Mplify Standard
Mplify 147

LSO Allegro, LSO Interlude and LSO Legato
Streaming Management
Developer Guide

February 2026

Disclaimer

The information in this publication is freely available for reproduction and use by any recipient and is believed to be accurate as of its publication date. Such information is subject to change without notice and Mplify Alliance (Mplify) is not responsible for any errors. Mplify does not assume responsibility to update or correct any information in this publication. No representation or warranty, expressed or implied, is made by Mplify concerning the completeness, accuracy, or applicability of any information contained herein and no liability of any kind shall be assumed by Mplify as a result of reliance upon such information.

The information contained herein is intended to be used without modification by the recipient or user of this document. Mplify is not responsible or liable for any modifications to this document made by any other party.

The receipt or any use of this document or its contents does not in any way create, by implication or otherwise:

a) any express or implied license or right to or under any patent, copyright, trademark or trade secret rights held or claimed by any Mplify member which are or may be associated with the ideas, techniques, concepts or expressions contained herein; nor

b) any warranty or representation that any Mplify members will announce any product(s) and/or service(s) related thereto, or if such announcements are made, that such announced product(s) and/or service(s) embody any or all of the ideas, technologies, or concepts contained herein; nor

c) any form of relationship between any Mplify member and the recipient or user of this document.

Implementation or use of specific Mplify standards, specifications, or recommendations will be voluntary, and no Member shall be obliged to implement them by virtue of participation in Mplify Alliance. Mplify is a global alliance of network, cloud, cybersecurity, and enterprise organizations working together to accelerate the AI-powered digital economy through standardization, automation, certification, and collaboration. Mplify does not, expressly or otherwise, endorse or promote any specific products or services.

Table of Contents

List of Contributing Members

The following members of the Mplify participated in the development of this document and have requested to be included in this list.

Member
Amartus

Table 1. Contributing Members

1. Abstract

This standard is intended to assist implementation of the Streaming Management functionality defined for the LSO Allegro, Legato, and Interlude Interface Reference Points (IRPs), for which requirements and use cases are defined in Mplify 133.1 *Allegro, Interlude and Legato Fault Management and Performance Monitoring BR&UC* [Mplify 133.1].

This standard normatively incorporates the following files by reference as if they were part of this document, from the GitHub repository:

MEF-LSO-Allegro-SDK

commit id: 753b61cbb1c78cd025526b12836b1e33a8d86a94

- `serviceApi/pm/streamingManagement.api.yaml`

MEF-LSO-Interlude-SDK

commit id: 621858bd092e45b4acc695d38c5bdd46b8eccb6d

- `serviceApi/pm/streamingManagement.api.yaml`

MEF-LSO-Legato-SDK

commit id: 0b5cf2f24c3179cc3deef700344375fc702a31b2

- `serviceApi/pm/streamingManagement.api.yaml`

Note: The repository contains `serviceApi/pm/streamingManagement.api.all-in-one.yaml` version of the OAS spec. This version is self-contained and does not use references to external resources.

The Streaming Management API is defined using OpenAPI 3.0 Specification [OAS-V3].

2. Terminology and Abbreviations

This section defines the terms used in this document. In many cases, the normative definitions of terms are found in other documents. In these cases, the third column is used to provide the reference that is controlling, in other Mplify or external documents.

Term	Description	Reference
Application Program Interface (API)	In the context of LSO, API describes one of the Management Interface Reference Points based on the requirements specified in an Interface Profile, along with a data model, the protocol that defines operations on the data, and the encoding format used to encode data according to the data model. In this document, API is used synonymously with REST API.	[MEF 55.1]
API Gateway	An API gateway is a software pattern or component that acts as an intermediary between clients and the backend services of the server.	This document
Buyer	In the context of this document denotes the organization or individual acting as the customer in a transaction over a Cantata (Customer <-> Service Provider) or Sonata (Service Provider <-> Partner) Interface. The Buyer consumes streaming management API exposed through Allegro and Interlude respectively.	This document; adapted from [MEF 80]
Client	In the context of this document, denotes a system that consumes the uses the stream management functionality.	This document
Consumer	A component that consumes messages from a data stream. In the scope of this document, synonymous to `Client`	This document
Event	A specific occurrence or a state change that is note-worthy to the system administrator.	[Mplify 133.1]
Message	Typically defined as a unit of information exchanged between components or services in a distributed system. In the context of this standard, a unit of information, that is a manifestation of an event exchanged between producer and consumer using an event-driven architectural pattern.	[Mplify 133.1]
Producer	A component that produces messages and exposes them via message stream to the consumers	This document
REST API	Representational State Transfer. REST provides a set of architectural constraints that, when applied as a whole, emphasizes the scalability of component interactions, the generality of interfaces, the independent deployment of components, and intermediary components to reduce interaction latency, enforce security, and encapsulate legacy systems.	[REST]
Seller	In the context of this document, denotes the organization or individual acting as the supplier in a transaction over a Cantata (Customer <-> Service Provider) or Sonata (Service Provider <-> Partner) Interface. The Seller exposes streaming management API through Allegro and Interlude respectively.	This document; adapted from [MEF 80]
Server	In the context of this document, denotes a system that exposes the stream management functionality and typically produces the stream messages.	This document

Table 2. Terminology

3. Compliance Levels

The key words **"MUST"**, **"MUST NOT"**, **"REQUIRED"**, **"SHALL"**, **"SHALL NOT"**, **"SHOULD"**, **"SHOULD NOT"**, **"RECOMMENDED"**, **"NOT RECOMMENDED"**, **"MAY"**, and **"OPTIONAL"** in this document are to be interpreted as described in BCP 14 (RFC 2119 [RFC2119], RFC 8174 [RFC8174]) when, and only when, they appear in all capitals, as shown here. All key words must be in bold text.

Items that are **REQUIRED** (contain the words **MUST** or **MUST NOT**) are labeled as **[Rx]** for required. Items that are **RECOMMENDED** (contain the words **SHOULD** or **SHOULD NOT**) are labeled as **[Dx]** for desirable. Items that are **OPTIONAL** (contain the words MAY or OPTIONAL) are labeled as **[Ox]** for optional.

A paragraph preceded by **[CRa]<** specifies a conditional mandatory requirement that **MUST** be followed if the condition(s) following the "<" have been met. For example, **"[CR1]<[D38]"** indicates that Conditional Mandatory Requirement 1 must be followed if Desirable Requirement 38 has been met. A paragraph preceded by **[CDb]<** specifies a Conditional Desirable Requirement that **SHOULD** be followed if the condition(s) following the "<" have been met. A paragraph preceded by **[COc]<**specifies a Conditional Optional Requirement that **MAY** be followed if the condition(s) following the "<" have been met.

4. Introduction

This standard specification document describes the Application Programming Interface (API) for Service Inventory Management functionality of the LSO Allegro, Interlude, and Legato Interface Reference Points (IRPs) as defined in the *MEF 55.1 Lifecycle Service Orchestration (LSO): Reference Architecture and Framework* [MEF 55.1]. The LSO Reference Architecture is shown in Figure 1 with the IRP highlighted.

