, via GET, of CurrentData and HistoryData is not required, and the system may not provide support for CurrentData and HistoryData.

8.3 Appendix – Stream record example

The following is an example of an alarm:

```
{
            "log-record-header": {
                        "token": "Rfm8:1666904039108_0:-1379579382:AAAAAQAAgnk=",
                        "log-append-time-stamp": "2022-10-27T20:46:35.359Z",
                        "entity-key": "-1542773359860369954",
                        "record-type": "RECORD_TYPE_CREATE_UPDATE",
                        "full-log-record-offset-id": [{
                                                "value-name": "partition",
                                                "value": "0"
                                    },
                                    {
                                                "value-name": "offset",
                                                "value": "377"
                                    }
                        ]
            },
            "log-record-body": {
                        "event-time-stamp": {
                                    "approx-date-and-time": {
                                                "primary-time-stamp": "2022-04-01T07:04:36.101Z",
                                                "spread": "SPREAD_AT",
                                                "source-precision": "SOURCE_PRECISION_SYNCHRONIZED"
                                    }
                        },
                        "record-content": "CONDITION_DETECTOR",
                        "condition-detector": {
                                    "condition-native-name": "Loss Of Signal",
                                    "measured-entity-uuid": "113f57a4-74a5-36a4-7a4b-5f4ffa38159a",
                                    "measured-entity-native-id": "fac-63782-1uag",
                                    "measured-entity-device-native-name": "LON_76728A",
                                    "condition-normalized-name": "LOS",
                                    "measured-entity-class": "CONNECTIVITY_OBJECT_TYPE_CONNECTION_END_POINT",
                                    "detector-uuid": "10d490f2-3e3d-38d9-9346-4c923fa38195",
                                    "detector-native-id": "-1542773359860369954",
                                    "condition-detector-type": " CONDITION_DETECTOR_TYPE_ALARM_DETECTOR",
                                    "tapi-fm:detected-condition": {
                                        "detected-condition-name": "ALARM_NAME_LOS",
                                        "detected-condition-native-name": "Loss Of Signal",
                                        "detected-condition-native-info": "56",
                                        "detector-info": {
                                                "perceived-severity": "CRITICAL",
                                                "service-affecting": "SERVICE_AFFECTING",
                                                "is-acknowledged": false,
                                                "detector-category": "DETECTOR_CATEGORY_CONNECTIVITY"
                                        },
                                        "simple-detector": {
                                                "simple-detector-state": "SIMPLE_DETECTOR_STATE_ACTIVE"
                                        }
                                },
                                “tapi-vendor-detector-info:detector-info”: {
                                        “node-type”: “Plant",
                                        “self-clearing”: true,
                                        “number-of-occurrences”: “10"

                                }
                        }
                }
}
```

A more compact conformant alarm record:

```
{
        "log-record-header": {
                "token": "Rfm8:1666904039108_0:-1379579382:AAAAAQAAgnk=",
                "log-append-time-stamp": "2022-10-27T20:46:35.359Z",
                "entity-key": "-1542773359860369954",
                "record-type": "RECORD_TYPE_CREATE_UPDATE",
                "full-log-record-offset-id": [{
                                        "value-name": "partition",
                                        "value": "0"
                                },
                                {
                                        "value-name": "offset",
                                        "value": "377"
                                }
                ]
        },
        "log-record-body": {
                "event-time-stamp": {
                        "approx-date-and-time": {
                                "primary-time-stamp": "2022-04-01T07:04:36.101Z",
                                "source-precision": "SOURCE_PRECISION_SYNCHRONIZED"
                        }
                },
                "record-content": "CONDITION_DETECTOR",
                "condition-detector": {
                        "measured-entity-native-id": "fac-63782-1uag",
                        "measured-entity-device-native-name": "LON_76728A",
                        "detector-native-id": "-1542773359860369954",
                        "condition-detector-type": " CONDITION_DETECTOR_TYPE_ALARM_DETECTOR",
                        "tapi-fm:detected-condition": {
                                "detected-condition-name": "ALARM_NAME_LOS",
                                "detected-condition-native-name": "Loss Of Signal",
                                "simple-detector": {
                                        "simple-detector-state": "SIMPLE_DETECTOR_STATE_ACTIVE"
                                }
                        }
                }
        }
}
```

8.4 Appendix – Detectors, detected conditions and alarms

This section deals with the conceptual model underlying alarm reporting.

