SMPTE PUBLIC CD

IMF Registration Service API

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 Media Systems, Control and Services.

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.

The imf-registration-api was developed in response to the need for IMF processes to share and manage assets at rest rather than to deliver assets between systems during the mastering process. A group of companies came together to define the properties of a minimum viable API for processes to communicate.

The canonical definition of the API was developed using the OpenAPI Specification v3.03.The standardized version of this API uses an equivalent JSON representation of this object structure. This standard provides additional clarification on the semantics and use of the fields of the API.

NOTE: This document is intended to be used for the SMPTE public CD process while the JSON Schema Normative Reference completes its standardisation within the IETF.

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

To define an Application Programming Interface for registering, listing and querying the location of assets used in IMF applications. Security, authentication, rights and privileges to perform the API functions are out of scope for this document.

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.

ISO/IEC 21778:2017 Information technology — The JSON data interchange syntax

JSON Schema Language draft-json-schema-language-02

https://tools.ietf.org/html/draft-json-schema-language-02

RDD 45:2017 - SMPTE Registered Disclosure Doc - Interoperable Master Format — Application ProRes

RFC 3174 US Secure Hash Algorithm 1 (SHA1)

RFC 7231 Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content

RFC 7972, Entertainment Identifier Registry (EIDR) URN Namespace Definition

ST 2067-2:2020 - SMPTE Standard - Interoperable Master Format - Core Constraints

ST 2067-3:2020 - SMPTE Standard - Interoperable Master Format — Composition Playlist

ST 2067-5:2020 - SMPTE Standard - Interoperable Master Format — Essence Component

ST 2067-100:2014 - SMPTE Standard - Interoperable Master Format — Output Profile List

ST 2114:2017 - SMPTE Standard - Unique Digital Media Identifier (C4 ID).

3 Terms and Definitions

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

3.1. application

software making requests and receiving responses using the API

Note 1 to entry: Multiple applications may communicate via the API

3.2. endpoint

software receiving requests and providing responses using the API

Note 1 to entry: Multiple endpoints may be accessed by the API

3.3. ETag

software making requests and receiving responses using the API entity tag as defined in the HTTP protocol

Note 1 to entry: An example of an `ETag` is given in the `ETag` header and schema section below

3.4. idempotent

returns identical results when invoked with identical parameters regardless of the number of times invoked

4 Overview

4.1 RESTful design

The API is intended to be used at a single endpoint called `/assets`. An application should consider all endpoints with different paths to be independent. This document does not prevent some mechanism being used to define a relationship between the responders at different endpoints. Any such mechanism is out of scope of this document.

EXAMPLE: These two endpoints shall be considered to be independent systems:

1. `https://example.com/mam-01/assets`
2. `https://example.com/mam-02/assets`

The API uses a RESTful pattern where the HTTP verbs have the following meanings:

1. `GET /assets` - retrieve registration records from a provider.
2. `POST /assets` - create a new registration record with a provider.
3. `PUT /assets` - idempotent method to update a registration record with a provider.
4. `DELETE /assets` - destroy a registration record with a provider.

4.2 OpenAPI definition

The API can be represented by the OpenAPI specification and a full, testable, JSON representation is available in **element-a** of this document. An informative representation, using the more typical OpenAPI YAML syntax, is also presented as **element-b**.

This standard document has extracted schemas from the JSON in **element-a** to normatively constrain certain properties of the API. The extracted schemas definitions are presented as normative elements of the standard. In the case of conflict between the document text and the JSON Schema element, the normative text shall prevail.

NOTE: Cross references in the schema fragments refer to paths in **element-a**. Where possible, schema fragments that reference each other are presented next to each other in the document.

