SMPTE STANDARD

Microservice Status Reporting and Logging

Table of Contents

Page

Foreword **3**
Introduction **3**
1 Scope **3**
2 Conformance **3**
3 Normative References **4**
4 Terms and Definitions **4**
5 Overview (Informative) **6**
6 Jobs **7**
6.1 Job Overview 7
6.2 @type 7
6.3 id 7
6.4 jobProfile 7
6.5 jobInput 8
6.6 status 8
6.7 error 8
6.8 jobOutput 8
6.9 progress 8
6.10 parentId 8
6.11 timeout 8
6.12 deadline 8
6.13 tracker 8
6.14 notificationEndpoint 8
7 Transaction Tracker **9**
7.1 Transaction Tracker Overview 9
7.2 @type 9
7.3 id 9
7.4 label 9
7.5 custom 9
8 Tracker Example (Informative) **10**

Copyright © 2025 by THE SOCIETY OF MOTION PICTURE AND TELEVISION ENGINEERS
50 Main St., Suite 1000, White Plains, NY 10606, (914) 761-1100
Approved August 29, 2025

9 Custom Properties 10
9.1 Custom Properties Overview 10
9.2 Custom Properties Example (Informative) 10
10 Logs 11
10.1 Log Entries Overview 11
10.2 type 11
10.3 level 12
10.4 source 12
10.5 requestId 12
10.6 timestamp 12
10.7 trackerId 12
10.8 trackerLabel 12
10.9 tracker 12
10.10 message 12
11 Log Entry Example (Informative) 13
12 Job Status Logging 13
12.1 Job Status Logging Overview 13
12.2 jobId 14
12.3 jobType 14
12.4 jobProfile 14
12.5 jobProfileName 14
12.6 jobExecution 14
12.7 jobAssignment 14
12.8 jobInput 14
12.9 jobStatus 14
12.10 jobError 14
12.11 jobActualStartDate 14
12.12 jobActualEndDate 14
12.13 jobActualDuration 14
12.14 jobOutput 14
13 Log Entry example with job status data (Informative) 15
14 Source Code (Informative) 15
Bibliography 16

© SMPTE 2025 – All Rights Reserved

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.

Introduction

This clause 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 standard specifies a status reporting and logging method to manage the challenges of a microservice architecture. It starts with an overview, followed by a brief overview of jobs, the log entry structure, transaction tracker, and details about how job progress and status are logged.

2 Conformance

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 clause 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.

© SMPTE 2025 – All Rights Reserved

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.

3 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 RFC 7321, *Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content, Internet Engineering Task Force*. https://datatracker.ietf.org/doc/rfc7231/

IETF RFC 7807, *Problem Details for HTTP APIs, Internet Engineering Task Force, March 2016*. https://tools.ietf.org/html/rfc7807

ISO 8601, *Date and Time Format, International Organization for Standardization, 2017*, https://www.iso.org/iso-8601-date-and-time-format.html

ISO/IEC 21778:2017, Information technology — *The JSON data interchange syntax, International Organization for Standardization*. https://www.iso.org/standard/71616.html

4 Terms and Definitions

For the purposes of this document, the following terms and definitions apply.

4.1
event-driven architecture

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

Note 1 to entry: When given events are detected, ECA (Event-Condition-Action) rules are triggered, and appropriate actions taken or functions implemented.

4.2
Function as a Service
FaaS

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

© SMPTE 2025 – All Rights Reserved

4.3
JavaScript Object Notation
JSON

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

4.4
job

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

4.5
log

record of statuses collected during job execution

4.6
rules-based architecture

deterministic architecture in which work is performed according to a set of rules

4.7
status reporting

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

4.8
tracker

token passed between the microservices involved in the completion of a transaction, allowing their correlation

4.9
tracker identifier
tracker id

unique identifier associated with a tracker

4.10
transaction

set of related tasks treated as a single action, forming a logical unit of work

© SMPTE 2025 – All Rights Reserved

5 Overview (Informative)

The concepts covered 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 by submitting it to the Job Processor service.