8.4.1 Detect

As a result of observation of some properties of a thing (entity, assembly, environment etc.) there is a recognition of some relevant states distinct from the previously recognized states.

8.4.2 Detector

A device that both observes (monitors) a thing in a particular way and recognizes the state and change of state.

8.4.3 Condition

A specific combination of states that is defined and is of interest.

8.4.4 Condition detector

A device that detects a specific condition and reports the presence that condition (the detected condition). It also reports when that condition is no longer present.

The condition detector represents any monitoring component that assesses properties of something and determines from those properties what conditions are associated with the thing.

For example, a thing might be "too hot" or might be "unreliable".

The monitor may a multi-state output.

The condition detector lifecycle depends upon the lifecycle of the thing it is monitoring. For example, once the thing is removed/deleted, so is the condition detector.

8.4.5 Detected condition

The detected condition may be a state with associated values and with importance/priority etc.

8.4.6 Alarm

An alarm is an alert that relates to a specific detected condition that is considered problematic, where it is likely that some action will need to be taken to adjust system state such that the condition is cleared. Whilst the condition (alarm condition) is present, the alarm is ACTIVE and when not present the alarm is CLEAR.

8.4.7 Alarm detector

A detector that detects and highlights a specific alarm condition by reporting that the alarm condition is ACTIVE and the absence of that condition by reporting that the alarm is CLEAR.

8.4.8 Traditional alarm reporting and legacy-properties

The legacy-properties are provided to deal with the traditional alarm reporting properties. Alarm systems of the 20th century were based primarily on local lamps (initially filament bulbs) and bells. Lamps can only be on or off, and bells sounding or not sounding, so alarms were Boolean in nature. Where a detector was essentially multi-state it was converted into multiple Boolean statements.

The management of the network devices (equipments) was essentially human only and local only (there were rarely remote systems). The network device with the problem was the only possible indicator of importance and it had only three distinct bulbs to illuminate (filament bulbs tend to fail requiring costly replacement).

The network devices were relatively simple in function and analysis of the detectors was crude. There was only the network device to indicate severity. The network device also could provide the best view as to whether a service was impacted, although clearly it had almost no knowledge.

In a modern solution with well-connected remote systems that increasingly analyse problems and where there is increasingly 'lights out' building operation, the network device's guess at severity etc. is irrelevant. In addition, with sophisticated resilience mechanisms, the network device cannot make any relevant statement on whether the customer service has been impacted.

Likewise, in a world where there were no remote systems and local management was the only practice, alarms had to be locally 'acknowledged'. Where there are remote systems, per alarm acknowledge is burdensome.

8.5 Appendix – Measurement data and performance monitoring solution architecture

The figure below shows a hierarchical solution. In this solution the High-Level Controller takes a detailed view of tasks such as network rebalancing and multi-layer analysis of problem. It delegates coordination of provisioning, some restoration and information density improvements to the Lower Controller. The High-Level Controller takes on all responsibilities for analysis of measurement data.

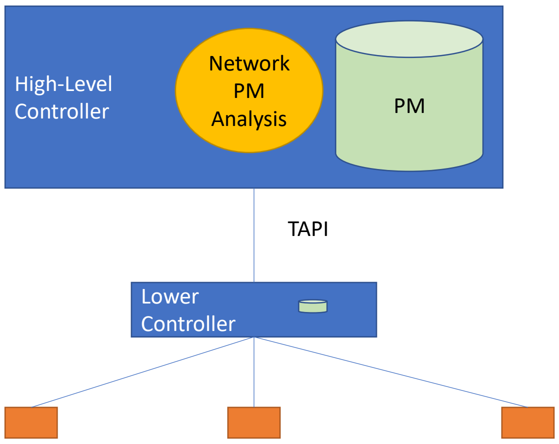
The High-Level Controller is assumed to not be a monolith but is instead a system of many components (where TAPI may be used between components).

The Lower Controller collects, cleans, normalizes and consolidates measurement data from a set of network devices and control plane elements. It the exports it to the High-Level Controller as rapidly and efficiently as possible.

In an ideal formation of this architecture, the components and functions of the High-Level Controller and the Lower Controller are actually parts of the same single control fabric where both collections of components from the overall Control System where components interact using standard model definitions such as TAPI.