5 RESTful API End Points

5.1 `GET` methods

5.1.1 `GET` parameters

When querying a large set of records, there may be potentially many records. Paging of the response from a GET query shall be supported by all implementations. Paging is implemented by a requestor using the `limit` and `skip` HTTP query parameters defined below:

- `limit` shall comply with `limitSchema`. It shall be an unsigned, non-zero integer or the string `ALL`. `limit` is the maximum number of records that a responder shall return in a response. A responder

may return fewer records. If the value of `limit` is greater than the value supported by a responder, then a smaller, supported value of `limit` shall be returned in a response.

Table 1 - limitSchema

```
{
  "title": "limitSchema",
  "oneOf": [
    {
      "$ref": "#/components/schemas/limitValueSchema"
    },
    {
      "$ref": "#/components/schemas/limitAllSchema"
    }
  ]
}
{
  "title": "limitValueSchema",
  "description": "A limit can be an integer value indicating the maximum of entries to return in a query. A limit value
set greater than the system's internal max limit value will be clamped to the max limit value number.\n",
  "type": "integer",
  "default": 20,
  "example": 20
}
{
  "title": "limitAllSchema",
  "description": "A value of ALL for limit will result in the maximum number of entries allowed by the system (max limit
)\n",
  "type": "string",
  "enum": [
    "ALL"
  ]
}
```

- `skip` - shall comply with the `skipParamSchema`. It shall be an unsigned integer. It indicates the number of records that should be skipped before the first entry in the response.

Table 2 - skipParamSchema

```
{
  "title": "skipParamSchema",
  "type": "integer",
  "default": 0
}
```

NOTE 1: There is no obligation for a responder to provide any sorting mechanism in the responses.

NOTE 2: Registration records may be created, updated or removed between page requests. Handling of these race conditions is outside the scope of this document. Signaling any change in sort mechanism is outside the scope of this document.

5.1.2 `GET` responses

6 `assetInfoSchema` response Schema

All `GET` requests shall respond with a page of data that validates against the `assetInfoSchema`

Table 3 - assetInfoSchema

```
{
  "title": "Asset Info",
  "type": "object",
  "required": [
    "identifiers",
    "locations"
  ],
```

```
  "properties": {
    "file_size": {
      "type": "integer"
    },
    "file_type": {
      "description": "cc.ft.xml cc.ft.mxf cc.ft.asdcp-mpeg2-ves cc.ft.asdcp-jp2k-j2c cc.ft.asdcp-jp2k-j2c-stereo cc.ft.a
sdcp-pcm cc.ft.asdcp-text cc.ft.asdcp-aux-data cc.ft.asdcp-atmos cc.ft.cinecanvas cc.ft.png cc.ft.ttf cc.ft.otf cc.ft.cp
l cc.ft.pkl cc.ft.map cc.ft.kdm cc.ft.dcdm-text cc.ft.imf-cpl cc.ft.as-02-jp2k-j2c cc.ft.as-02-pcm cc.ft.as-02-text cc.f
t.ttml cc.ft.quicktime cc.ft.as-02-prores cc.ft.imf-opl cc.ft.imf-sidecar\n",
      "type": "string"
    },
    "identifiers": {
      "type": "array",
      "items": {
        "$ref": "#/components/schemas/identifierSchema"
      }
    },
    "locations": {
      "description": "locations is a dictionary of arrays where the array key represents a providerIds for each array of
locations.",
      "type": "object",
      "properties": {
        "providerId": {
          "description": "providerId should actually be a patternProperty matching ^.+$ however openapi 3 does not suppo
rt patternProperty expressions.",
          "type": "array",
          "items": {
            "$ref": "#/components/schemas/locationSchema"
          }
        }
      }
    }
  },
  "example": {
    "file_size": 456345,
    "file_type": "cc.ft.cpl",
    "identifiers": [
      "urn:sha1:0R5e2nhq3NjcIeaUqnDwc3t6XRo=",
      "urn:uuid:26e38a18-8e2f-11e9-9bcb-60f81dc89b92",
      "urn:x-short-digest:0R5e2nhq3NjcIeaUqnDwc3t6XRo="
    ],
    "locations": {
      "localhost": [
        "near_line/vol6/example.mxf",
        "content/pkg42/example.mxf"
      ],
      "urn:example.com:providerId2": [
        "content/pkg42/example.mxf"
      ]
    }
  }
}
```

