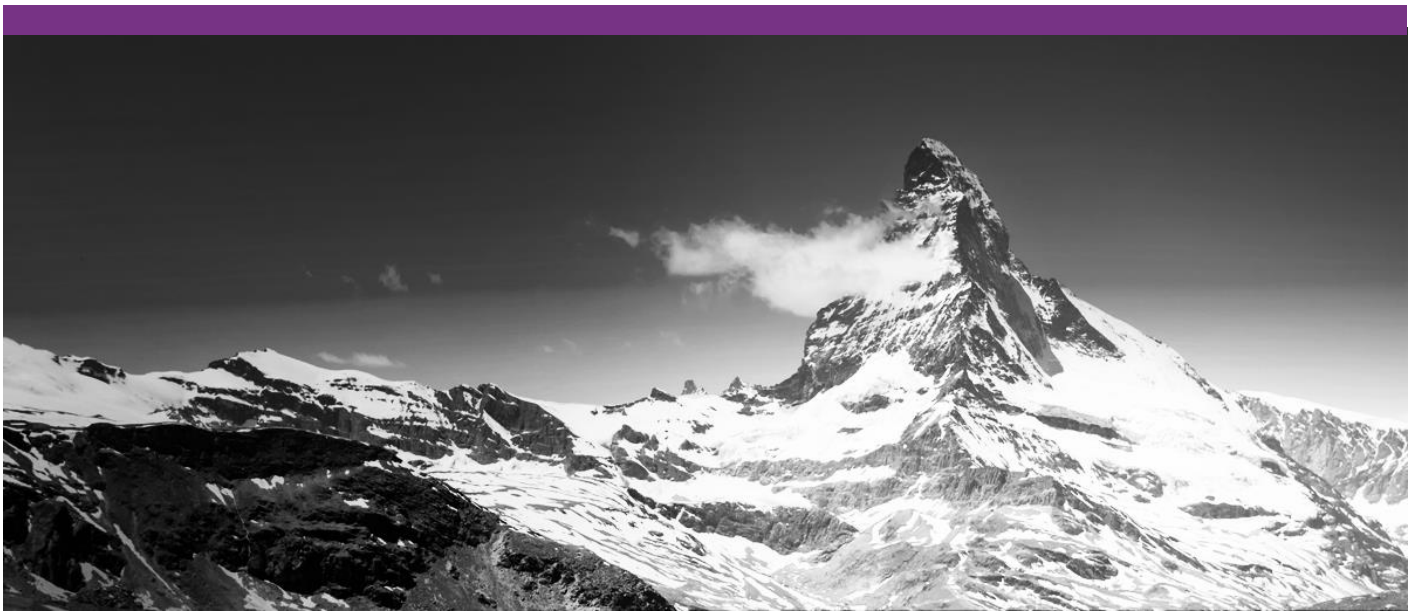
SOLENIX



Training API Standard

Training Calendar

Prepared by:

Solenix Deutschland GmbH

Project Ref.: SLXDE/LEARNTECH/2019

Doc. Ref.: SLXDE-LEARNTECH-2019-TN-01

Title:	Training API Standard
Volume:	Training Calendar
Customer:	EUMETSAT
Customer Reference:	EUMETSAT
Project Reference:	SLXDE/LEARNTECH/2019
Document Reference:	SLXDE-LEARNTECH-2019-TN-01
Date:	15/04/2021
Version:	01.01
Document Responsible:	Martin Tykal, Bruno Agatão
Approved:	
	Martin Tykal, Team Manager

Company:	Solenix Deutschland GmbH Spreestrasse 3 64295 Darmstadt Germany	Phone: Fax: E-Mail: Internet:	+49 6151 870 91 0 +49 6151 870 91 66 info@solenix.de www.solenix.de

Document Log

Revision	Date	Responsible	Comment
01.00	17/03/2021	Martin Tykal	Document Creation
01.01	15/04/2021	Martin Tykal	Addition of filtering & offsetting, hashes and additional field recommendations