Note that the term:

- Orchestrator is not use as it tends to suggest a minimalistic light touch whereas there is a need for a broad and deep understanding of the network detail across the entire network.
- Domain Controller is not use as Domain is ambiguous and use of the term can lead to inappropriate demarcation of the problem space.



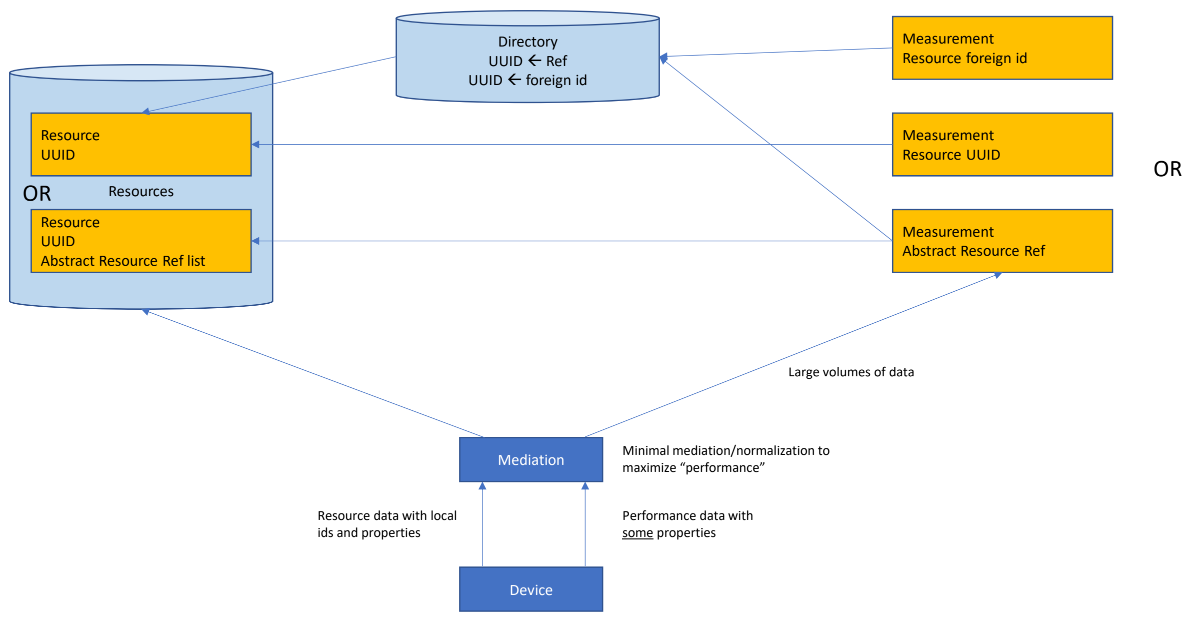
8.6 Appendix – Measurement data – Referencing resources

The diagram below illustrates three approaches to relate measurement data records to resources.

The measurement data records, shown on the right, illustrate the three approaches:

- Resource foreign id: An id that is not present in any of the resources but that does relate to one or more resource and that can be looked up in a directory
- Resource UUID reference: The usual approach used for entity references where the UUID is the identifier of a TAPI resource
- Abstract resource reference: A reference that is not the formal identifier of the resource but that is available in the entity representing the resource. There may be several resources with the same value for abstract resource reference and there may be several abstract resource references present in a resource. The measurement data will include only one abstract resource reference.
    - Note that the abstract resource reference could also be used in a similar way to the foreign id with in that it can be looked up in a directory.

The abstract resource reference approach (abstract-resource-ref) is one of the options supported by the measurement data stream structure.



8.7 Appendix – Protobuf encoding sketch

The fragment of data below illustrates the compression of data that is achieved using protobuf as opposed to raw ASCII. Note that some of the fields are encoded as strings and hence there is still a significant volume of ASCII in the sample.

8.7.1 Message fragment in JSON

The following sample instance, shown as a JSON fragment, was assessed.

```
{
  "measurementDetails": {
    "timeMeasurementWasSampled": "1693240037000",
    "qualifiedMeasuredValue": {
      "value": -5.2,
      "valueQualifier": "SUSPECT",
      "units": "dBm",
      "qualifiedValueName": "MIN"
    },
    "nativeResourceId": "OPT-8-7-13",
    "directionOfMeasuredSignal": "RECEIVE",
    "sampleInterval": "900000000000",
    "abstractResourceRef": "NE1-OPT-8-7-13",
    "nativeMeasurementType": "RX_OPTICAL_POWER_MIN",
    "normalizedMeasurementType": "OPT_TOTAL_PWR_INPUT"
  }
}
```

The JSON fragment is >>400 bytes (ignoring white space formatting).

