SMPTE PUBLIC CD

Microservice Status Reporting and Logging

Table of Contents Page

Foreword

SMPTE (the Society of Motion Picture and Television Engineers) is an internationally-recognized standards developing organization. Headquartered and incorporated in the United States of America, SMPTE has members in over 80 countries on six continents. SMPTE"s Engineering Documents, including Standards, Recommended Practices, and Engineering Guidelines, are prepared by SMPTE"s Technology Committees. Participation in these Committees is open to all with a bona fide interest in their work. SMPTE cooperates closely with other standards-developing organizations, including ISO, IEC and ITU.

SMPTE Engineering Documents are drafted in accordance with the rules given in its Standards Operations Manual. This SMPTE Engineering Document was prepared by Technology Committee 34CS.

Normative text is text that describes elements of the design that are indispensable or contains the conformance language keywords: "shall", "should", or "may". Informative text is text that is potentially helpful to the user, but not indispensable, and can be removed, changed, or added editorially without affecting interoperability. Informative text does not contain any conformance keywords.

All text in this document is, by default, normative, except: the Introduction, any section explicitly labeled as "Informative" or individual paragraphs that start with "Note:"

The keywords "shall" and "shall not" indicate requirements strictly to be followed in order to conform to the document and from which no deviation is permitted.

The keywords "should" and "should not" indicate that, among several possibilities, one is recommended as particularly suitable, without mentioning or excluding others; or that a certain course of action is preferred but not necessarily required; or that (in the negative form) a certain possibility or course of action is deprecated but not prohibited.

The keywords "may" and "need not" indicate courses of action permissible within the limits of the document.

The keyword "reserved" indicates a provision that is not defined at this time, shall not be used, and may be defined in the future. The keyword "forbidden" indicates "reserved" and in addition indicates that the provision will never be defined in the future.

A conformant implementation according to this document is one that includes all mandatory provisions ("shall") and, if implemented, all recommended provisions ("should") as described. A conformant implementation need not implement optional provisions ("may") and need not implement them as described.

Unless otherwise specified, the order of precedence of the types of normative information in this document shall be as follows: Normative prose shall be the authoritative definition; Tables shall be next; then formal languages; then figures; and then any other language forms.

Introduction

This section is entirely informative and does not form an integral part of this Engineering Document.

Media Cloud Microservices Architecture (MCMA) is an initiative of the European Broadcasting Union (EBU) that has focused on approaches for microservices in media. One of the key areas it has addressed is the area of logging, as there is no common approach in our industry currently, leading to challenging implementations of microservices across vendor solutions. This document uses their approach (as documented in their publication “MCMA Logging”) as inspiration for this standardized methodology for Microservice Status Reporting and Logging.

At the time of publication, no notice had been received by SMPTE claiming patent rights essential to the implementation of this Engineering Document. However, attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. SMPTE shall not be held responsible for identifying any or all such patent rights.

1 Scope

This document describes a status reporting and logging method to cope with the challenges that a microservice architecture brings. It starts with an overview, followed by a short introduction about jobs, the log entry structure, transaction tracker, and details about how job progress and status is logged.

2 Normative References

The following Standard contains provisions that, through reference in this text, constitute provisions of this standard. Dated references require that the specific edition cited shall be used as the reference. Undated citations refer to the edition of the referenced document (including any amendments) current at the date of publication of this document. All standards are subject to revision, and users of this engineering document are encouraged to investigate the possibility of applying the most recent edition of any undated reference.

IETF RFC7807, Problem Details for HTTP APIs, Internet Engineering Task Force, March 2016

ISO 8601, Date and Time Format, International Organization for Standardization, 2017

3 Terms and Definitions

For the purposes of this document, the terms and definitions given in [external reference(s)] and the following apply:

3.1 Event-Driven Architecture

a deterministic architecture where work is performed in response to specified occurrences or state changes

Note: When given events are detected ECA (Event-Condition-Action) rules are triggered, and appropriate actions taken or functions implemented.