- **`file_size`** [optional] an integer representing the number of stored bytes for this asset
- **`file_type`** [optional] a string hinting at the file type of the asset. Annex A contains and informative list of known values. If file type cannot be determined then the field should be omitted.
- **identifiers** [required] an identifier that validates against the `identifierSchema` in one of the following `urn` encoded schemes:
    1. **urn:uuid:** - identifier of the asset as would have been written in the IMF `/AssetMap/AssetList/Asset[]/Id` element of an `ASSETMAP.xml` or the IMF `/PackingList/AssetList/Asset[]/Id`element of a `PKL.xml` document that referenced that asset
    2. **urn:sha1:** - the sha-1 encoded value of an IMF asset as a hex string.

NOTE 1: The hash value written into the IMF `/PackingList/AssetList/Asset[]/Hash` element of a PKL.xml document is the base64 encoding of this id. See ST 429-8 §6.3.

3. **urn:c4id:** identifier of the asset as computed using SMPTE ST 2114.
4. **urn:eidr:** identifier of the asset, its parent or an abstraction encoded according to IETF RFC 7972.
5. **urn:x-** identifier used for experimental purposes.

Table 4 - identifierSchema

```
{
  "title": "Identifier",
  "description": "Values to identify asset. Digest string values shall be prefixed (see pattern) to discriminate between possible equivalent alternatives. Multiple identifiers may be associated with a given asset. Hash-based (sha1, c4, etc) values shall be unique to each registration entry.\n",
  "type": "string",
  "pattern": "^(urn:uuid:|urn:sha1:|urn:c4id:|urn:eidr:|urn:x-).*",
  "example": "urn:uuid:506...60c"
}
```

- **locations** is an object where each property is a child array that represents a collection of locations. The key to each child array is assigned by the endpoint and is used to identify the provider of the locations in the array.
  - **array key** known as **providerId** shall be a string without spaces
  - **array values** shall be location items. Each item in the array shall validate against the `locationSchema`.

  NOTE 2: A typical use of the providerId key value is to identify different asset management systems that have registered an identifier.

  NOTE 3: `PUT` and `POST` requests are instructed to used `localhost` as a `providerId` in later sections of this document

```
{
  "title": "locationSchema",
  "type": "string"
}
```

6.1.1 `GET /assets`

This request shall return the next page of assets.

EXAMPLE: `GET /assets?limit=20&skip=1234`

6.1.2 `GET /assets/{id}`

This request shall return a paged list of all assets that match the urn encoded identifier. The identifier shall start with a urn pattern from the `identifierSchema`.

EXAMPLE:

`GET /assets/urn:sha1:0R5e2nhq3NjcIeaUqnDwc3t6XRo=?limit=20&skip=1234`

6.1.2.1 `ETag` header and schema

An endpoint shall respond with a valid `ETag` value that validates against the `ETagSchema` in Table 5 below.

An endpoint is responsible for generating a new and unique value of the `ETag` for every update of the asset registration record.

An application and an endpoint shall treat the `ETag` value as case sensitive according to the HTTP strong validation requirements.

An application shall treat the `ETag` as an opaque identifier with no intrinsic meaning other than to identify the asset registration record with a specific endpoint using this API.

EXAMPLE: Typical headers from a GET request

```
> request
  http://localhost:3000/demo/1/assets/urn:sha1:44e4c5776622426278bd99e131be956636f571a8

> response:
  connection: keep-alive
  content-length: 343
  content-type: application/json
  date: Fri, 15 May 2020 10:40:01 GMT
  etag: 17217eb311c
```

Table 5 - locationSchema

```
{
  "title": "ETag schema",
  "type": "string",
  "example": "7a93350a310f4c1af5c7de4d6e19a9ae"
}
```

6.2 `POST` methods

6.2.1 `POST` General

The `POST` method is used to create new registrations. It is not idempotent. A lightweight duplicate record prevention mechanism may be implemented using the `If-None-Match` header.