Distribution List

Name	Organisation

Table of Content

Document Log **3**

Distribution List **3**

1 Introduction **6**

1.1 Purpose & Scope 6

1.2 Document Structure 6

1.3 Applicable Documents 6

1.4 Reference Documents 6

1.5 Acronyms and Abbreviations 6

2 Training Calendar APIs **8**

2.1 Landscape 8

2.2 Structure 9

2.3 Challenges 9

3 The Training API Standard **11**

3.1 API Structure 11

3.2 Events 13

3.3 Metadata 16

3.4 Documentation 17

3.5 Evolutions 17

4 Shared Endpoints **19**

4.1 Retrieval of Events 19

4.2 Retrieval of Metadata 20

4.3 Documentation 22

5 Check Lists **23**

5.1 API Structure 23

5.2 Events 23

5.3 Metadata 24

5.4 Documentation 24

1 Introduction

1.1 Purpose & Scope

This document describes a standard for APIs that aim to provide training events to support and foster the harmonisation between several Training Calendar implementations.

1.2 Document Structure

Section 2 provides an overview of the current Training Calendar landscape and the role of APIs therein. Further discussion is taken on the current challenges this standard attempts to overcome.

In section 3, the API standard is further described. Attention is given to its premise, structure, the inclusion of event and metadata endpoints, as well as recommendations on documentation and the description of versioning.

Section 4 provides a concise overview of the required endpoints.

The document concludes with section 5, which provides some helpful check lists for developers to ensure compatibility with this standard.

1.3 Applicable Documents

Ref.	Document Title	Reference
AD-01		

1.4 Reference Documents

Ref.	Document Title	Reference
RD-01		

1.5 Acronyms and Abbreviations

Acronym	Description
API	Application Programming Interface
CRUD	Create, Read, Update, Delete
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation

Acronym	Description
VLab	WMO-CGMS Virtual Laboratory on Satellite Meteorology Training and Education

2 Training Calendar APIs

Worldwide meteorological satellite operators work together to facilitate the usage of data and products of earth observation satellites. These operators are organised within the WMO-CGMS Virtual Laboratory on Satellite Meteorology Training and Education (VLab). Within this organisation, different training calendars are offered to advertise upcoming trainings. The training calendars of different organisations within VLab offer RESTful APIs that enable public users to fetch and display training events across different calendar implementations. As such, it is important to foster the harmonisation of these APIs to enable displaying events organised across VLab in all associated training calendars.

2.1 Landscape

Several organisations within VLab offer training calendars to advertise training events across the network, among others EUMETSAT[1], Global Campus[2], VLab[3], EUMETCAL[4], CEOS[5], EO-College[6] and NASA. Typically, APIs provide access for several clients to one data store. However, in the case of the training community, there are several collaborating event providers with their own data stores, APIs and web frontends, as shown in Figure 1. While the data stores and APIs are typically organisation specific (in green), the collaboration mainly focuses on advertising each other's training events across several calendar implementations (in purple).

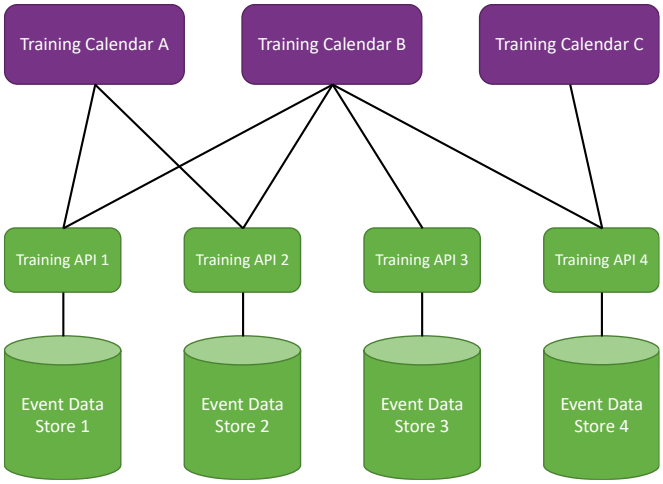


Figure 1: Training Calendar Landscape

RESTful APIs, such as the ones relevant in this context, are by definition stateless and uniform. Thus, their duplication should be avoided to ensure maintainability and compatibility with clients. However, due to the fact that so many different organisations collaborate together to organise and advertise events, a solution

[1] https://trainingevents.eumetsat.int/trui/
[2] https://learningevents.wmo.int
[3] https://www.wmo-sat.info/vlab/calendar-of-events/
[4] https://www.eumetcal.eu/en/app#/catalog/search/courses
[5] https://training.ceos.org
[6] https://eo-college.org/events/

needs to be found that enables organisations to freely build their web services with RESTful APIs, while ensuring that parts of the APIs follow some standard that ensures compatibility across the entire training community.

