AlpineBits®

DestinationData 2022-04

AlpineBits® is an interface specification for exchanging data in the tourism sector, specially tailored for alpine tourism.

AlpineBits® DestinationData specifies a REST API for exchanging destination data based on the AlpineBits® DestinationData Ontology. The API is build upon JSON:API v1.0 designed to support the client-server communication model, with a focus on system-to-system communication.

Table of Contents

About the AlpineBits Alliance

The "AlpineBits Alliance" is a group of SME operating in the touristic sector working together to innovate and open the data exchange in the alpine tourism, and therefore to optimize the online presence, sales and marketing efforts of the hotels, other accommodations and destinations in the alpine territory and also worldwide.

AlpineBits Alliance
Via Bolzano 63/A
39057 Frangarto / Appiano s.s.d.v. (BZ) - ITALY
VAT Reg No: IT02797280217
https://www.alpinebits.org
info@alpinebits.org

AlpineBits Alliance Members

ADDITIVE OHG - https://www.additive.eu
Altea Software Srl - http://www.altea.it
aries.creative KG - http://www.ariescreative.com
ASA OHG - http://www.asaon.com
Brandnamic GmbH - http://www.brandnamic.com
Consisto Arl - https://www.consisto.it
Destination Srl - http://www.destinationsrl.it
GardenaNet snc - http://www.gardena.net
Giggle GmbH - https://giggle.tips
HGV - http://www.hgv.it
IDM Südtirol - Alto Adige - http://www.idm-suedtirol.com
Internet Consulting GmbH - https://www.internet-consulting.it
Internet Service GmbH - http://www.internetservice.it
LTS - http://www.lts.it
Marketing Factory GmbH - http://www.marketingfactory.it
NOI Techpark - https://noi.bz.it
Outdooractive GmbH - https://corporate.outdooractive.com
PCS Hotelsoftware GmbH - http://www.pcs-phoenix.com
Peer GmbH - http://www.peer.biz
Schneemenschen GmbH - http://www.schneemenschen.de
SiMedia GmbH - http://www.simedia.com
Südtiroler Bauernbund - www.sbb.it
XLbit snc - http://www.xlbit.com
Yanovis - http://www.yanovis.com

Authors of this document:

AlpineBits® DestinationData repository contributors - https://gitlab.com/alpinebits/destinationdata/standard-specification

This document has been produced in collaboration with the Conceptual and Cognitive Modelling Research Group (CORE) of the Free University of Bozen-Bolzano.

Document Change Log

Important note: make sure to have the latest version of this document! The latest version is available at https://www.alpinebits.org.

Protocol version	Doc. release date	Description
2022-04	2022-04-30	This release contains the following improvements: • Added resource creation specification • Added resource update specification • Added resource deletion specification • Added support to virtual and hybrid events • Updated licensing information on `mediaObjects` resources with the replacement of the `copyrightOwner` relationship with `licenseHolder` and introduction of the `author` attribute • Improvement of the definition of the `dataProvider` field on the `meta` of any resource • Improvement of the definition of the `editions` and `series` relationships on `events` and `eventSeries` resources • Improvement of the definition of the `connections` relationships on `lifts`, `mountainAreas`, `skiSlopes`, and `snowparks` resources • Improvement of the definitions of the ontology • Updated list of HTTP status codes • Re-structuring of chapter `API Architecture` into chapters `API Architecture`, `Requests and Responses`, and `Routes` for improved organization of contents
2021-04	2021-03-24	This release contains the following improvements: • Added filtering specification • Added searching specification • Added sorting specification • Added random sorting specification • Added field selection specification • New resource type: `categories` • Transformed `categories` attribute into a relationship • New resource type: `features` • Transformed `features` attribute into a relationship • Changed the versioning format of server routes • Renamed the `trails` resource type to `skiSlopes` • Improved guidelines on how to handle bad requests • Added recommendation for server to provide OpenAPI-based documentations
2020-04	2020-04-30	First release published.