8.7.2 Protobuf hex

The hex protobuf for the fragment above is

```
0A 5B 08 88 9D E2 E7 A3 31 12 15 09 CD CC CC CC CC CC 14 C0 10 01 1A 03 64 42 6D 22 03 4D 49 4E 1A 0A 4F
50 54 2D 38 2D 37 2D 31 33 30 01 38 80 D0 B8 E1 98 1A 42 0E 4E 45 31 2D 4F 50 54 2D 38 2D 37 2D 31 33 4A
14 52 58 5F 4F 50 54 49 43 41 4C 5F 50 4F 57 45 52 5F 4D 49 4E 50 16
```

This is 93 bytes.

8.7.3 Decoded protobuf

The protobuf decodes as follows

```
0A = Identifier of measurementDetails (1) → 00001 010 (id 1, wire type 2)
5B = length (91 bytes)

08 = Identifier of timeMeasurementWasSampled (1) → 00001 000 (id 1, wire type 0)
88 9D E2 E7 A3 31 = decodes as 1693240037000 (MSB set for all but 31 encoding 1693240037000)

12 = Identifier of qualifiedMeasuredValue (2) → 00010 010 (id 2, wire type 2)
15 = Length (21 bytes)

09 = Identifier of value (01) → 00001 001 (id 1, wire type 1 (fixed 8 byte quantity))
CD CC CC CC CC CC 14 C0

10 = Identifier of valueQualifier (02) → 00010 000 (id 2, wire type 0)
01 = SUSPECT

1A = Identifier of units (3) → 00011 010 (id 3, wire type 2)
```

```
03 = Length
64 42 6D = dBm

22 = Identifier of qualifiedValueName (04) → 00100 010 (id 4, wire type 2)
03 = Length
4D 49 4E = MIN (last of 21 byte structure)

1A = Identifier of nativeResourceId (03) → 00011 010 (id 3 wire type 2)
0A = Length
4F 50 54 2D 38 2D 37 2D 31 33 = OPT-8-7-13

30 = Identifier of directionOfMeasuredSignal (06) → 00110 000 (id 6, wire type 0)
01 = RECEIVE

38 = Identifier of sampleInterval (07) 38 → 00111 000 (id 7, wire type 0)
80 D0 B8 E1 98 1A = decodes as 900000000000 (MSB is used to identify continuation, 7 bit quantities)

42 = Identifier of abstractResourceRef (08) 42 → 01000 010 (id 8, wire type 2)
0E = Length
4E 45 31 2D 4F 50 54 2D 38 2D 37 2D 31 33 = NE1-OPT-8-7-13

4A = Identifier of nativeMeasurementType (09) 4A → 01001 010 (id 9, wire type 2)
14 = Length
52 58 5F 4F 50 54 49 43 41 4C 5F 50 4F 57 45 52 5F 4D 49 4E = RX_OPTICAL_POWER_MIN

50 = Identifier of normalizedMeasurementType (10) 5A → 01011 000 (id 10, wire type 0)
16 = OPT_TOTAL_PWR_INPUT
```

8.7.4 Byte reduction

The protobuf form requires less than 0.25 of the data to convey the message.

If the strings were to be removed or replaced by enumerations, the dominant volume would then come from the two time properties.

It may be relevant in future to consider relative time for the measurement sample as the varint coding would then reduce the field size.

8.7.5 Protobuf encoding

Protobuf hex decoding tools are available, e.g., [PROTO-DEC].

Protobuf encoding is explained at [PROTO-ENC].

8.8 Appendix – Measurement YANG

The tapi-streaming-gnmi YANG is available at https://github.com/Open-Network-Models-and-Interfaces-ONMI/TAPI/tree/v2.6.0/YANG.