2.2 Structure

The structure of each training calendar is different in terms of adopted technologies and approaches. The common ground is the usage of a RESTful API. These APIs are interfaces to the underlying data store, made available through HTTP, to offer users the possibility of data retrieval or means to execute operations on the data, using popular internet formats such as XML or JSON as the transportation language. The focus of these APIs is on resources and how to provide access to these resources. Typically, the training APIs map resources to database tables. As in the database, the main resources are usually the events and the users, but also additional resources are provided, such as course applications, WMO regions, countries, languages, event types, expertise fields, groups, profiles, email alerts, etc.

In REST, every resource is uniquely identified by an endpoint expressed as a URI (Uniform Resource Identifier), and the web service uses a directory hierarchy to address its resources. For instance, the event with id=1 is referred with the following URI: https://<host>/trapi/resources/public/events/1

The operation to execute on a certain resource or collection of resources is determined by the HTTP method or verb, see Table 1. With them, CRUD operations support is provided for the resources in the underlying data store.

Table 1: CRUD Overview

Verb	Description
GET	Fetch a resource
POST	Create a new resource
PUT	Update/replace a resource
DELETE	Delete a resource
PATCH	Update/modify a resource
OPTIONS	List supported operations on a resource

Typically, training calendar APIs consist of public and private endpoints. As the name suggests, public endpoints can be queried by all internet users, while private endpoints require additional authentication before returning any data.

2.3 Challenges

Defining an API standard shares many challenges with defining any kind of standard, e.g. hitting the right spot between constraints and benefits. The challenges of this API standard mainly revolve around finding a common ground between existing implementations to maximise compatibility and ensuring maintainability over a long period.

In the past years, several training event APIs were developed. Some, but not all, are already RESTful. Independently, REST does only denote a software architecture style with certain constraints and best-practices. As such, it does not specify the exact implementation. Consequently, RESTful APIs can differ considerably from one other. The goal of this standard is to keep changes to existing APIs to a minimum while ensuring full compatibility across all implemented APIs.

The compatibility is further challenged by the fact that no common definition of events, their content and lifecycle exist. Events can contain more than just a title and time, e.g. participants, trainers, organisations, topics, etc. Furthermore, events can be in different states, e.g. draft, final or deleted. Not all states are used equally across the implementations. Deleting an event, for example, can signify that it is removed from the database, or, alternatively, that its status is changed to a different value, such as “cancelled” or “deleted”. These different approaches complicate a unified usage of APIs, as specific queries and filters need to be used to receive the required information.

Maintaining an API is a common challenge. Requirements might change over time and API endpoints might be added, altered, deprecated or removed over time. Even small changes on an API endpoint can turn out to be breaking changes for some users. Luckily, versioning schemas have emerged in the usage of APIs, which will be reused for this standard.

3 The Training API Standard

The trade-off in every standard is between constraining the design and usage to ensure compliance and providing sufficient flexibility for unique features. In this case, it is the intention to maintain utmost flexibility while describing common elements that should be shared across all APIs implementing this standard.

3.1 API Structure

The provision of both public and private endpoints is a typical need for meteorological APIs. Public endpoints provide basic event information accessible to everyone. Private endpoints are used to administrate events, manage trainings and more. It is **not** the intention to standardise these public and private endpoints. It should be up to the event providers to decide and implement these endpoints according to their specific requirements. This standard only considers shared endpoints that are required for disseminating event information.

Shared endpoints have the advantage that training providers have greater control over which events and metadata are shared with other calendars. The reduced complexity of the shared endpoints allows developers to query and visualise events based solely on knowledge of the base path of the APIs to be integrated in a calendar.

JSON Format

While REST APIs can communicate in arbitrary formats, using XML or JSON are the most common approaches. APIs compatible to this standard have to accept JSON as request payload and respond with JSON on the shared endpoints. JSON is a current standard for transferring data. Almost every networked technology can use it: JavaScript has built-in methods to encode and decode JSON either through the Fetch API or other HTTP clients. Server-side technologies have libraries that can decode JSON without doing much work.

To ensure that clients interpret the responses correctly, the `Content-Type` field in the response header should be set to `application/json` after a request is made.