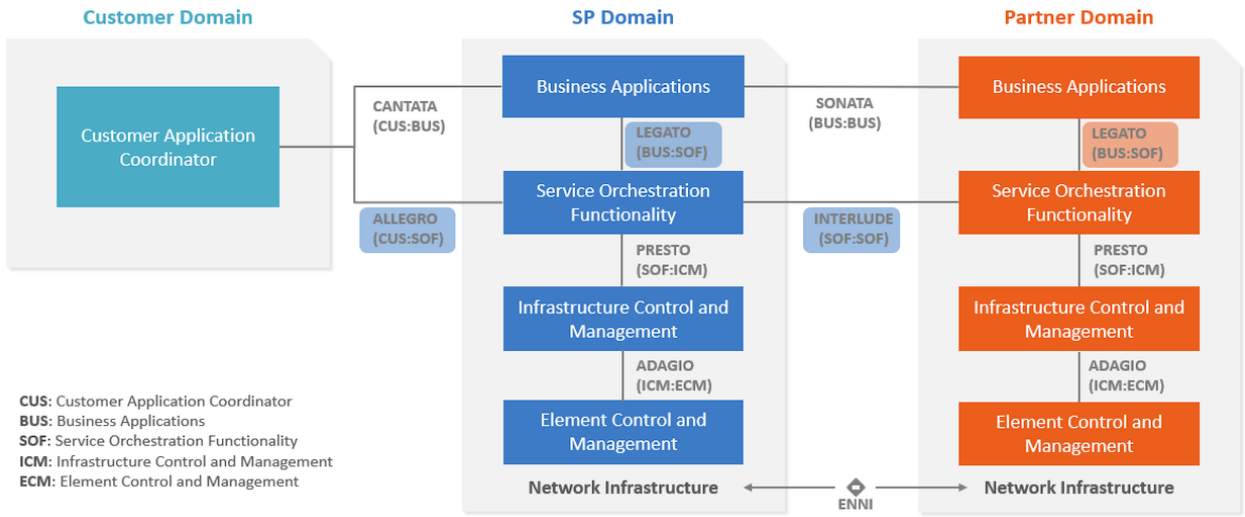

Figure 1. The LSO Reference Architecture

This document is structured as follows:

- Chapter 4 provides an introduction to Streaming Management in a broader context of Allegro, Interlude, and Legato APIs and their corresponding SDKs.
- Chapter 5 briefly discusses API and the supported use cases along with mapping into Mplify 133.1.
- Chapter 6 describes use cases in detail.
- Chapter 7 provides an in-detail discussion of the data model defined for the Streaming Management API

4.1 Description

This standard is scoped to cover APIs for Streaming Management. Although the management API might be used in different contexts we restrict the description in this standard to use cases that allow for exchanging performance data as defined in Mplify 133.1.

This document supports interactions over the Legato interface within a single operator as well as interaction with the Customer Domain and Partner Domain through Allegro and Interlude interfaces respectively.

Streaming Management API is used to:

- discover available `Topics` exposed by the `Server`
- manage subscriptions to these topics
- list existing subscriptions

4.2. Conventions in the Document

- Code samples are formatted using code blocks. When notation `<< some text >>` is used in the payload sample it indicates that a comment is provided instead of an example value and it might not comply with the OpenAPI definition.

- Model definitions are formatted as in-line code (e.g. `Service`).
- In UML diagrams the default cardinality of associations is `0..1`. Other cardinality markers are compliant with the UML standard.
- In the API details tables and UML diagrams required attributes are marked with a `*` next to their names.
- In UML sequence diagrams `{{variable}}` notation is used to indicate a variable to be substituted with a correct value.

4.3. Relation to Other Documents

This API implements the Performance Monitoring Streaming requirements and use cases that are defined in [Mplify 133.1].

4.4. Approach

As presented in Figure 2. the Allegro, Interlude, and Legato API frameworks consist of three structural components:

- Generic API framework
- Service-independent information (Function-specific information and Function-specific operations)
- Service-specific information (Mplify service specification data model)

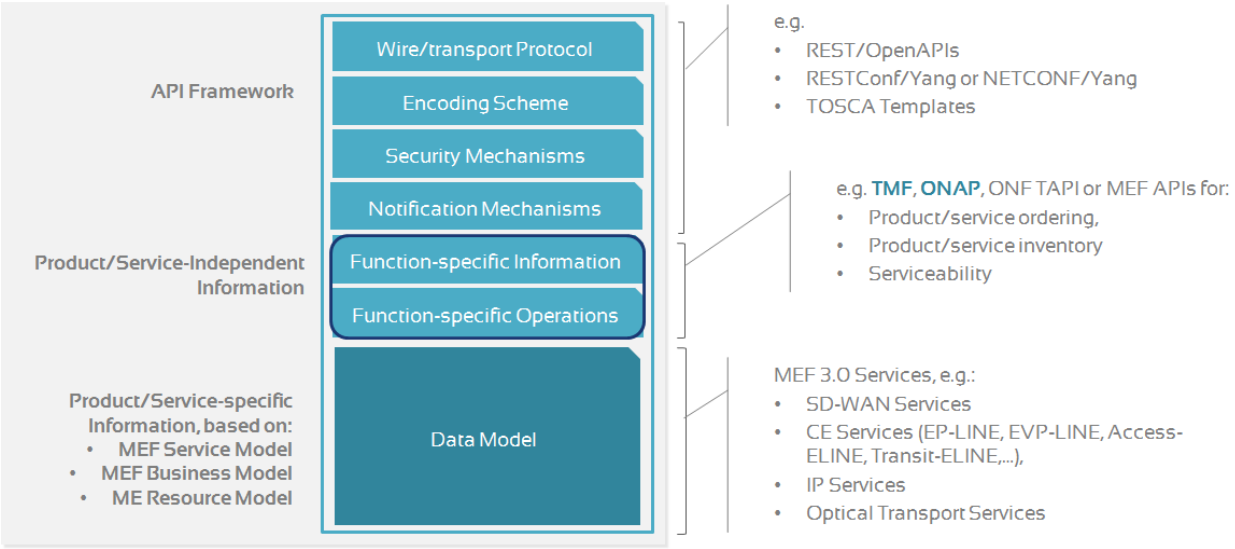

Figure 2. Allegro, Interlude, and Legato API Structure

The essential concept behind the framework is to decouple the common structure, information, and operations from the specific service information content.

Firstly, the Generic API Framework defines a set of design rules and patterns that are applied across all Allegro, Interlude, and Legato APIs.

Secondly, the service-independent information of the framework focuses on a model of a particular Allegro, Interlude, or Legato functionality and is agnostic to any of the service specifications. For example, this standard describes the Streaming Management model and operations that allow subscribing to streams of any service.

Finally, the service-specific information part of the framework focuses on service-related attributes and requirements that are being exchanged between streaming clients and producers.

4.5. High-Level Flow

The Streaming API in essence allows the `Client` to subscribe to a data stream exposed by the `Server`. Figure 3 presents a high-level flow in which a `Client`:

1. select the topic of interest - Subscribe section;
2. consumes messages - Consume section;
3. finalizes subscription - Unsubscribe section.

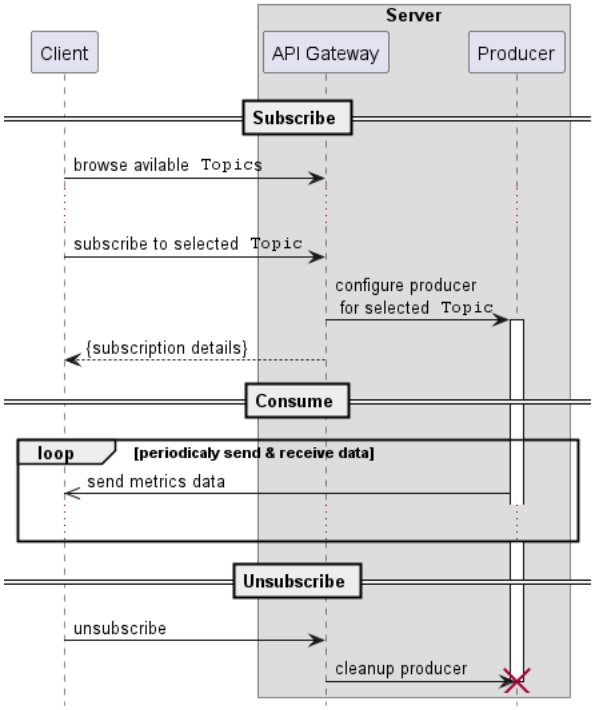

Figure 3. End-to-end high-level flow

The *Consume* section of this flow is not supported by management API as it is specific to the selected data exchange protocol. However, Mplify 133.1[Mplify 133.1] defines requirements for this part of the end-to-end flow.