3.2 Rule-based Architecture

a deterministic architecture where work is performed according to a set of rules

3.3 Function as a Services (FaaS)

a category of cloud computing services that allows the creation of very granular tasks (functions) as small, self-contained pieces of code that can be concatenated together in a rule-based, event-driven architecture to perform automated processes

3.4 Hypertext Transfer Protocol (HTTP)

a stateless application-level request/response protocol that uses extensible semantics and self-descriptive message payloads for flexible interaction with network-based hypertext information systems

3.5 JavaScript Object Notation (JSON)

a lightweight, text-based, language-independent data-interchange format used to represent structured data

3.6 Job

a stored set of processes that may be used to perform a specific piece of work or operation on media files as needed

3.7 Log

a record of statuses collected during job execution

3.8 Status Reporting

the real-time reporting of the success, failure, or other status of the job execution

3.9 Tracker

a token (including a unique identifier) passed between the microservices involved in the completion of a transaction, allowing their correlation

3.10 Uniform Resource Locator (URL)

a unique identifier for a web resource that specifies its location on a computer network and a mechanism for retrieving it

4 Overview (Informative)

The concepts that follow in this document are more easily understood if seen in terms of data flow. As shown in Figure 1, a job is created with a Tracker identifier. A Job Processor service then invokes the appropriate service(s) with that Tracker ID. The service(s) are then executed (examples shown in Figure 1 include AI (Artificial Intelligence) and Transcode services, but these could be anything). This all generates log entries, which can be viewed and analyzed in the log repository of your choice.

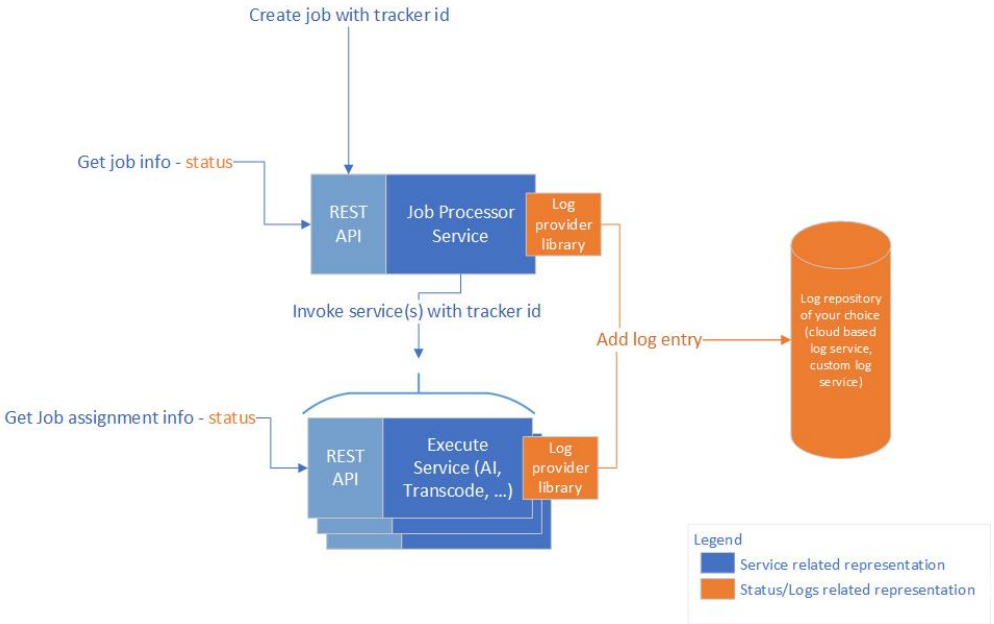

Figure 1: Dataflow representation of jobs and status reporting/logging

5 Jobs

A job is a stored set of processes that may be used to perform a specific piece of work or operation on media files repeatedly and as needed. Jobs shall be stored in a library (the central job processor) and contain the information necessary for them to be appropriately assigned to a service for execution.

Note: Properties shown below are JSON fields.

A job shall have the properties shown in Table 1.

Table 1: Job Properties

Property	Type	Required for Logging?	Clause in this doc
@type	string	Yes	5.1
id	string	Yes	5.2
jobProfile	string	Yes	5.3
jobInput	object	Yes	5.4
status	string	Yes	5.5
error	Problem Detail (RFC 7807)	Yes	5.6
jobOutput	object	Yes	5.7
progress	number	No	5.8
parentId	string	No	5.9
timeout	number	No	5.10
deadline	string	No	5.11
tracker	McmaTracker	No	5.12
notificationEndpoint	NotificationEndpoint	No	5.13

5.1 @type

Indicates the general type of job and used for routing the request to the correct service. Examples are ‘CaptureJob’, ‘TransferJob’, ‘TransformJob’, ‘QAJob’, ‘AIJob’, etc.