1. Introduction

This document describes a standard for exchanging data in the tourism domain, called **AlpineBits® DestinationData**, which is built upon:

- an OntoUML ontology that describes the conceptualization and scope of the standard;
- the REST architectural style;
- the JSON:API v1.0 specification for REST APIs that exchange JSON data through HTTP messages;
- HTTPS and basic authentication protocols for secure communication;
- the JSON Schema standard, draft 7 for message validation;
- the GeoJSON standard for JSON geospatial modelling; and
- Schema.org, used as an inspiration for designing resource types.
- the OpenAPI specification, recommended as an additional standard for documentation of server implementations.

The **AlpineBits® DestinationData** standard was designed to support data exchange between systems acting as CLIENTS and SERVER, where CLIENTS consume the data provided by SERVERS.

The **AlpineBits® DestinationData** standard defines:

- resource types
- server routes
- support to GET requests
- request and response headers and parameters
- additional request features (e.g., pagination and hypermedia controls)

The current version of the standard supports exchanging data about events, event series, event venues, mountain areas, lifts, ski slopes, snowparks, agents, and media objects.

1.1. Conventions and Definitions

The keywords “MUST”, “MUST NOT”, “REQUIRED”, “SHALL", “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in RFC 2119.

Additionally, the following terminology is consistently used throughout this document:

- `CLIENT`: a system that consumes data provided by a SERVER via HTTP requests
- `SERVER`: a system that stores and exposes data to CLIENTS in conformance with the **AlpineBits® DestinationData** API.
- `Resource Type`: a set of attributes and relationships used to characterize resources
- `Resource`: an object with a persistent identifier that conforms to a resource type
- `Endpoint`: the trailing part of a URL that provides access to HTTP requests on an API

2. The AlpineBits® DestinationData Ontology

The **AlpineBits® DestinationData Ontology** formalizes the worldview shared by AlpineBits members over concepts of the tourism domain. It supports semantic interoperability between members through a technology-agnostic conceptual model.

The ontology documents the members' agreement at the conceptual level and drives the development of the **AlpineBits® DestinationData**, while the API specification documents their agreement at the technological level is defined in the subsequent chapters.

A user of the **AlpineBits® DestinationData** may use this ontology as a reference to interpret the real-world semantics of the data she will consume from a SERVER, thus avoiding interoperability issues.

2.1. Modeling Language: OntoUML

The **AlpineBits® DestinationData Ontology** is designed using the OntoUML, an ontologically well-founded extension of the UML Class Diagram that can be used to represent a domain of interest from a computationally independent perspective.

OntoUML consists of a set of stereotypes applicable to classes, associations, and attributes, with precisely defined formal semantics derived from the Unified Foundational Ontology (UFO), an axiomatic formal theory based on theories from analytic metaphysics, philosophical logic, cognitive psychology, and linguistics.

An important meta-property of an OntoUML class stereotype is dubbed **rigidity**.

- A *rigid class* statically classifies its instances, i.e., an instance of a rigid class must instantiate it throughout its whole existence. Examples include `Person`, `Animal`, `Organization`, `Contract`, and `Marriage`.
- An *anti-rigid class* dynamically classifies its instances, i.e., an instance of an anti-rigid class at a given point in time may cease to be so later on. Examples include `Teenager`, `Adult`, `Student`, and `Spouse`.

Another fundamental meta-property for classes is **sortality**.

- a *sortal class* is one whose instances share a common identity principle, where the sortal class must either provide this principle or inherit it from a superclass. Examples of classes that provide identity principles include `Person`, `House`, and `Car`, and those that inherit such principles include `Student`, `Man`, `Adult`, `Town House`, and `Sports Car`.
- a *non-sortal class* is one whose set of instances includes entities complying with different identity principles. Examples include `Agent`, which classifies instances of sortal classes `Person` and `Organization`, and `Physical Object`, which classifies instances of sortal classes `Car` and `House` (among others).

The OntoUML class stereotypes used in the **AlpineBits® DestinationData Ontology** are:

- «`kind`»: a rigid sortal (identity provider) class that classifies object-like entities. Examples of typical kinds include `Person`, `Organization`, `Car`, and `House`.
- «`relator`»: a rigid sortal (identity provider) class that classifies relational entities, also known as relationships. Examples include `Marriage`, which relates two instances of `Spouse`; `Contract`, which relates instances of `Party` in the context of formal agreements; and `Enrollment`, which relates an instance of `Student` to an instance of `Educational Institution`.
- «`quality`»: a rigid sortal (identity provider) class that classifies entities that represent aspects of other entities and are measurable in some value space (i.e., conceptual space). A quality may be used to compare individuals, on the basis of the value it takes in a certain quality space (for example, a mass in the kilogram scale, or a position within the RGB spectrum). Examples include `Weight` (as in the weight of a person), `Name` (as in the name of an organization), `Color` (as in the color of a car), and `Duration` (as in the duration of a concert).