Please note that the consumption of the data might be realized by various protocols leveraging broker (e.g., Kafka, AMQP) or broker-less (e.g., Web Socket, SSE) communication patterns.

5. API Description

This section presents the API structure and design patterns. It starts with the high-level use cases diagram. Then it describes the REST endpoints with use case mapping. Next, it gives an overview of the API resource model.

5.1. High-level use cases

Figure 4 presents a high-level use case diagram that is relevant for streaming management API. The mapping to appropriate Mplify 133.1 use cases is provided in the bottom part of the ellipse shape representing the use case.

A full list of the use cases for streaming can be found in [Mplify 133.1] in section 12. Use cases from Figure 4 are described extensively in Chapter 6.

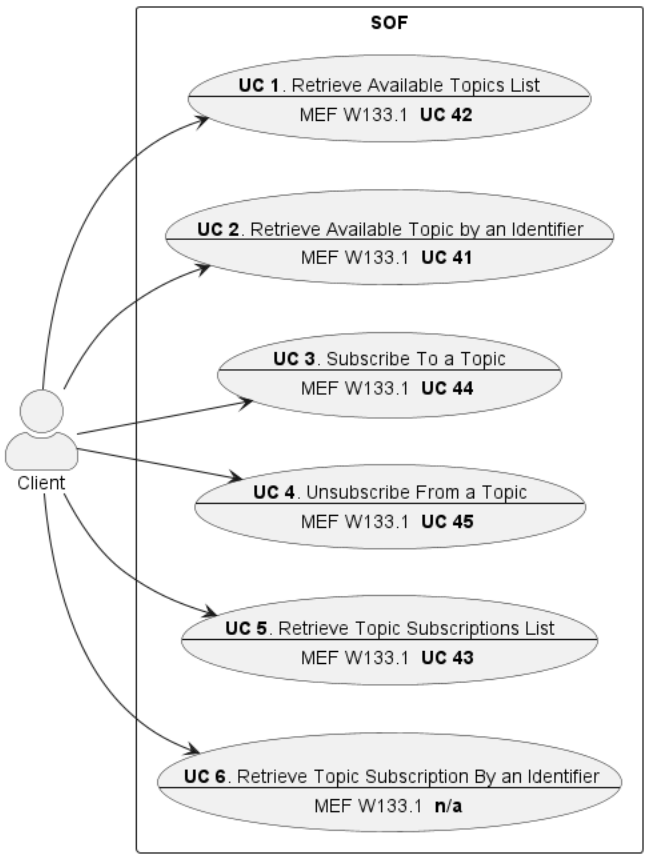

Figure 4. Use cases

The Service Orchestration Functionality (`SOF`) that exposes the Streaming management API, and manages streaming infrastructure and event production is referred to as `Server` in other parts of this document.

The `Client` interacts with the API to obtain information that allows for event consumption. The `Client` might be a system from *Customer Domain*, *Partner Domain*'s `SOF`, or a `Business Application` from *SP Domain* depending on the considered IRP as presented in Figure 1.

5.2. API Endpoint and Operation Description

Base URL for IRP:

```
https://{{serverBase}}{{?/seller_prefix}}/mefApi/{{irp}}/streamingManagement/v1/
```

the supported IPRs for this API are `legato`, `allegro`, and `interlude`.

The following API endpoints are implemented by the Server and allow the client to retrieve information about available topics and manage subscriptions.

`serviceApi/pm/streamingManagement.api.yaml`.

Streaming Management API to use case mapping:

API endpoint	Description	Use Case mapping
`GET /topic`	List Topics available for Subscription	UC 1: Retrieve Available Topics List
`GET /topic/{{id}}`	Retrieve Topic information by identifier	UC 2: Retrieve Available Topic by an Identifier
`POST /subscription`	Subscribe to a Topic	UC 3: Subscribe To a Topic
`DELETE /subscription/{{id}}`	Remove a Subscription for a Topic	UC 4: Unsubscribe From a Topic
`GET /subscription`	List all the Subscriptions to the Topics	UC 5: Retrieve Topic Subscriptions List
`GET /subscription/{{id}}`	Retrieve information about Subscription to a Topic	UC 6: Retrieve Topic Subscription By an Identifier

Table 3. Server-side mandatory Streaming Management API endpoints

[R1] The `Server` **MUST** support Streaming Management API endpoints listed in Table 3.

5.3. Integration of the Service-Specific Models

This section provides details on the extension mechanism available for Streaming Management API as well as the data payload that is exchanged between the `Server` and `Client`.

5.3.1. Streaming Management API Extension

Streaming management API allows for subscription to available topics as explained above.

The subscription models for request and response are depicted in Figure 5.

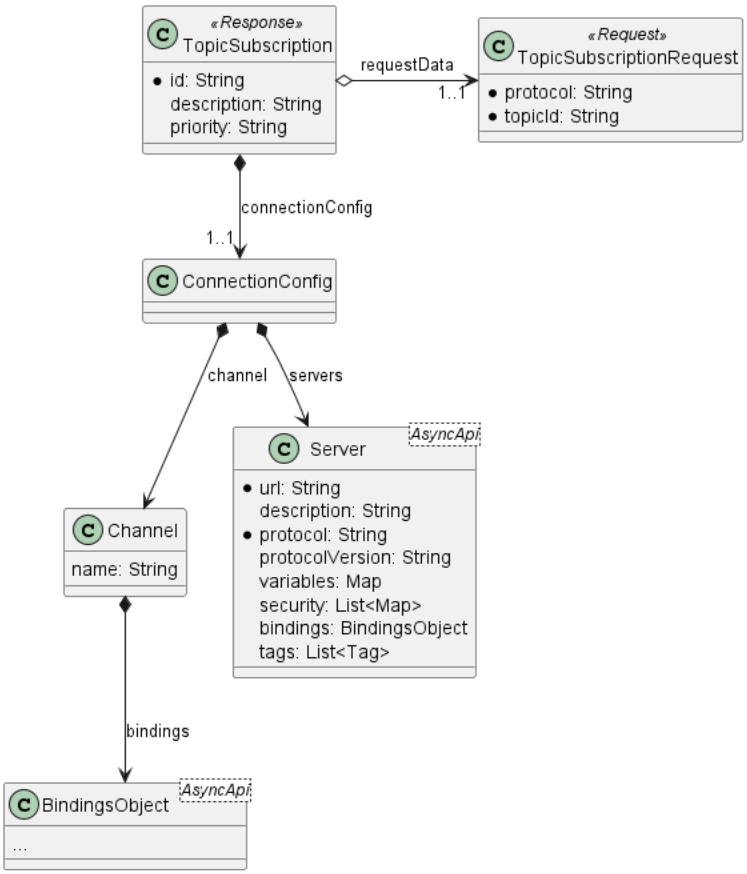

Figure 5. Subscription data model

This data model allows for two types of extensions in the response and request of the API call for UC 3.

5.3.1.1. Response extension

Response extension allows specifying additional information that is required to consume data from the stream using the selected protocol. The extension point for this information is `BindingsObject`. However, this standard does not specify how the extension should be introduced into the model. We recommend using *AsyncAPI Specification* [AsyncAPI] channel binding for well-known asynchronous protocols, as defined in *AsyncApi binding definitions at GitHub* [AsyncApiB].

For example, to provide additional information for *Kafka* channel configuration [AsyncApiBKC] can be used.

The listening below provides an example of a simple configuration for *Kafka*.

```
{
  "bindings": {
    "kafka": {
      "topic": "metrics-kafka",
      "partitions": 1,
      "topicConfiguration": {
        "cleanup.policy": ["compact"],
        "retention.ms": 604800000
      },
      "bindingVersion": "0.4.0"
    }
  }
}
```

This sample provides details on *Kafka* broker configuration for the subscription. It specifically defines the topic, the number of partitions, and the data retention configuration. The complete description of configuration options is beyond the scope of this standard. For more details, please refer to the AsyncAPI bindings configuration and the Kafka documentation.