Subsequently, this Job Processor service invokes the appropriate service(s), passing along that tracker identifier. The service(s) then execute the desired operations. All of these operations are considered a transaction and while processing, it generates log entries which also contain the tracker identifier. The log entries can be viewed and analyzed in the log repository of your choice. The tracker identifier provides the ability to distinguish one transaction from another, as multiple transactions could occur simultaneously. Examples shown in Figure 1 include AI (Artificial Intelligence) and Transcode services, but these could be any services.

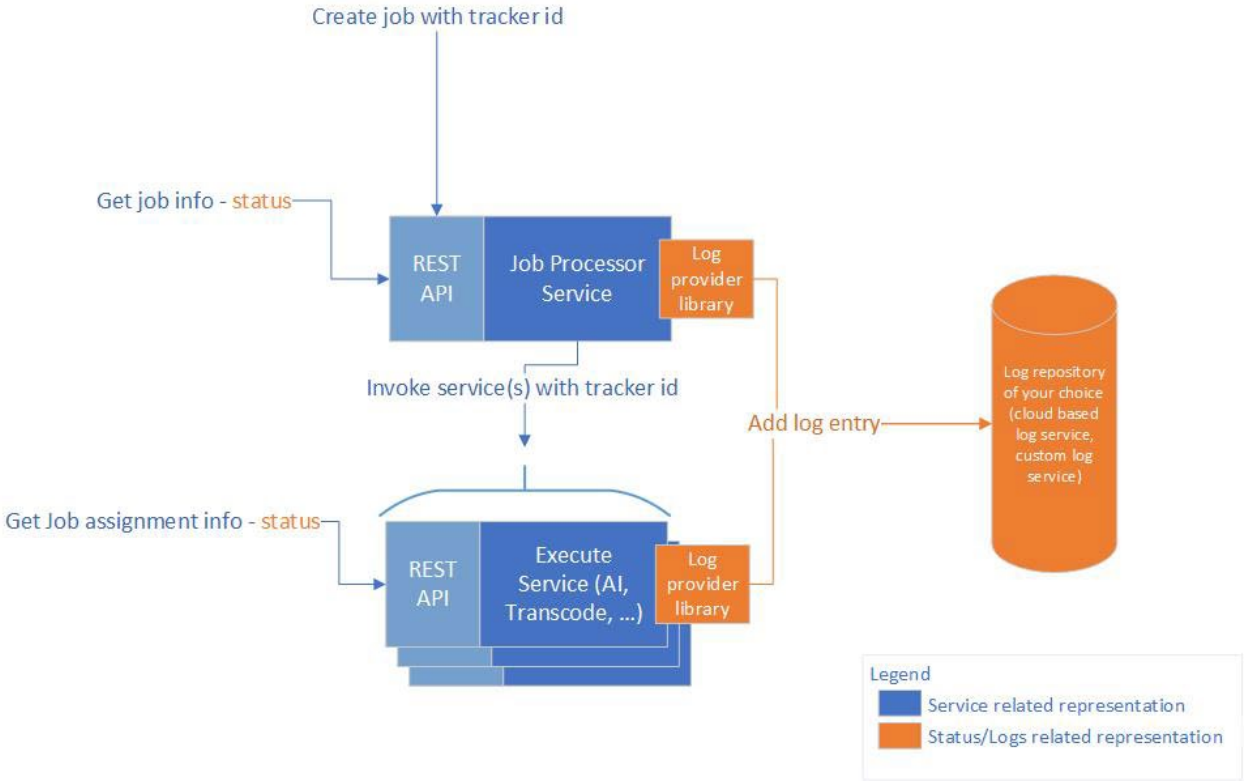

Figure 1 — Dataflow representation of jobs and status reporting/logging

© SMPTE 2025 – All Rights Reserved

6 Jobs

6.1 Job Overview

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.