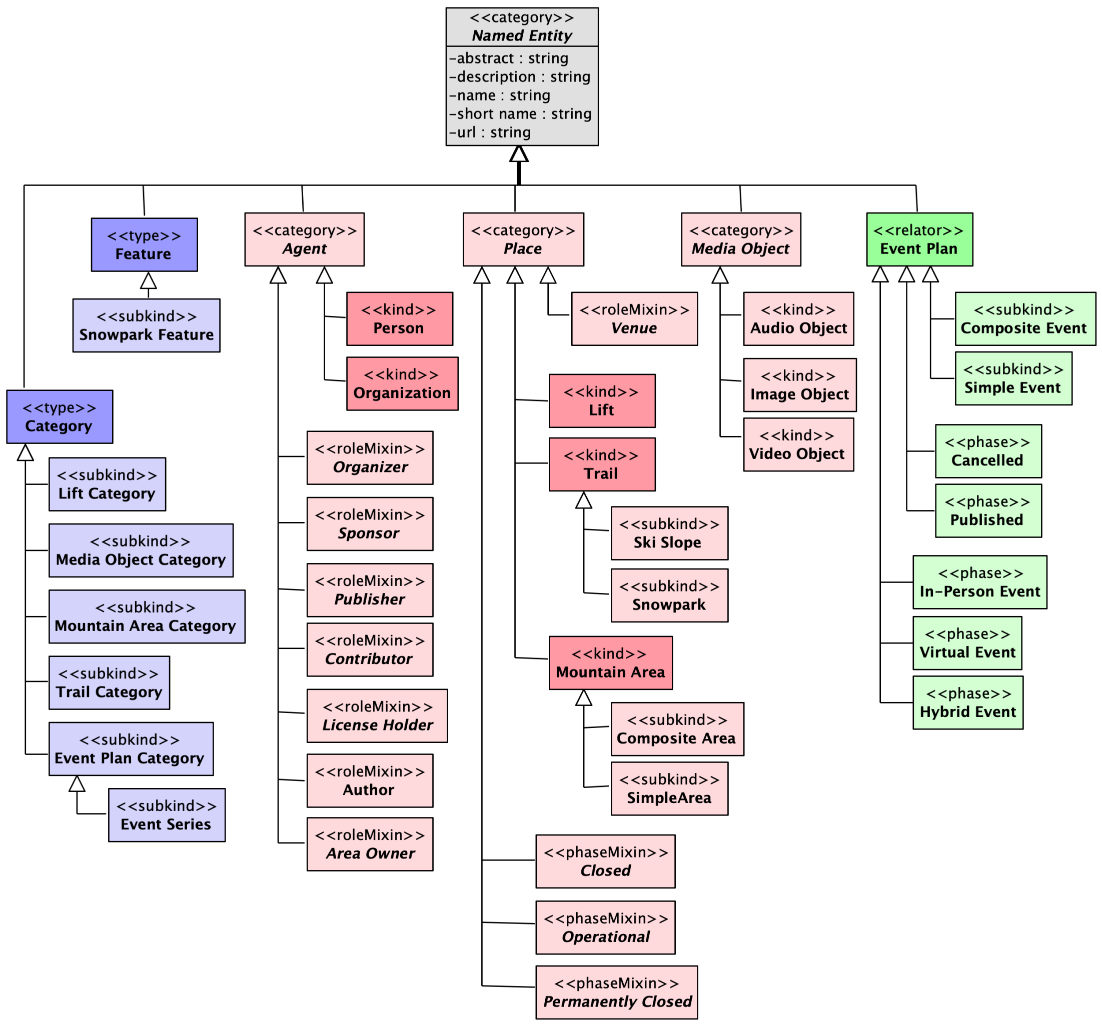
- «`subkind`»: a rigid sortal class that inherits its identity from another sortal. Examples of subkinds include `For-Profit Organization`, `Fiat 500`, and `Civil Marriage`.
- «`role`»: an anti-rigid sortal class whose instantiation depends on a relational condition. Examples include `Student`, `Artist`, and `Legally Recognized Marriage`.
- «`phase`»: an anti-rigid sortal class whose instantiation depends on a change in an intrinsic property. For example, `Child` may be a subclass of `Person` whose instances are people up to 12 years old.
- «`category`»: a rigid non-sortal class that classifies entities of different sorts. Examples include `Agent`, which classifies instances of both `Person` and `Organization`; and `Animal`, which classifies instances of different species of animals, including `Felis Catus` (domestic cat), `Canis Lupus Familiaris` (dog), and `Panthera Leo` (lion).
- «`roleMixin`»: an anti-rigid non-sortal class that classifies roles playable by individuals of different sorts. Examples include `Organizer`, as the agent that organizes some event, and `Author`, as the agent that holds the right to some intellectual property.
- «`type`»: a rigid class that classifies entities that have instances themselves (i.e., other classes). Examples include `Car Model`, whose instances may include `Fiat 500` and `Tesla Model S`; and `Event Type`, whose instances may include `Musical Event` and `Sports Event`.
- «`datatype`»: a class that classifies values contained in a well-defined conceptual space, e.g., integer and real numbers (in their respective sets), `Mass in Kilograms`, and `RGB Color`.
- «`enumeration`»: a class that classifies values within a discrete finite conceptual space. Examples include `Day of Week`, whose possible instances are the 7 days of the week; and `Driver License Category`, whose possible instances are `A`, `B`, `C`, `D`, and `E`.