9 References

[DDD]	https://en.wikipedia.org/wiki/Domain-driven_design
[GNMI]	https://github.com/openconfig/reference/tree/master/rpc/gnmi
[GNMI-SPEC]	https://github.com/openconfig/reference/blob/master/rpc/gnmi/gnmi-specification.md (version 0.10.0 from master branch current on 20230712)
[GNMI-HIST]	https://github.com/openconfig/reference/blob/master/rpc/gnmi/gnmi-history.md (version 0.1.0 from master branch)
[GNMI-PROTO]	https://github.com/openconfig/gnmi/blob/master/proto/gnmi/gnmi.proto
[IETF-MOBO]	https://datatracker.ietf.org/doc/html/draft-davis-netmod-modelling-boundaries-01
[JSON]	https://www.json.org/json-en.html
[KAFKA]	https://kafka.apache.org/
[KAFKA-COMP]	https://kafka.apache.org/documentation/#compaction
[NYQUIST]	https://en.wikipedia.org/wiki/Nyquist-Shannon_sampling_theorem
[ONF TR-512]	https://www.opennetworking.org/wp-content/uploads/2018/12/TR-512_v1.4_OnfCoreIm-info.zip (also available as ITU-T G.7711 at https://www.itu.int/rec/T-REC-G.7711/en)
[LF TR-547]	TAPI v2.6.0 Reference Implementation Agreement (TR-547 v3.3). Available at https://github.com/Open-Network-Models-and-Interfaces-ONMI/TAPI-Documentation/tree/v2.6.0/ReferenceImplementationAgreements
[PROTO-ENC]	https://protobuf.dev/programming-guides/encoding/#simple
[PROTO-DEC]	https://protobuf-decoder.netlify.app/
[RESTCONF]	https://tools.ietf.org/html/rfc8040
[RIA AT]	Located on https://wiki.opennetworking.org/display/OTCC/TAPI+RIA+Associated+Documents find the most recent TAPI_Alarm_TCA_List.xlsx
[RIA SS]	Located on https://wiki.opennetworking.org/display/OTCC/TAPI+RIA+Associated+Documents find the most recent TAPI_Notification_and_Streaming_Sequences.xlsx
[RFC6455]	https://tools.ietf.org/html/rfc6455 The websocket protocol
[RFC6455-AUTH]	https://tools.ietf.org/html/rfc6455#page-53
[RFC6455-PONG]	https://tools.ietf.org/html/rfc6455#section-5.5.3
[RFC6750]	https://tools.ietf.org/html/rfc6750
[RFC6902]	https://datatracker.ietf.org/doc/html/rfc6902#page-6
[TAPI PROTO]	https://github.com/Open-Network-Models-and-Interfaces-ONMI/TAPI/tree/v2.6.0/PROTOBUF
[WC3 SSE]	https://www.w3.org/TR/eventsource/ Server-Sent Events
[WC3 XML]	https://www.w3.org/standards/xml/
[YANG]	https://tools.ietf.org/html/rfc6020

10 Definitions and Terminology

No terms are special to this work.

This document uses terms defined elsewhere. In some cases, the term is used in a particular way in this document; Particular usage is highlighted.

CEP	Connection End Point (a TAPI class) [ONF TR-547]
Client	In this document, an application/system (usually a controller (see below)) which uses TAPI streaming (hence is a TAPI client) to gain and maintain alignment with the current state of a network that is exposed through a provider system • The client is a superior controller to the provider
Control	To attempt to achieve an agreed intent and measure the results to validate achievement and where necessary to initiate actions to fix any problems • Note that automated management is control (see [ONF TR-512] (specifically TR-512.8))
Controlled network	A controlled system that consists of devices that provide networking functionality.
Controlled system	A system of devices that are managed/controlled by a controller. • Also covers managed system as automation of management is the focus and automated management is control
Controller	A devices that controls other devices. • Examples, with varying degrees of control responsibility and varying degrees of automation, include devices mentioned in this definition list such as SDTN Controller, SDN-C, Orchestrator, EMS, NMS, OSS (see Figure 1 Example SDN architecture for WDM/OTN network on page 10) • The primary focus for Streaming in TAPI 2.4 is simple and efficient ongoing alignment of a Controller (client) with a view presented by another Controller (provider).
DDD	Domain Driven Design
Device	A thing formed from an assembly of electronic (and mechanical) equipment usually running software that is made or adapted for a particular purpose. See also: • network device • controller
EMS	Element Management System
ForwardingDomain	An ONF Core Model class [ONF TR-512]
GUI	Graphical (human) User Interface
KAFKA	https://kafka.apache.org/
Log	A sequential store
Management	(see control)
Management-Control	(see control)
MEP	Maintenance End Point (a TAPI class) [LF TR-547]
Network device	A device that is designed to and/or is used to support networking functions (e.g., an OTN Switch).
NMS	Network Management System