The `POST` method creates a mapping between one or more asset identifiers, and one or more locations. At least one identifier provided must be a digest type (e.g. `urn:sha1:`, `urn:c4id:`) that uniquely identifies the asset.

An endpoint may require a specific identifier type to act as a primary id. If an identifier type is required but not found then a `422 Unprocessable entity` error shall be returned.

NOTE: Communication of facility-specific primary id requirements are outside the scope of this document. Descriptive Endpoint error messages are encouraged to allow Application users to identify the reasons for `422 Unprocessable entity` error generation.

If the submitted registration entry contains identifiers or locations that already exist for different registration entries this will result in a `409 Conflict` error.

6.2.2 `POST` parameters and headers

6.2.2.1 `If-None-Match` header

The `If-None-Match: *` header may be used to disallow re-registering an asset. Without the `If-None-Match: *` header subsequent registrations for a given asset may result in the unpredictable union of identifiers and locations.

`POST` requests shall only use the value `*` for this header.

6.2.3 POST request body

The POST request body shall be a single assetInfoSchema record as described in the GET section.

The value of provideId to act as a key for the array of locations shall be set to localhost.

6.2.4 5.2.4 POST responses

6.2.4.1 201 Asset registration created

This is the success response code.

If an asset registration already exists and the If-None-Match header is not specified, the system will match all existing hash-based identifiers before locations are added to the existing registration entry (duplicates are removed).

6.2.4.2 400 Incorrect Syntax

The request could not be fulfilled due to the incorrect syntax of the request.

6.2.4.3 409 Identifier or locations conflict

Conflict. Identifiers and/or locations already belong to a different registration entries.

6.2.4.4 412 Precondition failed

Precondition failed. If-None-Match was set and the asset registration entry already exists.

6.2.5 422 Unprocessable entity

Unprocessable entity is returned if the system requires a specific identifier type which was not found.

6.3 PUT methods

6.3.1 PUT Overview

The PUT method updates an asset registration. The method shall be idempotent and shall implement the If-Match header header mechanism to guard against race conditions.

Before an application makes a PUT request, it should make a GET request for the desired asset in order to obtain the ETag value. The returned ETag value is used in the subsequent PUT request.

6.3.2 PUT parameters

6.3.3 General

The body of a put request shall validate against the assetInfoSchema.

The value of provideId to act as a key for the array of locations shall be set to localhost.

6.3.3.1 `If-Match` header

The `If-Match` header shall be present on all `PUT` requests otherwise an endpoint shall respond with a `412` error. The value of

6.3.4 'PUT' responses

6.3.4.1 `204` No Content

Success. No Content. Asset registration entry updated.

6.3.4.2 `300` Multiple entries found

Multiple entries have been found for this Id. The body of the response contains a paged list of those entries according to the `multiChoiceResponse` schema described below.

Table 6 - multiChoiceResponse

```
{
  "title": "Multi-choice Response",
  "type": "object",
  "properties": {
    "limit": {
      "type": "integer"
    },
    "skip": {
      "type": "integer"
    },
    "total": {
      "type": "integer"
    },
    "results": {
      "type": "array",
      "items": {
        "$ref": "#/components/schemas/assetInfoSchema"
      }
    }
  },
  "example": {
    "skip": 0,
    "limit": 10,
    "total": 2,
    "results": [
      {
        "$ref": "#/components/schemas/assetInfoSchema"
      },
      {
        "$ref": "#/components/schemas/assetInfoSchema"
      }
    ]
  }
}
```

- **limit** identical in functionality to the GET response `limit` property
- **skip** identical in functionality to the GET response `skip` property
- **total** the total number of multiple entries matching this identifier
- **results** an array of items that validates against assetInfoSchema

6.3.4.3 400 Incorrect Syntax

The request could not be fulfilled due to the incorrect syntax of the request.