A job shall have the properties (JSON fields) as specified in Table 1.

Table 1 — Job Properties

Property	Type	Required for Logging?	Clause in this doc
@type	string	Yes	6.2
id	string	Yes	6.3
jobProfile	string	Yes	6.4
jobInput	object	Yes	6.5
status	string	Yes	6.6
error	Problem Detail (RFC 7807)	Yes	6.7
jobOutput	object	Yes	6.8
progress	non-negative integer	No	6.9
parentId	string	No	6.10
timeout	non-negative integer	No	6.11
deadline	string	No	6.12
tracker	McmaTracker	No	6.13
notificationEndpoint	string	No	6.14

6.2 @type

Indicates the general type of job and used for routing the request to the correct service.

EXAMPLES
@type examples include ‘CaptureJob’, ‘TransferJob’, ‘TransformJob’, ‘QAJob’, ‘AIJob’, etc.

6.3 id

URL where the job is stored and retrievable via an HTTP GET request.

6.4 jobProfile

URL where the job profile is stored and retrievable via an 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.

© SMPTE 2025 – All Rights Reserved

6.5 jobInput

Collection of input parameters, which the executing service processes.

6.6 status

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

6.7 error

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

This object conforms to the RFC 7807 specification where applicable RFC 7807 defines a "problem detail" as a way to carry machine-readable details of errors in an 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 may have various members, including possible extensions. The only member not used is “status”, which describes “The HTTP status code (RFC 7231, Section 6) generated by the origin server for this occurrence of the problem,” and which is not applicable in this (serverless) context.

6.8 jobOutput

Property which contains the output produced by the executing service.

6.9 progress

Property which may be populated by the executing service to indicate progress percentage, expressed as a non-negative integer with a maximum value of 100.

6.10 parentId

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

6.11 timeout

Time in minutes, expressed as a non-negative integer, 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.

6.12 deadline

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

6.13 tracker

Refer to Clause 7.

6.14 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.

© SMPTE 2025 – All Rights Reserved

7 Transaction Tracker

7.1 Transaction Tracker Overview

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 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 is not necessarily unique.

There is also an optional property for custom items, as shown in Clause 9.

A Transaction Tracker shall have properties (JSON fields) as specified in Table 2.

Table 2 — Transaction Tracker Properties

Property	Type	Clause in this doc
@type	string	7.2
id	string	7.3
label	string	7.4
custom	object	7.5

7.2 @type

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

7.3 id

Unique id for this transaction.

7.4 label

Human-readable label this transaction.

7.5 custom

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

© SMPTE 2025 – All Rights Reserved

8 Tracker Example (Informative)

EXAMPLE

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

9 Custom Properties

9.1 Custom Properties Overview

The tracker may be extended if required. To do so, 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”.

EXAMPLE
Custom Property for ingest name will appear as a log entry with a name of trackerIngestName.

9.2 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.

EXAMPLE

```
{
    "@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": "2025-06-28T00:00:40.559Z",
    "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 to search log records by these new fields.

© SMPTE 2025 – All Rights Reserved

10 Logs

10.1 Log Entries Overview

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

Log entries shall have the properties as specified in Table 3.

Table 3 — Log Properties

Property	Type	Clause in this doc
type	string	10.2
level	number	10.3
source	string	10.4
requestId	string	10.5
timestamp	string	10.6
trackerId	string	10.7
trackerLabel	string	10.8
tracker	any	10.9
message	any	10.10