Appendix A. contains examples of bindings for additional protocols.

5.3.1.2. Request extension

The baseline request payload is very simple as it allows for the selection of a protocol from the list of protocols defined for a given available topic.

In the case of performance-related messages, the subscription is made for all potential subjects (e.g., services) and for the full list of performance metrics defined for that topic.

The extension of the `TopicSubscriptionRequest` serves two purposes:

1. to allow for fine-grained specification of the subject and performance attributes.
2. to allow for the specification of protocol-specific parameters. For example, a `Server` may allow a `Client` to specify the consumer group name for *Kafka*.

An example model definition that accommodates both purposes is shown in Figure 6. The model demonstrates the extension to protocol configuration and allows for fine-grained tuning of the subscription target and performance attributes using the `performanceAttributesNames` list.

The `topicId` effectively serves as a discriminator for the extension model of `TopicSubscriptionRequest`, and the value of the `protocol` attribute becomes the discriminator for the protocol configuration part.

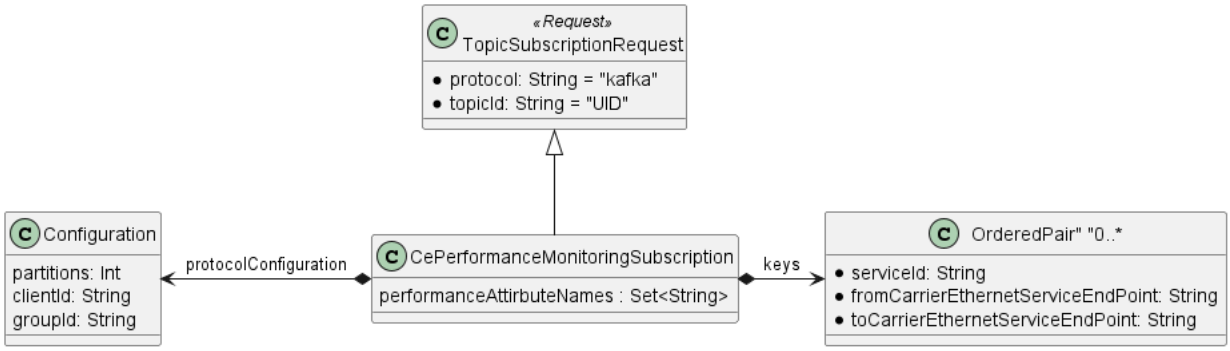

Figure 6. Subscription data model extension example

5.3.2. Message Data Model Extension

In this document, we use the term `Message` to define a unit of information exchanged via streaming. The `Message` is a vessel for `Event` data and additional meta information. The `Message` is defined in [Mplify 133.1] and based on [TMF688].

This section uses a simplified Carrier Ethernet Performance Monitoring data model to illustrate the extension mechanism.

The Message is open for extension using a variant of the TMF extension pattern. The discriminator attribute name is `eventType`. Figure 7 demonstrates how the generic `Message` is specialized to represent a Carrier Ethernet Performance Monitoring model. In this model, `eventType` discriminates a specialized `event` content. The event content defines two distinct components. The identifier of an observation subject (`OrderedPair`) which describes what do we

monitor. In this example, it is an ordered pair of endpoints for a given service. The set of performance measurements (`CePerformanceMetrics`) defines what type of measurements we are interested in for that subject.

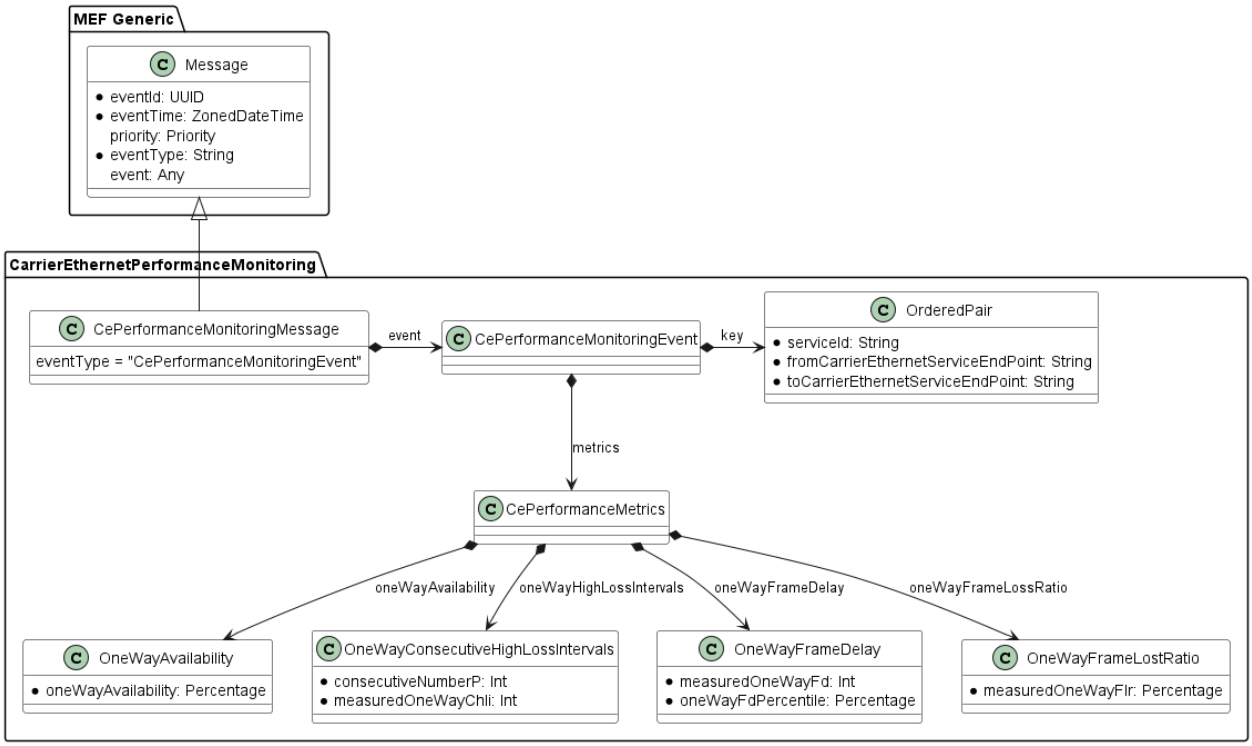

Figure 7. Message specialization with an example event model

The listing below presents an example of a simple payload that conforms to the model defined in Figure 7. In this case, data is encoded using JSON data format. All the attributes of the data model are sent as part of the payload.

```
{
  "eventId": "88b6cb2d-e2ca-4093-9550-90c64096be7b",
  "eventTime": "2023-05-30T14:11:42.515835+02:00",
  "priority": "LOW",
  "event": {
    "key": {
      "serviceId": "service1",
      "fromCarrierEthernetServiceEndPoint": "1",
      "toCarrierEthernetServiceEndPoint": "2"
    },
    "metrics": {
      "oneWayHighLossIntervals": {
        "consecutiveNumberP": 38,
        "measuredOneWayChli": 71
      },
      "oneWayFrameDelay": {
        "measuredOneWayFd": 67,
        "oneWayFdPercentile": 0.14565401998011449
      },
      "oneWayAvailability": {
        "oneWayAvailability": 0.9012261669223176
      },
      "oneWayFrameLossRatio": {
        "measuredOneWayFlr": 0.478900647869665
      }
    }
  },
  "eventType": "CePerformanceStatistics"
}
```

The payload represents a single message taken at a given time for an ordered pair of endpoints for `service1`. The message contains four performance-related metrics with their values.

Please note that JSON is one of many possible data formats and the use of a particular one depends on the particular technology selected for a given subscription. For example, in the case of *Kafka* transport, another popular format is Apache Avro [Avro].