The OntoUML association stereotypes used in the **AlpineBits® DestinationData Ontology** are:

- «`mediation`»: an existential dependence relation that connects relators to the entities they bind. For example, instances of the relator `Marriage` mediate instances of the class `Spouse`; instances of the relator `Contract` mediate instances of the class `Contract Party`.
- «`characterization`»: an existential dependence relation that connects an aspect (e.g., a quality) to the entity it characterizes. For example, instances of the class `Color` characterize instances of the class `Physical Object`.
- «`material`»: a relation that connects entities based on something dependent on these entities. For example, instances of the association "married with" connect instances of the class `Spouse` that have a marriage relationship dependent on them.
- «`componentOf`»: a part-whole relation that connects objects with their functional components. Examples include the composition relation between an instance of `Car` and the instance of `Engine` installed on it; the relation between an instance of `Human Body` and an instance of `Heart`.
- «`historicalDependence`» a relation that binds entities because of an event that happened in the past. For example, an instance of `Place` may be related through a historical dependence to an instance of `Image Object` that is a representation of that place at a point in time.
- «`instantiation`»: a relation between two classes representing that instances of one may be classified by instances of the other. For example, the relation between `Car` and `Car Model`, which represents that every car is an instance of a car model.

2.2. Ontology Description

The figure below depicts the taxonomy of named entity defined by the **AlpineBits® DestinationData Ontology** designed to represent the individuals and types present in the ontology’s domain:



All classes of entities characterized by description information in the **AlpineBits® DestinationData Ontology** specialize `Named Entity`. `Named Entity` has the following properties: `abstract`, a short description of the individual; `description`, a description of the individual more thorough than an abstract; `name`, the name of the individual; `short name`, the short version of the name of the individual (e.g., an abbreviation); `url`, a link that leads to additional information regarding the individual.

The additional classes of individuals and types in the figure above are described in the following sections.