Read-Only

Shared endpoints should be read-only to ensure that no unauthorised access is made to the underlying data. Consequently, the shared endpoints of this standard support **only** `GET` requests.

URL Structure

The URL structure shall follow the best practises and the following standardised approach. The best practises are:

- Nouns shall be used in the endpoint paths. There may not be any verbs used in the path. Verbs are only used in the HTTP request method (only GET in this standard).

  A valid example: `<base_path>/shared/v1/events/`

A non-valid example: `<base_path>/shared/getEvents/`

- Typically, collections of events are requested. As such, plural nouns should be used for collections, e.g. `/events`.

The general URL structure for shared endpoints needs to be kept simple. As such, the schema shall be as follows:

```
<base_path>/shared/v1/<entity>
```

where `<base_path>` is the base URL of the API and `<entity>` denotes the entity to be queried.

This base URL can be simple, e.g. https://trainingevents.eumetsat.int/ or more complex, e.g. https://trainingevents.eumetsat.int/trapi/resources/.

A list of endpoints that need to be provided by compatible APIs is given in section 4.

Filtering

Training event databases can become very large. As such, measures shall be taken to prevent that all data be required to be returned at once. Consequently, filters are necessary.

Filtering shall be provided to all endpoints providing data, see section 4.

The filter structure shall be `<entity>?field1=operator:value&field2=operator:value&…`

A valid example for filtering events could be:

```
https://trainingevents.eumetsat.int/shared/v1/events?title=Training&location=Darmstadt
```

More advanced filtering requires not only to set certain values, but also to query based on comparison operators, such equal, greater than, less than or equal to, etc. The list of comparison operators that shall be supported can be found in section 4.

Limit & Offset

Another measure to avoid that all data is returned at once from the database are limits and offsets. Limiting shall be implemented in such a way that the latest events, sorted by descending date, are returned.

Limits constrain the number of results that are returned. The value for limit must be an integer and must be greater than 0.

Offsets specify the first position to return from the results of the query. The default is 0, which starts the response with the first entry. The value for offset must be an integer and must be greater than or equal to 0.

Limits and offsets shall be provided to all endpoints providing data, see section 4.

The limit structure shall be `<entity>?limit=value`

The offset structure shall be `<entity>?offset=value`

Limits and offsets can be used together to implement pagination. A valid example for limiting events could be:

```
https://trainingevents.eumetsat.int/shared/v1/events?limit=10&offset=40
```

In this example, 10 events in places 40-49 would be returned.

Please note that the recommended approach to limit the number of returned events is via filtering. Especially filtering for a relevant timespan, e.g. all upcoming events, can drastically reduce the number of returned events, while ensuring that all relevant events are in fact returned. However, choosing a low value for limit can cause relevant events to not be returned, while a high value can result in too many non-relevant events being queried.

Error-Handling

Errors that could occur on the shared endpoints shall be handled graciously and return standard HTTP response codes that indicate which kind of error occurred. Ideally, specific error messages should be provided additionally to the error code. Common error HTTP status codes:

- 404 Not Found – A resource was not found.
- 500 Internal server error – This is a generic "catch-all" server error. Should not be thrown explicitly.
- 502 Bad Gateway – Indicates an invalid response from an upstream server.
- 503 Service Unavailable – Indicates a server-side error.

3.2 Events

This section provides an overview of how events should be structured to be compatible with the standard.

Minimal Set of Fields

In many cases, it is not necessary to display all available information on training events. Especially if only an overview of events is provided, it is sufficient to provide the most important information of title, date and time, a location and description. To keep the complexity of providing such overviews as low as possible, all events have to provide these fields in a standardised fashion.

Training events have to provide a minimal set of defined fields, which are:

Table 2: Minimal Set of Event Information

Name	JSON-Field	Description
Title	"title"	The title of the event as a plain string.
Start Time	"startDate"	Beginning of the event in UTC, expressed as in ISO 8601 format, such as *YYYY-MM-DDThh:mm:ssZ*.
Location	"location"	Location of the event as a plain string.
Description	"description"	Description of the event as a plain string.