5.2 id

URL where the job is stored and retrievable with a HTTP GET request

5.3 jobProfile

URL where the job profile is stored and retrievable with a HTTP GET request. The job profile indicates which specific operation is going to be performed. It also describes which input and output parameters are expected.

5.4 jobInput

The jobInput property contains a collection of input parameters, which the executing service will process.

5.5 status

Status of the job, which can be ‘New’, ‘Queued’, ‘Scheduled’, ‘Running’, ‘Completed’, ‘Failed’, or ‘Canceled’.

5.6 error

Detailed information provided when the job is in a ‘Failed’ state. (Absent when the job is not in a ‘Failed’ state.)

This object closely follows the RFC 7807 specification where applicable (https://tools.ietf.org/html/rfc7807). RFC 7807 defines a "problem detail" as a way to carry machine-readable details of errors in a HTTP response. Jobs represent long running transactions (as opposed to a short-lived HTTP request and response).

According to RFC 7807, the problem detail object can have various members, including possible extensions. The only member not used is “status”. This describes “The HTTP status code ([RFC7231], Section 6) generated by the origin server for this occurrence of the problem” which is not applicable in this (serverless) context.

5.7 jobOutput

Property which will contain the output produced by the executing service.

5.8 progress

Property which may be filled out by the executing service to indicate progress percentage, in values from 0 to 100.

5.9 parentId

Identifier set by the creator of the job to identify the parent job, if required.

5.10 timeout

Time in minutes indicating the maximum duration of the job. If the job does not complete before this timeout value is reached, the job status shall change to Failed.

5.11 deadline

Date/time in ISO 8601 format, indicating the deadline for the job to have been completed. If the job does not complete before this deadline value is reached, the job status shall change to Failed.

5.12 tracker

Refer to clause 6 of this document.

5.13 notificationEndpoint

This property is used to provide a notification endpoint to which updates to the job may be communicated.

Note: This exploits the event-driven, serverless architecture upon which this approach is based.

6 Transaction Tracker

In a microservice architecture, it is important to be able to correlate logs from multiple microservices that belong to the same distributed ‘transaction’, meaning here a specific implementation of a stored job. In order to achieve this, a ‘Tracker’ shall be used, which is basically a customizable token that is passed from one microservice to the next as an HTTP GET parameter, an HTTP Header, or as part of the message payload. The receiving service shall process this tracker and use it when creating log entries.

This token currently has two properties, ‘id’ and ‘label’, which are reflected in the log entries as ‘trackerId’ and ‘trackerLabel’. The trackerId shall be a unique id given to the transaction and the trackerLabel is intended to be human readable and may or may not be unique.

There is also an optional property for custom items, as shown in clause 8.

Note: Properties listed below are JSON fields.

A Transaction Tracker shall have properties shown in Table 2.

Table 2: Transaction Tracker Properties

Property	Type	Clause in this doc
@type	string	6.1
id	string	6.2
label	string	6.3
custom	object	6.4

6.1 @type

Must have the value ‘McmaTracker’ which indicates that it is an McmaTracker object.

6.2 id

Unique id for this transaction

6.3 label

Human readable label this transaction

6.4 custom

JSON object containing key value pairs of custom properties, which will be mapped to custom tracker properties in the log entry. Both key and value properties must be of type string.

7 Tracker Example (Informative)

```
{
    "@type": "McmaTracker",
    "id": "6fcf8dd2-a4dc-4282-8828-58631a37d41f",
    "label": "Workflow 'test7' with file '2015_GF_ORF_00_18_09_conv.mp4'",
}
```

8 Custom Properties

The tracker may be extended if required. To do this, an arbitrary number of extra properties may be added to the tracker in the form of custom name and value pairs. They shall be in camel case with the prefix “tracker” (e.g. Custom Property for ingest name will appear as a log entry with a name of trackerIngestName).