2.2.1. Categories and Features

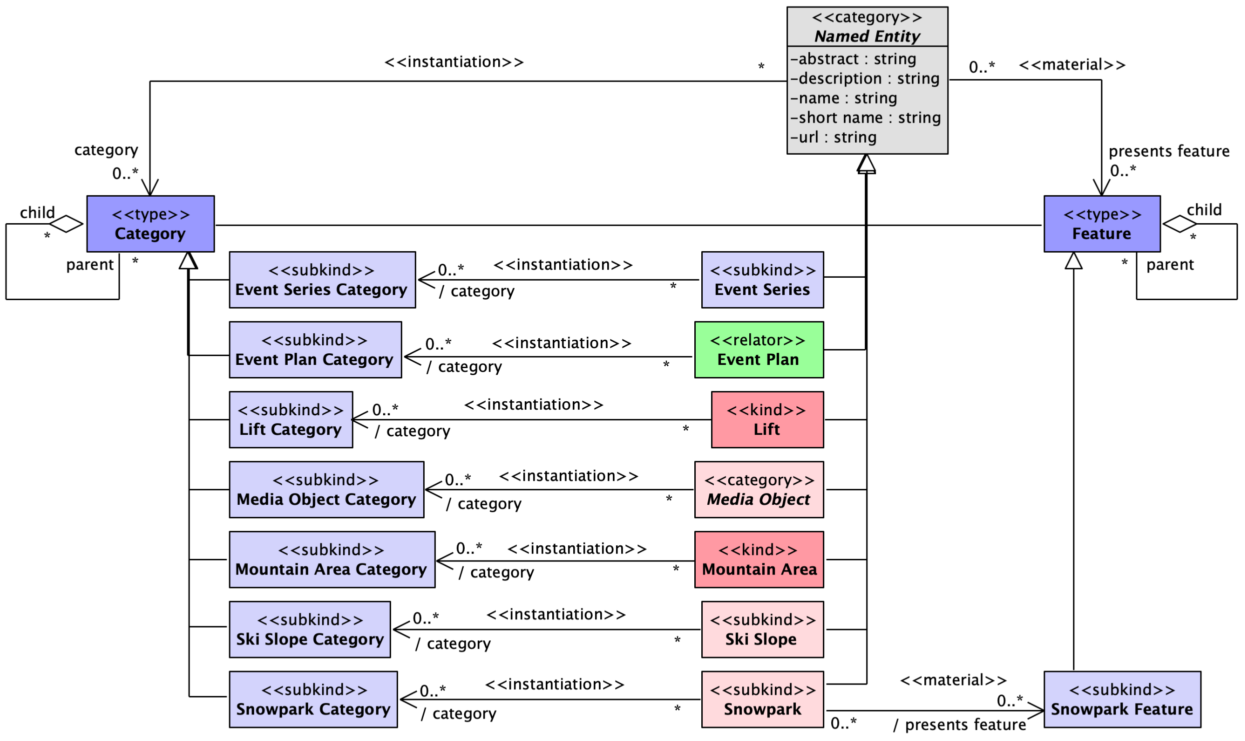
A `Category` is a type that classifies entities in the tourism domain. Examples of categories include "music event", which classifies the South Tyrol Jazz Festival 2018, and "public space", which classifies the Piazza Walther Von der Vogelweide.

A `Category` may classify one or more `Named Entities` of different types. For instance, "music event" classifies only instances of event plans, while "public space" classifies instances of venue and trail.

A `Feature` is an attraction or anything that can be present in a `Named Entity` in the tourism domain. Examples of features include, "auditorium" as in venues that present some available auditorium, "food and drinks" as in events that present food selling stations, or "parking space" for places that present a dedicated parking area.

A `Snowpark Feature` is a snowpark-specific feature. Examples of snowpark features include "rail" ands "ramp", these present in snowparks that enable the practice of special maneuvers in winter sports.

Categories and features can be organized into hierarchies where being an instance of a child category implies being an instance of the parent category and presenting a child feature implies presenting a parent feature. For example, an instance of the event plan category "conference" is also an instance of its parent category "business event". Moreover, a venue that presents the feature "olympic size swimming pool" also presents the its parent feature "swimming pool".