Name	JSON-Field	Description
Hash	“hash”	An MD5 hash of above parameters, which can be used to programmatically ensure uniqueness of events. The hash is to be calculated as *MD5(title + startDate + location + description)*

The minimal valid response of an event query in JSON can look as follows:

```
[{
    "title": "Training Event",
    "startDate": "2021-04-15T11:12:00Z",
    "location": "Darmstadt",
    "description": "This is the description of a sample event",
    "hash": "c4aa3268acab54a8b2c960223dc2f59a"
}]
```

In this example, the MD5 hash is calculated based on the String

“Training Event1632700800000DarmstadtThis is the description of a sample event”.

As all training events provide this elemental information in a standardised fashion, all calendar implementations can display them without further information required.

Plain Text

All text in events, e.g. titles and descriptions, should be plain text without styling information such as HTML tags or Markdown. Training calendars might interpret non-text components differently and there is a risk that event titles or descriptions become illegible.

Additional Fields

Of course, events do often contain more information, such as training organisations, trainees, topics, etc. This information can be added to the response as needed. There are no constraints on the number or nature of additional fields. To enable developers to use these additional fields programmatically, additional metadata endpoints are defined, which are described in the next section. Any JSON response that contains the fields described in Table 2 is considered valid.

Another example of a valid response of an event query in JSON with additional fields:

```
[{
    "title": "Training Event",
    "startDate": "2021-04-15T11:12:00Z",
    "location": "Darmstadt",
    "description": "This is the description of a sample event",
    "hash": "c4aa3268acab54a8b2c960223dc2f59a",
    "trainingPartner": [],
    "language": "en",
    "qualification": "Training: Education and Training Providers "
}]
```


Recommendations

While additional fields can be used as required, in practice it is beneficial to follow a standard for specific fields.

Table 3: Optional Field Recommendations

Field	Recommendation	Example
Country	ISO 3166-1 alpha-3	GBR, DEU, ESP, …
Language	ISO 639-1	en, de, es, …
WMO Region	WMO Regions[7]	Region I: Africa, Region II: Asia, …
Competences	Compendium of WMO Competency Frameworks[8]	Analyse and Continually Monitor the Evolving Meteorological and Hydrological Situation

Flat Event Objects

Especially for additional fields, it can be common to use nested objects. Nested objects are typically required to transport additional information, which is used in specific calendar implementations. The usage of nested objects increases the complexity, which hampers the overall compatibility.

Consequently, event objects provided by the shared endpoints should be flat. The transported information in a field shall be limited to the minimum required information for proper display.

For example, a nested language object

```
[{
  "language": {
    "id": "3",
    "iso639": "en",
    "value": "English"
  }
}]
```

shall be reduced to the bare minimum required information, in this case the language, and, as such, should follow the above recommendation:

```
 "language": "en",
```

Pre-Filtered Events

Events can often have different statuses, such as “Draft”, “Tentative”, “Final” or “Deleted”. It is important to only advertise events with valid information, and that are sure to take place, on the shared endpoints.

No deleted events or events with draft information shall be made accessible via the shared endpoints. The events returned on the shared events endpoints must be pre-filtered to contain only valid events.

[7] https://public.wmo.int/en/about-us/members
[8] https://library.wmo.int/index.php?lvl=notice_display&id=21607#.YHbY0OgzY2w

3.3 Metadata

Metadata is crucial to describe events beyond the minimal set of information described in section 3.1. Events typically consist of multiple fields with a mixture of content types. Some fields contain free text, while others are populated with pre-defined options or timestamps. In order to properly present events beyond their minimal information and/or provide useful filters in the calendar implementation, having this metadata available is crucial.

Required Fields

Each API has to provide a list of event metadata. This list contains sufficient information to utilise the metadata of events programmatically. Specifically, the list of event metadata has to contain the following information for each metadata property:

Table 4: Minimal Set of Event Information

Name	JSON-Field	Description
Name	“name”	Name of the metadata field as a string, as it should be displayed in a calendar.
Field Name	“fieldname”	The name of the metadata field in the event response.
Values	“values”	Array containing pre-defined values for the metadata field. If no pre-defined values exist, this field can be left empty.