8.1.1. Custom Properties Example (Informative)

This example illustrates the tracker in the case of extra information being added regarding a media asset, in this example, ingestName, filename, and ingestDescription.

```
{
    "@type": "McmaTracker",
    "id": "6fcf8dd2-a4dc-4282-8828-58631a37d41f",
    "label": "Workflow 'test7' with file '2015_GF_ORF_00_18_09_conv.mp4'",
    "custom": {
        "ingestName": "test7",
        "fileName": "2015_GF_ORF_00_18_09_conv.mp4",
        "ingestDescription": "test7"
    }
}
```

The corresponding log entry would then become:

```
{
    "type": "DEBUG",
    "level": 500,
    "source": "job-processor-worker",
    "requestId": "50ec3f76-aa35-41d8-bb0d-d1bf627af139",
    "timestamp": "2020-06-28T00:00:40.559Z",
    "message": "Handling worker operation 'DeleteJob'...",
    "trackerId": "6fcf8dd2-a4dc-4282-8828-58631a37d41f",
    "trackerLabel": "Workflow 'test7' with file '2015_GF_ORF_00_18_09_conv.mp4'",
```

```
    "trackerIngestName": "test7",
    "trackerFileName": "2015_GF_ORF_00_18_09_conv.mp4",
    "trackerIngestDescription": "test7"
}
```

This provides the capability of searching log records by these new fields.

9 Logs

Log entries shall be written in JSON, as shown below. This ensures that the logs will be machine readable.

Log entries shall have the properties shown in Table 3.

Table 3: Log Properties

Property	Type	Clause in this doc
type	string	9.1
level	number	9.2
source	string	9.3
requestId	tring	9.4
Timestamp	string	9.5
trackerId	string	9.6
trackerLabel	string	9.7
tracker	any	9.8
message	any	9.9

9.1 type

Indicates the log entry type and is used to classify log messages.

A complete list of predefined types that shall be supported is shown below. In addition, this field is freely extendable if needed, with optionally supported properties.

“FATAL”, “ERROR”, “WARN”, “INFO”, and “DEBUG” indicate the severity of a message.

“FUNCTION_START” and “FUNCTION_END” indicate the start and end of the execution of a function

"JOB_START", "JOB_UPDATE", and "JOB_END" indicate status updates regarding Jobs that are executed

9.2 level

Indicates the log level of a log entry and must be a positive number. The log level is used for filtering log entries. The higher the log level the more verbose the log entry is considered. Each log type has an associated level, as shown in Table 4.

Table 4: Log Levels

Type	Level
FATAL	100
ERROR	200
WARN	300
INFO	400
DEBUG	500
FUNCTION_START	450
FUNCTION_END	450
JOB_START	400
JOB_UPDATE	400
JOB_END	400

9.3 source

Indicates the source of the log entry, i.e. the service / component that is writing to the logs. This is particularly useful when multiple services are writing to the same logs

9.4 requestId

Unique identifier indicating the current request execution. In a multithreaded or FaaS (Function as a Service) environment, the same function may be executed concurrently and therefore may be writing logs

at the same moment with the same ‘source’. This property can then be used to distinguish between them. In case the programming model has no concept of a requestId, another identifier, e.g. threadId, may be used.

9.5 timestamp

Timestamp with millisecond precision as specified in ISO 8601.

9.6 trackerId

Unique id for this transaction. More information about this is located in the Transaction Tracker clause.

9.7 trackerLabel

Human readable label transaction. More information about this is located in the Transaction Tracker clause.

9.8 tracker

Customizable properties related to the transaction tracker, which allows an arbitrary number of tracker properties, identified with the prefix “tracker”. More information about this is located in the Transaction Tracker clause.

9.9 message

Message that is to be logged. Note that the message may be of any type, so it may be a string, but it may also be another JSON object.

10 Log Entry Example (Informative)

Shown below is an example, illustrating a log entry

```
{
    "type": "DEBUG",
    "level": 500,
    "source": "job-processor-worker",
    "requestId": "50ec3f76-aa35-41d8-bb0d-d1bf627af139",
    "timestamp": "2020-06-28T00:00:40.559Z",
    "message": "Handling worker operation 'DeleteJob'...",
    "trackerId": "6fcf8dd2-a4dc-4282-8828-58631a37d41f",
    "trackerLabel": "Workflow 'test7' with file '2015_GF_ORF_00_18_09_conv.mp4'",
    "trackerIngestName": "test7",
    "trackerFileName": "2015_GF_ORF_00_18_09_conv.mp4",
    "trackerIngestDescription": "test7"
}
```

11 Job Status Logging

Logging the progress and status of jobs is an important aspect of job management.

A “job processor” is a service which stores all the jobs (of any kind) that would need to be executed. In this service, whenever a job is starting, running or completing/failing, a log entry shall be created.

For these log entries, JSON shall be used as the message format, as this allows for easy machine parsing of the messages.

Log entries shall have the properties, shown in Table 5.