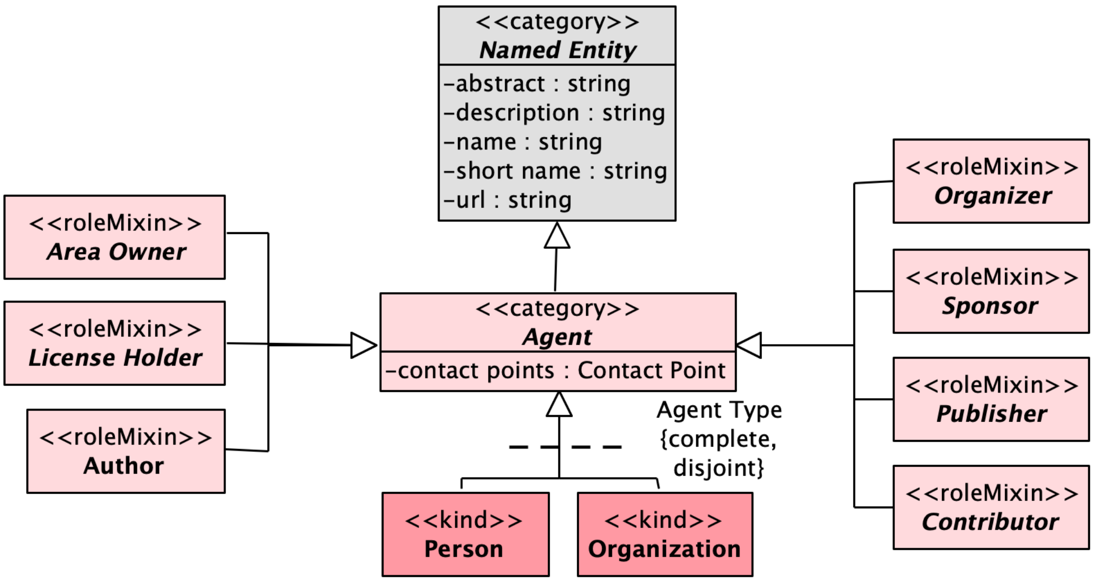
2.2.2. Agent

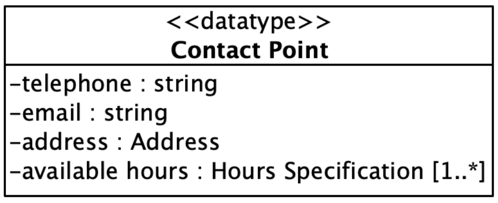
An individual who bears mental attitudes and is capable of performing actions and perceiving events.

An `Agent` is either a `Person`, such as Albert Einstein, Marie Curie, Lionel Messi, and Serena Williams, or an `Organization`, such as Apple, Facebook, the AlpineBits Alliance, and the Free University of Bozen-Bolzano.

An `Agent` can play several roles within the touristic domain, namely the `Organizer`, `Sponsor`, `Publisher`, or `Contributor` of an event, the `License Holder` or the `Author` of a media object, and the `Area Owner` of a mountain area.

A fragment of the ontology depicting `Agent` and its properties is presented below:





In addition to the attributes inherited from `Named Entity`, `Agents` have a property named `contact points`, which identifies a list of `Contact Points` one can use to get in touch with an `Agent`.

A `Contact Point` informs *how* to contact an `Agent`, namely its `email` address, `telephone` number, and physical `address`, but also *when* to do it by means of the `available hours` property.

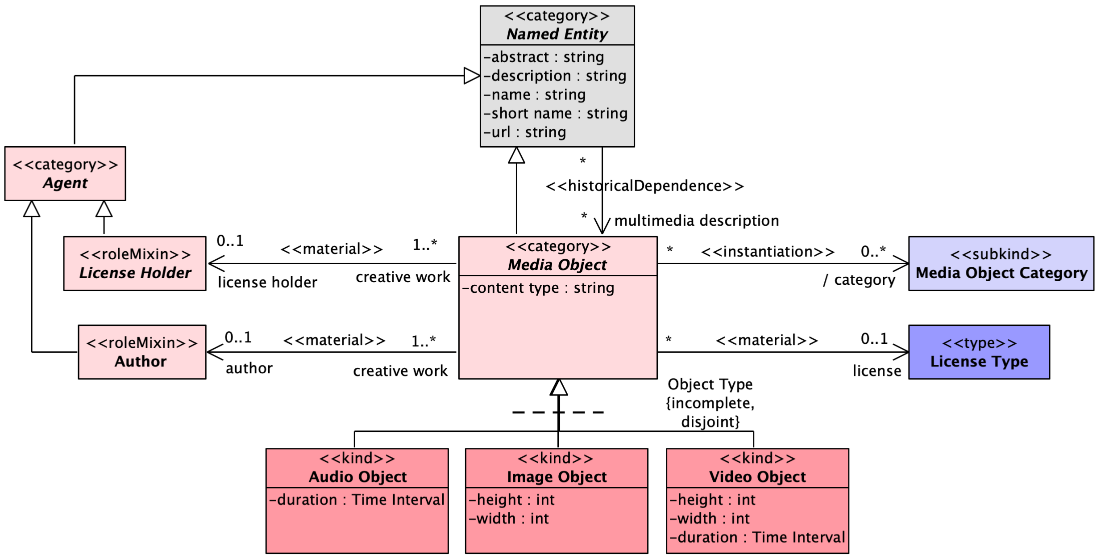
`Contact Point` information is relevant for tourists who want to contact the owner of a mountain area to know about its slopes conditions, or for customers who want to contact an event's organizer about ticket prices.