5.4. Model Structural Validation

The structure of the HTTP payloads exchanged via the API endpoints is defined using OpenAPI version 3.0.

[R2] Implementations **MUST** use payloads that conform to these definitions.

5.5. Security Considerations

There must be an authentication mechanism whereby a Server can be assured who a Client is and vice-versa. There must also be authorization mechanisms in place to control what a particular Client is allowed to do and what information may be obtained. However, the definition of the exact security mechanism and configuration is outside the scope of this document. Security considerations are standardized by *LSO API Security Profile* [MEF 128.1].

Please note that to secure access production or consumption of the performance events might require measures that are not in scope for MEF 128.1.

6. API Interactions and Flows

This section provides a detailed insight into the API functionality, use cases, and flows. It starts with Table 4 presenting a list and short description of all business use cases then examples for each of them.

Use Case #	Use Case Name	Use Case Description	Mplify 133.1 mapping
UC 1	Retrieve Available Topics List	A request initiated by the Client to list of all available topics.	UC 42
UC 2	Retrieve Available Topic by an Identifier	A request initiated by the Client to retrieve details for the selected available topic.	UC 41
UC 3	Subscribe To a Topic	A request initiated by the Client to create a new subscription for the topic of interest.	UC 44
UC 4	Unsubscribe From a Topic	A request initiated by the Client to remove a subscription to the topic of interest.	UC 45
UC 5	Retrieve Topic Subscriptions List	A request initiated by the Client to list of all existing subscriptions to the topics.	UC 43
UC 6	Retrieve Topic Subscription By an Identifier	A request initiated by the Client to retrieve details for a selected subscription.	n/a

Table 4. Use cases description

The detailed business requirements of each of the use cases are described in section 12 of [Mplify 133.1].

[R3] The `Server` **MUST** support `application/json` format of information exchange for all the use cases.

6.1. Use case 1: Retrieve Available Topics List

To get detailed information about the available topics, the Client sends a request using `GET /topic` operation with optional filtering criteria.

The flow is a simple request-response pattern, as presented in Figure 8:

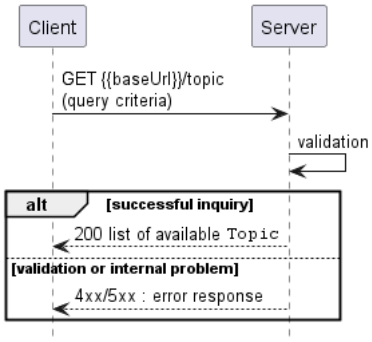

Figure 8. Use case 1: Retrieve Available Topics List

[R4] Server **MUST** return all results matching the filtering criteria [Mplify133.1 R124]

[R5] Server **MUST** return an empty list of Topic entities if there is no topic matching filtering criteria [Mplify133.1 R124]

The response is a list of `Topic`s where the topic data model is presented in Figure 9.

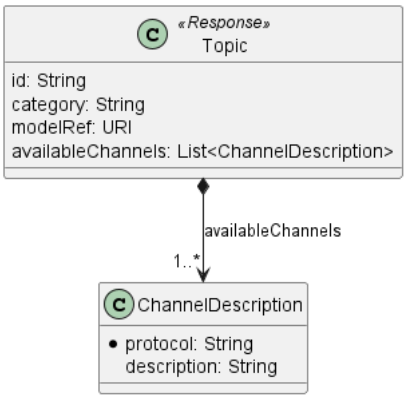

Figure 9. Topic response model

An example response payload is presented below. There are three available channels defined by the `Server`. Each topic can be consumed by a *Kafka* or *WebSocket*-capable `Client` after a successful subscription.

In this example, it is assumed that `modelRef` URI points to the JsonSchema resource that describes the data model introduced in the section above.


```
[
  {
    "id": "dde8b741-f7d3-483e-8c78-5ae84f0c6bb4",
    "category": "IP",
    "modelRef": "https://example.mef.net:7070/schemas/ip/cePerformanceStatistics.1.0.0.schema.json",
    "availableChannels": [
      {
        "protocol": "kafka",
        "description": "Kafka protocol"
      },
      {
        "protocol": "web-socket",
        "description": "WS protocol"
      }
    ]
  },
  {
    "id": "963f6664-5f34-4c3b-a769-6fb6d3dc348f",
    "category": "COMPUTING",
    "modelRef": "https://example.mef.net:7070/schemas/computing/resources.1.0.0.schema.json",
    "availableChannels": [
      {
        "protocol": "kafka",
        "description": "Kafka protocol"
      },
      {
        "protocol": "web-socket",
        "description": "WS protocol"
      }
    ]
  },
  {
    "id": "ae09e5d5-9038-4f90-9531-2ba7c12aa769",
    "category": "COMPUTING",
    "modelRef": "https://example.mef.net:7070/schemas/computing/cpu.1.0.0.schema.json",
    "availableChannels": [
      {
        "protocol": "kafka",
        "description": "Kafka protocol"
      },
      {
        "protocol": "web-socket",
        "description": "WS protocol"
```

```
      }
    ]
  }
]
```

[R6] The `id` **MUST** be unique within the `Server` domain.

6.2. Use case 2: Retrieve Available Topic by an Identifier

To get detailed information about the selected available topics, the Client sends a request using `GET /topic/{{id}}` operation where `{{id}}` is a unique identifier for that topic.

The flow is a simple request-response pattern, as presented in Figure 10.

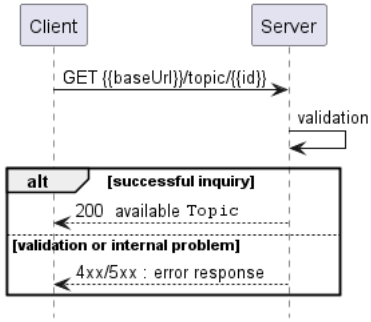

Figure 10. Use case 2: Retrieve Available Topic by an Identifier flow

The data model of the response is presented in Figure 10.

An example response payload is presented below.

```
{
  "id": "dde8b741-f7d3-483e-8c78-5ae84f0c6bb4",
  "category": "IP",
  "modelRef": "https://example.mef.net:7070/schemas/ip/cePerformanceStatistics.1.0.0.schema.json",
  "availableChannels": [
    {
      "protocol": "kafka",
      "description": "Kafka protocol"
    },
    {
      "protocol": "web-socket",
      "description": "WS protocol"
    }
  ]
}
```

6.3. Use case 3: Subscribe To a Topic

Figure 11 illustrates the flow of subscribing to a topic and subsequently consuming from the stream that the client has subscribed to using a broker-based solution.

The specific consumption pattern, whether broker-based (such as *Kafka*, *MQTT*, or *AMQP*) or broker-less (such as *SSE* or *WebSocket*), is transparent to the subscription API.

However, it is important to incorporate transport-specific details in the subscription response, as explained in section 5.3.1.1. Alternatively, these details can be agreed upon during the onboarding process between the `Client` and the `Server`.

A subscription request might trigger various configurations of broker and/or message producer infrastructure.

In this version of the standard, it is assumed that the configuration is finished before the `Server` returns a response to the `Client`. The model extension mechanism might be used to provide additional information that allows for handling the delayed initialization of the infrastructure.