Node	A TAPI Class [LF TR-547]
OAM	Operations Administration and Maintenance
ONF	Open Networking Foundation https://www.opennetworking.org/
Orchestrator	An overarching Controller that coordinates other subordinate controllers
OSS	Operations Support System
PoC	Proof of Concept
Provider	In this document, an application/system (usually a controller (see above)) which offers TAPI streaming (hence is a TAPI provider) to enable a client to gain and maintain alignment with the current state of the network that it, the provider, exposed • The client is a superior controller to the provider
SDN-C	Software Defined Network Controller [LF TR-547]
SDTN	Software-Defined Transport Network
SDTN Controller	An SDTN Orchestrator [LF TR-547]
TAPI	Transport API (Application Programmers Interface) [LF TR-547]
Tombstone	A special message that indicates that an entity no longer exists.
Topology	A TAPI Class [LF TR-547]
TR	Technical Report (from LF)
UML	Universal Modeling Language
UUID	Universally Unique Identifier
WS	WebSockets
YANG	Yet Another Next Generation… see [YANG]

11 Individuals engaged

11.1 Editors

Nigel Davis	Ciena

11.2 Contributors

Nigel Davis	Ciena
Brian Johnson	Ciena
Ramon Casellas	CTTC
Kam Lam	FiberHome Telecom USA
Roshan Joyce	Fujitsu
Karthik Sethuraman	NEC
Pedro Amaral	Nokia
Gabriele Galimberti	Nokia
Kishore Kumar	Nokia
Andrea Mazzini	Nokia
Jonathan Sadler	Nokia
Arturo Mayoral	Telecom Infra Project
Ronald Zabaleta	Telefónica
Malcolm Betts	ZTE
Xiaobing Niu	ZTE
Jai Qian	ZTE

12 Appendix: Changes between versions

12.1 Changes between v1.1 and v2.0

- Updated UML/YANG – 2.4.0 (including UML diagrams) covering v2.4.0 changes in tapi-streaming:
    - RPCs have been removed
    - Most properties are now Conditional with the CONDITION stated in the description.
    - record-trigger (ON_CHANGE, PERIODIC) aspect separated out from log-record-strategy (now CHANGE_ONLY and WHOLE_ENTITY)
    - object-class-identifier, used in v2.1.3 has been replaced by object-type from tapi-common and this is now augmented with classes from tapi-streaming (and all relevant classes in each other model)
    - Formalized enumerations (represented as identities) for connection-protocol and encoding-format replace strings used in v2.1.3 and gNMI and PROTOBUF added for streaming options
    - Two new compacted log properties (max-allowed-segment-roll-delay and max-compaction-lag added
    - information-record-strategy added in preparation for advanced PM streaming
    - detected-condition from tapi-fm.yang used in place of alarm-condition-detector-detail (which is deprecated)
    - condition-detector-type used to explain choice of details in detected-condition augmentation
- YANG explanations have been improved to reference condition statements in the YANG descriptions
- Various wording clarifications and improvements
- More data types included in the explanatory text
- Improvements to non-normative proposal for periodic measurements streaming
- parent-address explanation improved and specific usage for each object class detailed
- Use of tapi-fm introduced and explained
- References to TR-547 detail of relevant parameters improved
- Some use case steps improved
- Alarm examples has been updated (Appendix)
- References to the TAPI RIA Associated Documents (TAPI_Alarm_TCA_List.xlsx and TAPI_Notification_and_Streaming_Sequences.xlsx) have been added

12.2 Changes between v2.0 and v3.1

- Addition of detailed description of the use of the CHANGE_ONLY (delta) streaming approach
    - Adjustments to existing section to account for the additions.
- Addition of PM streaming via gNMI/Protobuf
    - Addition of new description, model and use cases.
- Improvements to some documentation.

12.3 Changes between v3.1 and v3.2

- ONF replaced with LF in most cases (other than where ONF is historically relevant, or the reference is only available as ONF material). LF copyright etc. has been added to the cover and footer.
- Clarification of usage of use of [GNMI-HIST] for use in subscription for “lossless” recovery.
- Explanation of use of available streams for gNMI/Protobuf streams
- A protobuf file has been constructed and published separately from this document, hence the protobuf content has been removed from this document.

- Clarified that this release only supports a single gNMI stream per port.
- There was a full YANG definition in an appendix in previous releases of this document, this has been removed and instead the formal YANG has been referenced.

End of Document