2.2.3. Media Object

An object that materializes creative works into a digital format to enable processing and sharing.

Three disjoint types of `Media Objects` are identified in the ontology, namely `Audio Object`, e.g. an audio file containing a recording of a song, `Image Object`, e.g. an image file depicting a lift, and `Video Object`, e.g. a video file containing a recording of a musical performance.

A fragment of the ontology depicting `Media Object`, its properties, and subtypes is shown below:



In addition to the attributes inherited from `Named Entity`, `Media Object` is characterized by `content type`, which refers to the Media Types (formerly known as MIME Types) defined by the Internet Assigned Numbers Authority (IANA).

A `Media Object`, or more precisely, the creative work embedded in it, can have its rights owned an by agent, termed the `License Holder`. Apart from the license holder, a creative work can also have an `Author` who may not hold the licensing right over its creation. Additionally, it may be available for reuse by others according to a `License Type`, such as the Creative Commons 1.0 Universal.

A `Media Object` often depicts a `Named Entity`, such as events, event venues, lifts, ski slopes, and snowparks. This depiction is captured in the ontology by means of the historical dependence between `Named Entity` and `Media Object`.

2.2.4. Place

An individual that has a fixed physical location and can be localized within a Global Positioning System (GPS) (adapted from Schema.org). Examples of places include a town square, a stadium, a ski slope, a gas station, and a park.

A `Place` may go through two phases: `In Operation` and `Permanently Closed`. The first classifies places that are functioning, regardless if they are currently open or not. The second classifies those that have terminated its operations.

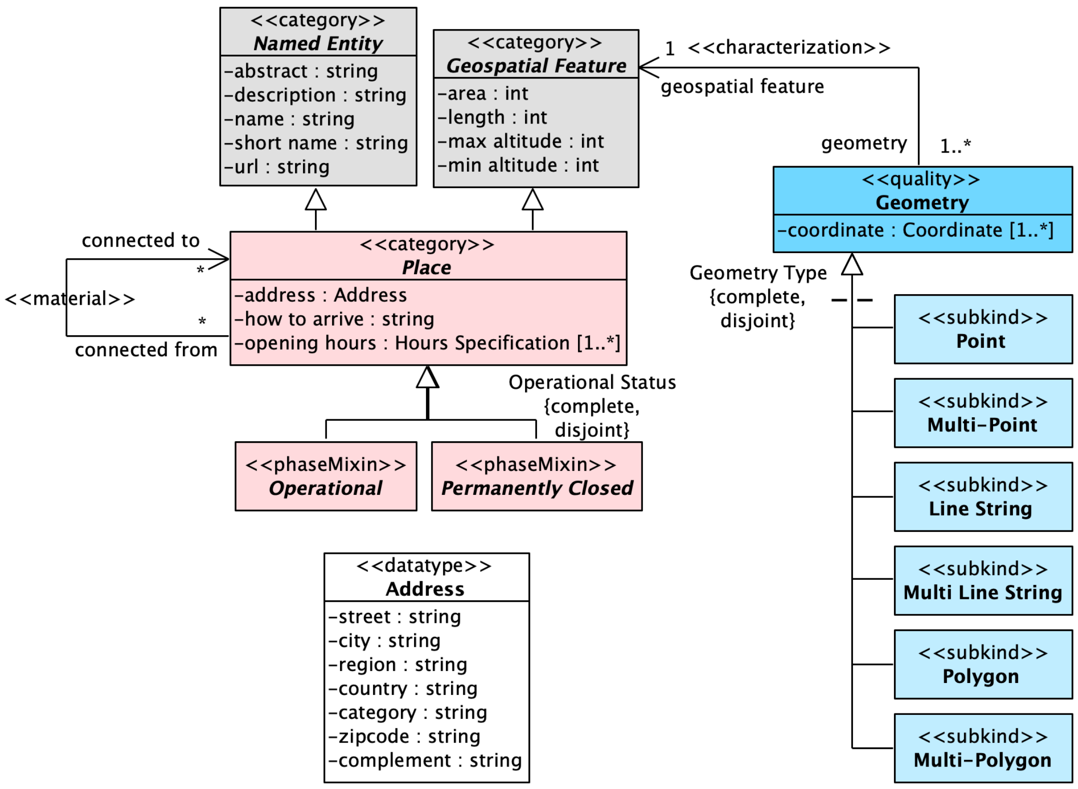
`Place` specializes `Named Entity` and thus, inherits all of its attributes.