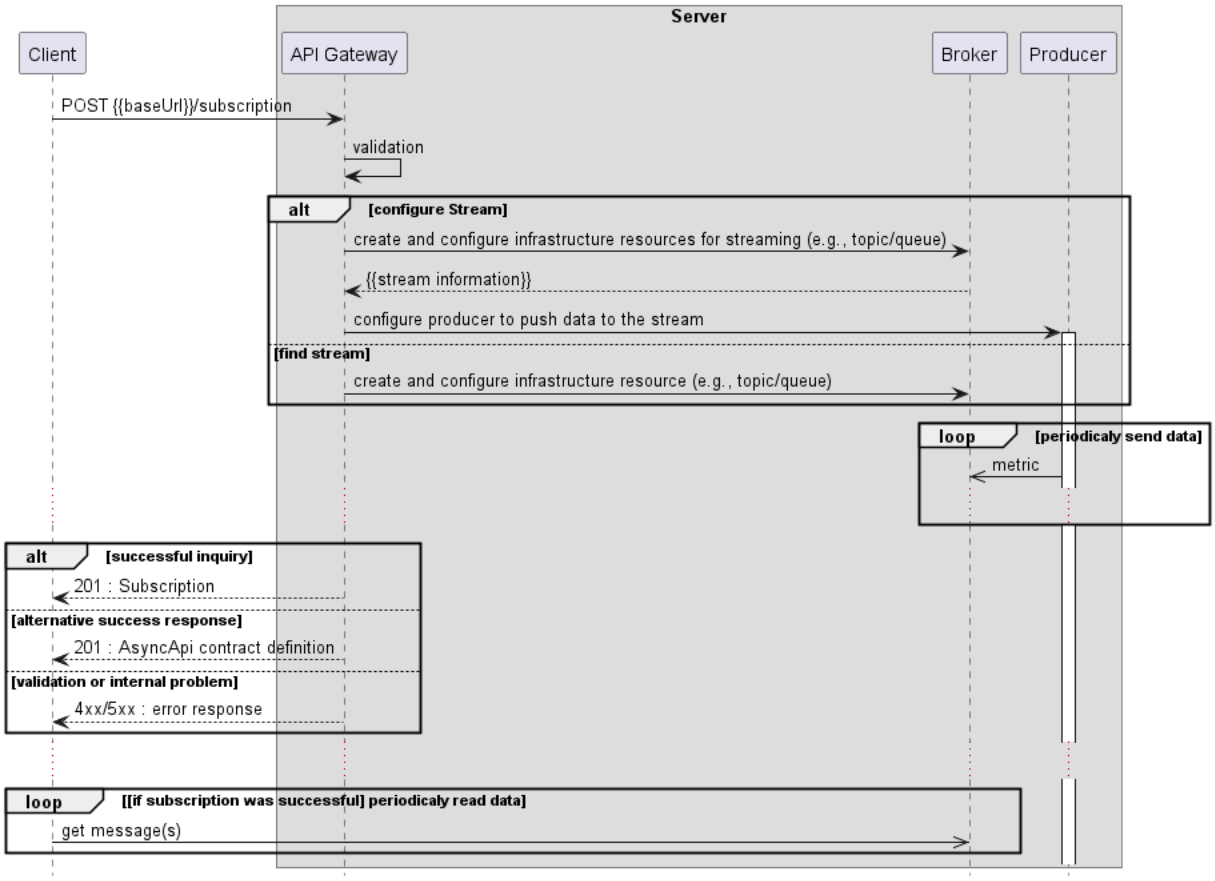

Figure 11. Use case 3: Subscribe To a Topic flow

[R7] The Client request **MUST** include the topic identifier and the protocol to be used for the subscription.

[O1] The Client request **MAY** include additional attributes that are necessary to configure the communication channel for the specified protocol.

[R8] The Server **MUST** indicate whether the request was accepted or declined with the appropriate error code [Mplify133.1 R128]

[R9] The response to the subscription query **MUST** include all details required to consume messages from the configured communication channel. [Mplify133.1 R131]

[R10] The Server **MUST** start streaming if the subscribe operation was successful [Mplify133.1 R131]

The subscription request is initiated by sending a `POST` request to `/subscription` endpoint. An example request payload:

```
{
  "topicId": "dde8b741-f7d3-483e-8c78-5ae84f0c6bb4",
  "protocol": "kafka"
}
```

An example response:

```
{
  "id": "634af680-eca7-499a-8d83-86b61242caeb",
  "connectionConfig": {
    "servers": {
      "kafka-prod": {
        "url": "https://perf.broker.mef.net:9092",
        "protocol": "kafka"
      }
    },
    "channel": {
      "name": "streaming/ce-performance/634af680-eca7-499a-8d83-86b61242caeb",
      "bindings": {
        "kafka": {
          "topic": "ce-performance-metrics-kafka-all",
          "partitions": 1,
          "topicConfiguration": {
            "cleanup.policy": ["compact"],
            "retention.ms": 604800000
          },
          "bindingVersion": "0.4.0"
        }
      }
    }
  },
  "protocol": "kafka",
  "topicId": "dde8b741-f7d3-483e-8c78-5ae84f0c6bb4"
}
```


6.4. Use case 4: Unsubscribe From a Topic

Figure 12 illustrates the flow of unsubscribing from a topic, using subscription id (`sid`) obtained in use case 3.

To unsubscribe, the Client sends a request `DELETE` request to `/subscription/{{sid}}`.

An unsubscribe request might trigger various configuration changes in broker and/or message producer infrastructure.

In this version of the standard, it is assumed that the configuration is finished before the `Server` returns a response to the `Client`.

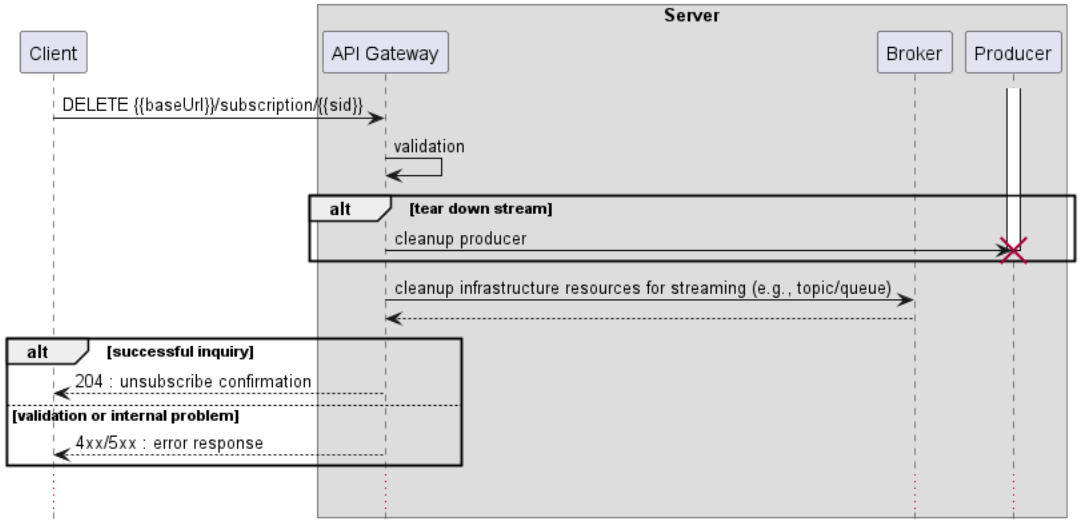

Figure 12. Use case 4: Unsubscribe From a Topic flow

[R11] The Server **MUST** indicate whether a request was accepted or declined with the appropriate error code [Mplify133.1 R133, R134]

[R12] The Server **MUST** stop streaming if an unsubscribe operation was successful [Mplify133.1 R135]

6.5. Use case 5: Retrieve Topic Subscriptions List

To get detailed information about the active subscriptions, the Client sends a request using `GET /subscription` operation with optional filtering criteria.