6.3.4.4 404 Not Found

Asset registration entry not found.

6.3.4.5 412 Precondition failed

Precondition failed. ETag did not match for update.

6.3.4.6 422 Unprocessable entity

Unprocessable entity. Returned if the system requires a specific identifier type which was not found.

6.3.4.7 428 Precondition required

Precondition required. Expecting If-Match header.

6.4 DELETE methods

6.4.1 DELETE general

The DELETE method deletes a registration entry.

NOTE: the ability to delete entries may require authentication that is outside the scope of this document.

6.4.2 DELETE /assets/{id}

This request shall delete a registration entry for an asset that matches the urn encoded identifier. The identifier pattern is described in the GET section

6.4.3 DELETE responses

6.4.3.1 204 No Content

No Content. Asset registration entry was successfully removed.

6.4.3.2 300 Multiple Entries Found

Multiple entries have been found for this Id. The body of the response contains a paged list of those entries according to the multiChoiceResponse schema described in the PUT 300 error above.

6.4.3.3 404 Not Found

Asset registration entry not found.

7 Annex A `file_type` values in the `assetInfoSchema` (informative)

Implementers are encouraged to use the following identifiers to help systems identify the likely content available at a target destination. The list contains data types that are not defined by internationally recognized standards and so this table is informative.

Where a standard exists, the identifier can be used if the file is known to be of that type, otherwise a more generic identifier is recommended.

identifier	Definition (with reference)
`cc.ft.xml`	XML document
`cc.ft.mxf`	MXF wrapped file of unknown contents
`cc.ft.asdcp-mpeg2-ves`	MPEG2 Video elementary stream
`cc.ft.asdcp-jp2k-j2c`	JPEG2000 j2c codestream
`cc.ft.asdcp-jp2k-j2c-stereo`	JPEG2000 j2c stereoscopic codestream
`cc.ft.asdcp-pcm`	PCM audio
`cc.ft.asdcp-text`	SMPTE ST 429-5 D-Cinema captions
`cc.ft.asdcp-aux-data`	CTA-708 caption stream
`cc.ft.asdcp-atmos`	Atmos audio codestream
`cc.ft.cinecanvas`	Cinecanvas captions
`cc.ft.png`	PNG graphic
`cc.ft.ttf`	Truetype Font
`cc.ft.otf`	OpenType Font
`cc.ft.cpl`	SMPTE ST 429-xxx DCP Composition Play List
`cc.ft.pkl`	SMPTE ST 429-8 Packing List
`cc.ft.map`	SMPTE ST 429-9 Asset Map
`cc.ft.kdm`	SMPTE ST 430-xxx Key Definition Message
`cc.ft.dcdm-text`	SMPTE ST 428-7 D-Cinema Distribution Master captions
`cc.ft.imf-cpl`	SMPTE ST 2067-3 IMF Composition Play List
`cc.ft.as-02-jp2k-j2c`	SMPTE ST 2067-5 IMF JPEG2000 MXF Track File
`cc.ft.as-02-pcm`	SMPTE ST 2067-2 IMF Audio MXF Track File
`cc.ft.as-02-text`	SMPTE ST 2067-2 IMF TTML MXF Track File
`cc.ft.ttml`	TTML XML document
`cc.ft.quicktime`	Apple QuickTime file of unknown codec type

cc.ft.as-02-prores	SMPTE RDD 45 IMF ProRes MXF Track File
cc.ft.imf-opl	SMPTE ST 2067-100 IMF OPL XML document
cc.ft.imf-sidecar	SMPTE ST 2067-9 IMF Sidecar Map XML document

Bibliography

c4id website and reference code https://cccc.io

Open API Specification, https://www.openapis.org/

YAML, YAML Ain't Markup Language, https://yaml.org