```
{
    "type": "DEBUG",
    "level": 500,
    "source": "job-processor-worker",
    "requestId": "50ec3f76-aa35-41d8-bb0d-d1bf627af139",
    "timestamp": "2025-06-28T00:00:40.559Z",
    "message": "Handling worker operation 'DeleteJob'...",
    "trackerId": "6fcf8dd2-a4dc-4282-8828-58631a37d41f",
    "trackerLabel": "Workflow 'test7' with file '2015_GF_ORF_00_18_09_conv.mp4'",
}
```

10.2 type

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

The following list of predefined types shall be supported. 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 a Job that is executed.

© SMPTE 2025 – All Rights Reserved

10.3 level

Indicates the log level of a log entry and shall be a positive integer, where “100” is the highest, and “400” the lowest. 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 specified in Table 4.

Table 4 — Log Levels

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

10.4 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.

10.5 requestId

Unique identifier indicating the current request execution. In a multithreaded or FaaS environment, the same function may be executed concurrently in multiple threads and therefore might be writing logs at the same moment with the same ‘source’. This property may then be used to distinguish between the individual requests. In the case that the programming model has no concept of a requestId, another identifier, e.g., threadId, can be used.

10.6 timestamp

Timestamp with millisecond precision as specified in ISO 8601.

10.7 trackerId

Unique id for this transaction. See Clause 7 for additional information about the trackerId.

10.8 trackerLabel

Human-readable label for this transaction. See Clause 7 for additional information about the trackerLabel.

10.9 tracker

Customizable properties related to the transaction tracker, which allows an arbitrary number of tracker properties, identified with the prefix “tracker”. See Clause 7 for additional information about the tracker.

10.10 message

Message that is to be logged. The message may be of any type. As such, it may be a string, but it may also be another JSON object.

© SMPTE 2025 – All Rights Reserved

11 Log Entry Example (Informative)

The following shows an example of a log entry:

EXAMPLE

```
{
    "type": "DEBUG",
    "level": 500,
    "source": "job-processor-worker",
    "requestId": "50ec3f76-aa35-41d8-bb0d-d1bf627af139",
    "timestamp": "2025-06-28T00:00:40.559Z",
    "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"
}
```

12 Job Status Logging

12.1 Job Status Logging Overview

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 starts, runs, or complies/fails, 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 as specified in Table 5.

Table 5 — Job Status Logging Properties

Property	Type	Clause in this doc
jobId	URL	12.2
jobType	string	12.3
jobProfile	URL	12.4
jobProfileName	object	12.5
jobExecution	URL	12.6
jobAssignment	URL	12.7
jobInput	object	12.8
jobStatus	object	12.9
jobError	Problem Detail as specified in IETF RFC 7807	12.10
jobActualStartDate	string	12.11
jobActualEndDate	String	12.12
jobActualDuration	non-negative integer	12.13
jobOutput	object	12.14

© SMPTE 2025 – All Rights Reserved

12.2 jobId

URL pointing to the job instance in the job processor.

12.3 jobType

Indicates the job type.

12.4 jobProfile

URL pointing to the job profile used by the job.

12.5 jobProfileName

Name of the job profile used by the job.

12.6 jobExecution

URL pointing to the jobExecution instance in the job processor.

12.7 jobAssignment

URL pointing to the jobAssignment instance in the executing service.

12.8 jobInput

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

12.9 jobStatus

Status of the job. Valid values shall be as follows: ‘New’, ‘Queued’, ‘Scheduled’, ‘Running’, ‘Completed’, ‘Failed’, or ‘Canceled’

12.10 jobError

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

12.11 jobActualStartDate

Date in ISO 8601 format when job was queued for processing.

12.12 jobActualEndDate

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

12.13 jobActualDuration

Job duration in milliseconds.

12.14 jobOutput

Collection of output results of the job that was executed.

© SMPTE 2025 – All Rights Reserved

13 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.

EXAMPLE


```
{
    "type": "JOB_END",
    "level": 400,
    "source": "job-processor-worker",
    "requestId": "4d2e0097-f311-4056-ba99-d7f6c10cd504",
    "timestamp": "2025-06-26T18:59:18.873Z",
    "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": "2025-06-26T18:59:14.043Z",
        "jobActualEndDate": "2025-06-26T18:59:16.675Z",
        "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"
}
```


14 Source Code (Informative)

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.

© SMPTE 2025 – All Rights Reserved

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.

© SMPTE 2025 – All Rights Reserved