The flow is a simple request-response pattern, as presented in Figure 13:

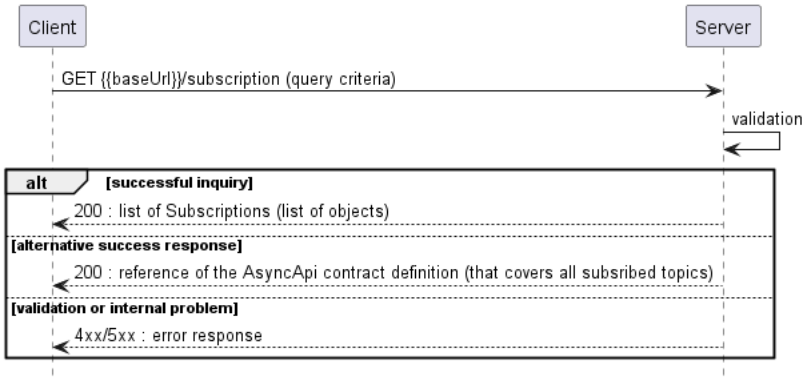

Figure 13. Use case 5: Retrieve Topic Subscriptions List flow

[R13] Server **MUST** return all results matching the filtering criteria [Mplify133.1 R125]

[R14] Server **MUST** return an empty list of subscriptions if there are no subscriptions matching filtering criteria [Mplify133.1 R126]

[O2] Server **MAY** support responses in AsyncAPI response (`application/vnd.aai.asyncapi+json`)

Example JSON response:


```
[
  {
    "id": "31b5485f-cf10-4a82-97f9-c7c1307ef811",
    "priority": "low",
    "connectionConfig": {
      "servers": {
        "kafka": {
          "url": "localhost:29092",
          "protocol": "kafka"
        }
      },
      "channel": {
        "name": "streaming/ce-performance/31b5485f-cf10-4a82-97f9-c7c1307ef811",
        "bindings": {
          "kafka": {
            "topic": "metrics-kafka",
            "partitions": 1,
            "topicConfiguration": {
              "cleanup.policy": ["compact"],
              "retention.ms": 604800000
            },
            "bindingVersion": "0.4.0"
          }
        }
      }
    },
    "protocol": "kafka",
    "topicId": "dde8b741-f7d3-483e-8c78-5ae84f0c6bb4"
  }
]
```


The above response may be represented as an Async API specification. The Async API response includes all channels subscribed by the `Client` and full model definitions. To retrieve that

AsyncAPI response the `Client` sends a request using `GET /subscription` operation with `Accept` header set to `application/vnd.aai.asyncapi+json`. If the `Server` does not support AsyncAPI response it returns `406 Not Acceptable` response.

The above response as Async API payload (JSON encoded):


```
{
  "asyncapi": "2.6.0",
  "info": {
    "title": "CePerformanceStatistics",
    "version": "1.0.0"
  },
  "servers": {
    "kafka": {
      "url": "localhost:29092",
      "protocol": "kafka"
    }
  },
  "channels": {
    "streaming/ce-performance/31b5485f-cf10-4a82-97f9-c7c1307ef811": {
      "subscribe": {
        "description": "read from the channel",
        "bindings": {
          "kafka": {
            "topic": "metrics-kafka",
            "partitions": 1,
            "topicConfiguration": {
              "cleanup.policy": ["compact"],
              "retention.ms": 604800000
            },
            "bindingVersion": "0.4.0"
          }
        },
        "message": {
          "$ref": "#/components/messages/Event"
        }
      }
    }
  },
  "components": {
    "schemas": {
      "CePerformanceMetrics": {
        <<< truncated content >>>
      },
      "Key": {
        <<< truncated content >>>
      },
      "CePerformanceStatistics": {
        "type": "object",
        "properties": {
          "key": {
            "$ref": "#/components/schemas/Key"
          },
          "metrics": {
            "$ref": "#/components/schemas/CePerformanceMetrics"
          }
        }
      }
    },
    "messages": {
      "Event": {
        "payload": {
          "type": "object",
          "properties": {
            "eventId": {
              "description": "Unique identifier for the event.",
              "type": "string",
              "format": "uuid"
            },
            "eventTime": {
              "description": "Date and time when the event occurred.",
              "type": "string",
              "format": "date-time"
            },
            "priority": {
              "description": "Priority of the event.",
              "type": "string",
```

```
              "enum": ["LOW", "MEDIUM", "HIGH"]
            },
            "event": {
              "$ref": "#/components/schemas/CePerformanceStatistics"
            }
          }
        }
      }
    }
  }
}
```

The model used in the above JSON listing is described in section 5.3.2. Please note that there are two possible renderings of the base Message Data Model that might be used in AsyncAPI. The first (presented above) renders all attributes as part of the AsyncAPI message payload. The second renders some of these attributes (e.g. `eventId`) in the header part of the AsyncAPI message.

6.6. Use case 6: Retrieve Topic Subscription By an Identifier

To get detailed information about the subscription to a particular topic, the Client sends a request using `GET /subscription/{{id}}` operation where `{{id}}` is a unique identifier for that subscription.

The flow is a simple request-response pattern, as presented in Figure 14.

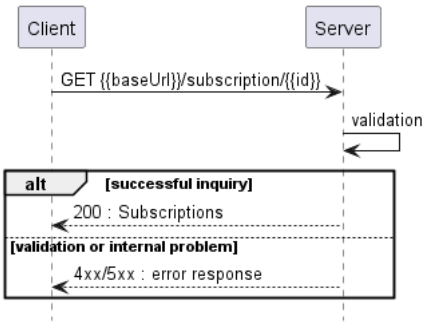

Figure 14. Use case 6: Retrieve Topic Subscription By an Identifier flow

7. API Details

7.1. Management API Data model

7.1.1. Topic

The topic model exposes information about the available topics.

7.1.1.1. `enum` Category

Description: The category of the topic. This can be used to group topics based on their characteristics.

Value	Mplify 133.1
Layer 1	LAYER 1
Ethernet	ETHERNET
IP	IP
SD-WAN	SD-WAN
Computing	COMPUTING
Storage	STORAGE
Memory	MEMORY

7.1.1.2. Type ChannelDescription

Description:

Name	Type	M/O	Description	Mplify 133.1
protocol	string	M	Name of a technical protocol allowing for consumption	n/a
description	string	O	Human-friendly description of the protocol	n/a

7.1.1.3. Type Topic

Description: Provides metadata describing a topic available for subscription. This object is used to define available consumption mechanisms and the data model.

Name	Type	M/O	Description	Mplify 133.1
id	string	O	An identifier for the topic. This can be used to uniquely identify the topic within the Server system.	Topic Identifier
category	Category	O	The category of the topic. This can be used to group topics based on their characteristics.	Topic Category

Name	Type	M/O	Description	Mplify 133.1
modelRef	uri *format = uri*	O	A reference to a model that describes the structure of the data associated with the topic.	Indirect mapping to 'Service Specific Attributes'
available Channels	Channel Description[] *minItems = 1*	O	An array of channel descriptions that provide information about the channels through which the topic can be accessed.	Indirect mapping to 'Service Specific Attributes'

[R14] The `modelRef` **MUST** be a valid `URI` that references a data model that defines a contract between Client and Server for the data exchange for all subscriptions obtained for this topic via any of the defined channels.

[R15] `Server` **MUST** define at least one available channel for each `Topic`.

7.1.2. Subscription

The subscription models for request and response are depicted in Figure 5. The model allows requesting (`TopicSubscriptionRequest`) consumption of data from an available topic. After a successful subscription provides all details allowing for consumption (`TopicSubscription`)

7.1.2.1. Type Channel

Description: Defines the specific protocol bindings and configurations for the channel.

Name	Type	M/O	Description	Mplify 133.1
name	uri-template *minLength = 1* *format = uri-template*	O	The name of the channel through which the stream data is transmitted.	
bindings	bindingsObject	O	Defines the specific protocol bindings and configurations for the channel. We reuse AsyncAPI definition of this type	

7.1.2.2. Type ConnectionConfig

Description: Configuration settings for establishing a connection to the stream.

Name	Type	M/O	Description	Mplify 133.1
servers	servers	M	List of the servers through which the subscription is available. We reuse the AsyncAPI definition of this type	n/a
channel	Channel	O		n/a

7.1.2.3. Type Subscription

Description: Provides stream metadata information for stream consumption.