A valid response of querying the metadata list endpoint in JSON can look as follows:

```
[{
     "name": "Language",
     "url": "https://trainingevents.eumetsat.int/trapi/resources/public/languages”,
     "fieldname": "language",
     "values": ["en", "fr", "zh", "ru"]
}]
```

There are several benefits to this approach:

- The “name” field specifies how the metadata field should be displayed in a calendar application. This information can be used to create labels for additional event information.
- The “fieldname” allows for the mapping of fields in the event response to metadata. This can be used for example to assign the correct event values to labels mentioned in point 1.
- The “values” field contains pre-defined values for the metadata. Pre-defined values are typically options for the respective event property. For example, for languages the values field would contain all languages trainings are offered in. Or for a field “Organisation” it could contain all training organisations. The content of this field can be used for example to populate filter dropdown menus.

Flat Metadata Objects

Metadata can become arbitrarily complex when describing events. It is not unusual to use nested objects to define metadata. For example, a language object returned by the respective API endpoint could contain an id, iso639 representation (“en”) and a value (“English”). While this complexity can be warranted for internal purposes, it has to be avoided for events to be shared with other calendars.

No nested objects shall be used in the values field. Instead, the recommended values defined in section 3.2 shall be used where applicable.

3.4 Documentation

The standard as described above allows for the retrieval and displaying of events without requiring additional information besides the `base_path` of an API. However, the goal is not to exclude the private and public endpoints from providing more information than the shared endpoints. It is possible that calendar implementations emerge which offer more functionality than pure presentation, e.g. creating and editing events when provided with proper authentication.

To facilitate a deeper usage of training APIs by their clients, the usage of the REST API documentation tool Swagger UI[9] is strongly recommended. Swagger UI can scan the code of the API on each launch and auto-generate documentation. The basic documentation can be amended by annotations in the code to provide more information. It is up to the API developer to configure which endpoints are documented. Private endpoints, for example, could be excluded.

Using such an auto-generated REST API documentation requires little effort, but can benefit the community greatly. It substantially lowers the hurdle of creating good clients for an API.

The documentation created by Swagger UI shall be available on the endpoint

```
<base_path>/shared/v1/documentation
```

3.5 Evolutions

Clients of an API rely on the promise that no breaking changes are introduced in the API. In practice this can be difficult to achieve in the lifecycle of an API. New requirements are added over time, while others are descoped. Consequently, the API can change with the requirements, introducing breaking changes for clients relying on an older version.

Consequently, new versions of the API have to be introduced for any changes that may break clients. For this standard, a version is added to all endpoints, e.g. `<base_path>/shared/v1/events`. This way, old endpoints can be phased out gradually. Endpoints of an older version can stay active for clients that are not changed, while the newer version endpoints can serve new features for those willing to upgrade.

[9] https://swagger.io/tools/swagger-ui/

It is recognised that the version of the shared endpoints of this standard may not coincide with the version number of the underlying API. The API of a training provider can have a different version than the version of this standard. For this reason, the version number of the shared endpoints defined by this standard are after the `/shared` path. Any version number of the underlying API can be part of the `base_path`, e.g. https://some.trainingprovider.com/api/v3/shared/v1/events.

The current version of the shared endpoints is *v1*.

4 Shared Endpoints

4.1 Retrieval of Events

Retrieval of all the events

HTTP Request Type	GET
URL	http://<base URL>/shared/v1/events
Description	Querying this endpoint returns the list of all events.

Retrieval of events with filter

HTTP Request Type	GET
URL	http://<base URL>/shared/v1/events?*field*='*value*'
Description	Adding filters as URL parameters allows to reduce the returned set of events, e.g. *http://<base URL>/shared/v1/events?title=Training&location=Darmstadt* The following comparison operators shall be supported: **eq** The default operator if no operator is specified. Returns results where a field is *equal* to the value. *http://<base URL>/shared/v1/events?title=Trainingevent* *http://<base URL>/shared/v1/events?title=eq:Trainingevent* **not** Returns results where a field is not equal to the value. *http://<base URL>/shared/v1/events?title=not:Trainingevent* **gt** Returns results where a field is *greater than* the value. *http://<base URL>/shared/v1/events?startDate=gt:2021-04-15T11:12:00Z* **gte** Returns results where a field is *greater than or equal to* the value.

	http://<base URL>/shared/v1/events?startDate=gte:2021-04-15T11:12:00Z **lt** Returns results where a field is *less than* the value. *http://<base URL>/shared/v1/events?startDate=lt:2021-04-15T11:12:00Z* **lte** Returns results where a field is *less than or equal to* the value. *http://<base URL>/shared/v1/events?startDate=lte:2021-04-15T11:12:00Z*