Table 5: Job Status Logging Properties

Property	Type	Clause in this doc
jobId	URL	11.1
jobType	string	11.2
jobProfile	URL	11.3
jobProfileName	object	11.4
jobExecution	URL	11.5
jobAssignment	URL	11.6
jobInput	object	11.7
jobStatus	object	11.8
jobError	Problem Detail as specified in IETF RFC 7807	11.9
jobActualStartDate	string	11.10
jobActualEndDate	string	11.11
jobActualDuration	number	11.12
jobOutput	object	11.13

11.1 jobId

URL pointing to the job instance in the job processor

11.2 jobType

Indicates the job type

11.3 jobProfile

URL pointing to the job profile used by the job

11.4 jobProfileName

Name of the job profile used by the job.

11.5 jobExecution

URL pointing to the jobExecution instance in the job processor

11.6 jobAssignment

URL pointing to the jobAssignment instance in the executing service

11.7 jobInput

Collection of input parameters that were provided in the job when it was created

11.8 jobStatus

Status of the job. Valid values are: ‘New’, ‘Queued’, ‘Scheduled’, ‘Running’, ‘Completed’, ‘Failed’, or ‘Canceled’

11.9 jobError

Detailed information about the problem which caused the job to get into the ‘Failed’ state. Absent when the job is not in a ‘Failed’ state

11.10 jobActualStartDate

Date in ISO 8601 format when job was queued for processing

11.11 jobActualEndDate

Date in ISO 8601 format when job completed, failed or canceled

11.12 jobActualDuration

Job duration in milliseconds

11.13 jobOutput

Collection of output results of the job that was executed

12 Log Entry example with job status data (Informative)

In the following example, a 'TransformJob' with Profile 'CreateProxy' failed with a job error, which is an instance of the RFC 7807 ProblemDetail, describing that it failed to run ffmpeg.


```
{
    "type": "JOB_END",
    "level": 400,
    "source": "job-processor-worker",
    "requestId": "4d2e0097-f311-4056-ba99-d7f6c10cd504",
    "timestamp": "2020-06-26T18:59:18.873Z",
    "message": {
        "jobId": "https://job-processor/dev/jobs/36391c5c-d999-4232-9c13-eb2d3d9e9251",
        "jobType": "TransformJob",
        "jobProfile": "https://service-registry/dev/job-profiles/628de386-078f-42b2-aab3-
3cf21b9d5c74",
        "jobProfileName": "CreateProxy",
        "jobExecution": "https://job-processor/dev/jobs/36391c5c-d999-4232-9c13-
eb2d3d9e9251/executions/1",
        "jobAssignment": "https://transcode-service/dev/job-assignments/e5c32653-d887-4bd1-bd6c-
82741d575a05",
        "jobInput": {
            "outputLocation": {
                "bucket": "ch.ebu.mcma.eu-west-1.dev.website",
                "keyPrefix": "ProxyJobResults/",
                "@type": "FolderLocator"
            },
            "inputFile": {
                "bucket": "ch.ebu.mcma.eu-west-1.dev.temp",
                "key": "temp/proxy.mp4",
                "@type": "FileLocator"
            },
            "@type": "JobParameterBag"
        },
        "jobStatus": "Failed",
        "jobError": {
            "@type": "ProblemDetail",
            "type": "uri://mcma.ebu.ch/rfc7807/generic-job-failure",
            "title": "Generic Job failure",
            "detail": "Failed to run ffmpeg"
        },
        "jobActualStartDate": "2020-06-26T18:59:14.043Z",
        "jobActualEndDate": "2020-06-26T18:59:16.675Z",
        "jobActualDuration": 2632,
        "jobOutput": {
            "@type": "JobParameterBag"
        }
    },
    "trackerId": "021bbb38-9917-42e5-b14d-c272c21b972f",
    "trackerLabel": "Workflow 'test8' with file '2015_GF_ORF_00_18_09_conv.mp4'",
    "trackerIngestName": "test8",
    "trackerFileName": "2015_GF_ORF_00_18_09_conv.mp4",
    "trackerIngestDescription": "test8"
}
```


13 Source Code

Libraries containing the source code for implementing the microservice status reporting and logging described in this document can be found here: https://github.com/ebu/mcma-libraries/releases/tag/v0.12.1.

Bibliography

MCMA Logging, European Broadcasting Union, 2020

Additional resources can also be found on GitHub (https://github.com/ebu/mcma-libraries).

Example implementations using the Node.JS libraries can be found here: https://github.com/ebu/mcma-projects.