Name	Type	M/O	Description	Mplify 133.1
id	string	M	A unique identifier for the stream.	Stream Identifier
description	string	O	An explanatory description of the stream.	Description
priority	string	O	The priority level of the stream. Can be high, medium, or low.	priority
connectionConfig	ConnectionConfig	O		Addresses connection configuration concerns listed in Table 68

7.1.2.4. Type TopicSubscription

Description: Information about the subscription to a specific topic

Inherits from:

- TopicSubscriptionRequest
- Subscription

7.1.2.5. Type TopicSubscriptionRequest

Description:

Name	Type	M/O	Description	Mplify 133.1
protocol	string	M	Name of the protocol consumer is intended to use to consume data from `topicId`. The name of the protocol must be one of the defined for the topic.	n/a
topicId	string	M	Identifier of the topic consumer wants to subscribe to	Topic Identifier

7.1.3. Error models

7.1.3.1. Type Error

Description: Standard Class used to describe API response error Not intended to be used directly. The `code` in the HTTP header is used as a discriminator for the type of error returned in runtime.

Name	Type	Description
message	string	Text that provides mode details and corrective actions related to the error. This can be shown to a client user.
reason*	string *maxLength = 255*	Text that explains the reason for the error. This can be shown to a client user.

Name	Type	Description
referenceError	uri *format* = *uri*	URL pointing to documentation describing the error

7.1.3.2. Type Error400

Description: Bad Request. (https://tools.ietf.org/html/rfc7231#section-6.5.1)

Inherits from:

- Error

Name	Type	Description
code*	Error400Code	

7.1.3.3. `enum` Error400Code

Description: One of the following error codes:

- missingQueryParameter: The URI is missing a required query-string parameter
- missingQueryValue: The URI is missing a required query-string parameter value
- invalidQuery: The query section of the URI is invalid.
- invalidBody: The request has an invalid body

7.1.3.4. Type Error401

Description: Unauthorized. (https://tools.ietf.org/html/rfc7235#section-3.1)

Inherits from:

- Error

Name	Type	Description
code*	Error401Code	

7.1.3.5. `enum` Error401Code

Description: One of the following error codes:

- missingCredentials: No credentials provided.
- invalidCredentials: Provided credentials are invalid or expired

7.1.3.6. Type Error403

Description: Forbidden. This code indicates that the server understood the request but refuses to authorize it. (https://tools.ietf.org/html/rfc7231#section-6.5.3)

Inherits from:

- Error

Name	Type	Description
code*	Error403Code	

7.1.3.7. enum Error403Code

Description: This code indicates that the server understood the request but refuses to authorize it because of one of the following error codes:

- accessDenied: Access denied
- forbiddenRequester: Forbidden requester
- tooManyUsers: Too many users

7.1.3.8. Type Error404

Description: Resource for the requested path not found. (https://tools.ietf.org/html/rfc7231#section-6.5.4)

Inherits from:

- Error

Name	Type	Description
code*	string	The following error code: - notFound: A current representation for the target resource not found

7.1.3.9. Type Error422

Description: Unprocessable entity due to a business validation problem. (https://tools.ietf.org/html/rfc4918#section-11.2)

Inherits from:

- Error

Name	Type	Description
code*	Error422Code	
propertyPath	string	A pointer to a particular property of the payload that caused the validation issue. It is highly recommended that this property should be used. Defined using JavaScript Object Notation (JSON) Pointer (https://tools.ietf.org/html/rfc6901).

7.1.3.10. enum Error422Code

Description: One of the following error codes:

- missingProperty: The property that was expected is not present in the payload
- invalidValue: The property has an incorrect value
- invalidFormat: The property value does not comply with the expected value format
- referenceNotFound: The object referenced by the property cannot be identified in the target system
- unexpectedProperty: Additional, not expected property has been provided

- tooLargeDataset: Requested entity will produce too many data
- tooManyRecords: The number of records to be provided in the response exceeds the threshold
- tooManyRequests: The number of simultaneous requests from one API client exceeds the threshold
- otherIssue: Other problem was identified (detailed information provided in a reason)

7.1.3.11. Type Error500

Description: Internal Server Error. (https://tools.ietf.org/html/rfc7231#section-6.6.1)

Inherits from:

- Error

Name	Type	Description
code*	string	The following error code: - internalError: Internal server error - the server encountered an unexpected condition that prevented it from fulfilling the request.

7.2. Message model

The Message model is not part of the Stream Management API specification. Instead, it is a model that describes the whole data payload exchanged through a stream the Client is subscribed to.

The Message is open for extension. The details of the extension mechanism are described above.

7.2.1. Message

The Message is produced by the `Server` and consumed by the `Client`. The object is meant to be extended. By convention, a specialization of the `Message` should introduce an `event` attribute whose structure conforms to the data model definition indicated by `eventType`.

Note: The `eventType` is a discriminator and plays an analogous role to `@type` attribute used in other Mplify API. We use `eventType` to be compatible with the model definition from [TMF688].

Name	Type	Description	Mplify 133.1
eventId*	string	The unique identifier of the event	Event ID
eventTime	string	Time of the event occurrence	Event Time
eventType*	string	Event type - discriminator that allows for de-marshaling of the event-specific data, which is added by event specialization objects	Event Type
priority	string	The priority of the event	Priority
description	string	Free text that might be associated with the event	Description

8. References

- AsyncApi AsyncAPI Specification v2.6.0, February 2023
- AsyncApiB AsyncApi binding definitions Github, accessed June 2023
- AsyncApiBKC AsyncApi binding definitions for kafka channel - JsonSchema, accessed June 2023
- Avro Apache Avro v1.11.1, accessed June 2023
- ITU X.734 Information Technology - Open Systems Interconnection - Systems Management: Event Report Management Function, November 2013
- OAS-v3 Open API 3.0, February 2020
- MEF 55.1, Lifecycle Service Orchestration (LSO): Reference Architecture and Framework, February 2021
- MEF 80, Quote Management Requirements and Use Cases, July 2021
- Mplify 133.1, Allegro, Interlude and Legato Fault Management and Performance Monitoring BR&UC, October 2025
- [REST] Chapter 5: Representational State Transfer (REST), Fielding, Roy Thomas, Architectural Styles and the Design of Network-based Software Architectures (Ph.D.).
- RFC 2119, Key words for use in RFCs to Indicate Requirement Levels, March 1997
- RFC 3986, Uniform Resource Identifier (URI): Generic Syntax, January 2005
- RFC 8174, Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words, May 2017
- TMF688, TMF688 Event Management API User Guide v4.0.0

Appendix A. Channel binding examples

Appendix A. provides selected examples of binding definitions for various transport protocols.

Kafka binding example

Data model: https://github.com/asyncapi/bindings/blob/master/kafka/json_schemas/channel.json

```
{
  "bindings": {
    "kafka": {
      "topic": "ce-performance-metrics-kafka-all",
      "partitions": 1,
      "topicConfiguration": {
        "cleanup.policy": ["compact"],
        "retention.ms": 604800000
      },
      "bindingVersion": "0.4.0"
    }
  }
}
```

AMQP binding example

Data model: https://github.com/asyncapi/bindings/blob/master/amqp/json_schemas/channel.json

```
{
  "bindings": {
    "amqp": {
      "is": "queue",
      "queue": {
        "name": "ce-performance-metrics-all",
        "durable": true,
        "vhost": "/"
      },
      "bindingVersion": "0.2.0"
    }
  }
}
```

Web Socket binding example

Data model: https://github.com/asyncapi/bindings/blob/master/websockets/json_schemas/channel.json

```
{
  "bindings": {
    "web-socket": {
      "method": "POST",
      "bindingVersion": "0.1.0"
    }
  }
}
```

The concrete URL for the POST is provided in the `servers` section not shown in this Appendix.

Appendix B Acknowledgments

Mike **BENCHECK**

Michał **ŁĄCZYŃSKI**

Bartosz **MICHALIK**

Dominik **OGRODNIK**

Jack **PUGACZEWSKI**

Boris **TRINAJSTIC**