Retrieval of events with limit

HTTP Request Type	GET
URL	http://<base URL>/shared/v1/events?*limit*='*value*'
Description	Adding a limit allows to reduce the returned set of events, e.g. *http://<base URL>/shared/v1/events?limit=500*

Retrieval of events with offset

HTTP Request Type	GET
URL	http://<base URL>/shared/v1/events?*offset*='*value*'
Description	Adding an offset allows the implementation of pagination, e.g. *http://<base URL>/shared/v1/events?offset=40&limit=10*

4.2 Retrieval of Metadata

Retrieval of all metadata

HTTP Request Type	GET
URL	http://<base URL>/shared/v1/metadata
Description	Querying this endpoint returns the list of all metadata.

Retrieval of metadata with filter

HTTP Request Type	GET
URL	http://<base URL>/shared/v1/metadata?*field*='*value*'
Description	Adding filters as URL parameters allows to reduce the returned set of metadata, e.g. *http://<base URL>/shared/v1/metadata?fieldname=country* The following comparison operators shall be supported: **eq** The default operator if no operator is specified. Returns results where a field is *equal* to the value. *http://<base URL>/shared/v1/metadata?fieldname=country* *http://<base URL>/shared/v1/metadata?fieldname=eq:country* **not** Returns results where a field is not equal to the value. *http://<base URL>/shared/v1/metadata?fieldname=not:country* **gt** Returns results where a field is *greater than* the value. *http://<base URL>/shared/v1/metadata?value=gt:5* **gte** Returns results where a field is *greater than or equal to* the value. *http://<base URL>/shared/v1/metadata?value=gte:5* **lt** Returns results where a field is *less than* the value. *http://<base URL>/shared/v1/metadata?value=lt:5* **lte** Returns results where a field is *less than or equal to* the value. *http://<base URL>/shared/v1/metadata?value=lte:5*

Retrieval of metadata with filter

HTTP Request Type	GET
URL	http://<base URL>/shared/v1/metadata?*limit*='*value*'
Description	Adding filters as URL parameters allows to reduce the returned set of metadata, e.g. *http://<base URL>/shared/v1/metadata?limit=500*

4.3 Documentation

Accessing the Swagger UI Documentation

HTTP Request Type	GET
URL	http://<base URL>/shared/v1/documentation
Description	A webpage displaying the auto-generated documentation of the API.

5 Check Lists

This section contains check lists that can help ensure an API has been implemented compatibly with the standard.

5.1 API Structure

Item	Description	Implemented?
JSON Format	Shared endpoints must accept and respond with JSON.	
Read-Only	Shared endpoints must only accept GET requests.	
URL Structure	The URL structure of shared endpoints must follow the schema `<base_path>/shared/v1/<entity>`.	
Filtering	Shared endpoints must support filtering according to the schema `<base_path>/shared/v1/<entity>?field1=value&field2=value&…`	
Limit	Shared endpoints must support setting a limit according to the schema `<base_path>/shared/v1/<entity>?limit=value`	
Offset	Shared endpoints must support setting an offset according to the schema `<base_path>/shared/v1/<entity>?offset=value`	
Error Handling	Shared endpoints must return HTTP error codes if errors occur.	

5.2 Events

Item	Description	Implemented?
Minimal Set of Fields	Events must contain the fields “title”, “startDate”, “location”, “description” and “hash”.	
Hash	Event hashes are in MD5 shall be calculated as *MD5(title + startDate + location + description)*	
Plain Text	All text fields must be returned in plain text.	
Additional Fields	Are there any additional fields that should be made available? If yes, describe via metadata (see section 5.3)	
Recommendation	Do additional fields follow the recommendation where applicable (see Table 3)?	
Flat Event Objects	Event objects must have a flat structure to avoid unforeseen complexity.	
Pre-Filtered Events	Events returned must be pre-filtered to ensure that only valid events are advertised.	

5.3 Metadata

Item	Description	Implemented?
Required Fields	Metadata must contain the fields “name”, “url”, “fieldname” and “values”.	
Flat Metadata Objects	Metadata objects must have a flat structure to avoid unforeseen complexity.	

5.4 Documentation

Item	Description	Implemented?
Swagger UI	Setting up Swagger UI to auto-generate the API documentation.	
Availability	Making the documentation available on the respective URL.	