`Place` is additionally characterized by the attributes `address`, `how to arrive`, and `openingHours`: `address` represents the physical address of a place; `how to arrive` is a textual description on how to arrive at a place; and `opening hours` identifies when a place is (or should be) open.

A `Place` may be physically connected to other places and give access to them. For instance, the Falzeben Gondola at Merano 2000 gives access to the Falzeben I and Wallpach slopes. In the ontology, this relation is captured by the `connections` self-type association defined for `Place`.

Notice, however, that the `connection` relation is non-symmetric in cases where two places are only connected in one direction. For example, A `Lift` may give access to a `Snowpark`, but the `Snowpark` (due to the descent, for instance) may not give access back to the `Lift`.

A fragment of the ontology centered around the `Place` concept, its properties, and subtypes is shown below:



The GPS information that can be associated to a `Place` is represented by means of the `Geometry` quality, which is sub-classified into the following subtypes according to its geometrical shape: `Point`, `Multi Point`, `Line String`, `Multi Line String`, `Polygon`, and `Multi Polygon`. Instances of `Geometry` must instantiate exactly one of the aforementioned subtypes.

The `coordinates` attribute in the `Geometry` class is the list of points that compose the geometry.

2.2.5. Event, Event Series and Venue

An `Event Plan` is a plan established by one or more `Organizers` aiming some `Target Audience` (adapted from the Core Public Event Vocabulary). `Event Plans` are planned to be held at some particular date and time.

Examples include the Südtirol Jazz Festival 2018, the South Tyrol Free Software Conference organized in 2019, the Bolzano Christmas Market of 2019, and a Serie A match between Juventus and Napoli.

In addition to the attributes inherited from `Named Entity`, an `Event Plan` has the attributes `start date`, `end date`, `in-person capacity`, `online capacity`, `participation url`, `registration url`, and `recorded`: a `start date` which describes when the event is planned to start; an `end date` which describes when the event is planned to end; `in-person capacity` which describes the event's capacity for in-person attendance; `online capacity` which describes the event's capacity for virtual attendance; `participation url` which is the URL for virtual attendance in the event; `registration url` which is the URL for registration to the event; and `recorded` which describes whether the event is planned to be recorded.

`Event Plans` can be `In-Person`, `Virtual`, or `Hybrid`, depending on whether they participants are planed to join the event in-person, virtually, or by either of these options.

`In-Person Events` and `Hybrid Events` happen at one or more places, which are dubbed its `Venues`. For instance, the venue of Bolzano Christmas Market in 2019 was Piazza Walther.

`Virtual Events` and `Hybrid Events` are supported by one or more `Streaming Platforms`. For instance, the South Tyrol Free Software Conference organized in 2021 was streamed to Element and YouTube.

An `Event Plan` may involve several `Agents`, i.e. persons or organizations, in different ways. An

`Organizer` plans the event and is legally responsible for it; the `Publisher` provides data about it; a `Sponsor` supports its organization, usually by making financial contributions; and a `Contributor` actively participates in the event, such as a presenter, a singer or an expositor.

`Events` may be composed of smaller events. For instance, the Südtirol Jazz Festival 2019 was composed of over 30 individual concerts. This type of event is classified as a `Composite Event` in the ontology, while those without sub-events are deemed `Simple Events`.

An `Event Plan` is in one of two phases, namely `Published` and `Cancelled`. The former refers to those whose plans are or were valid, while the latter refers to those that have been canceled.

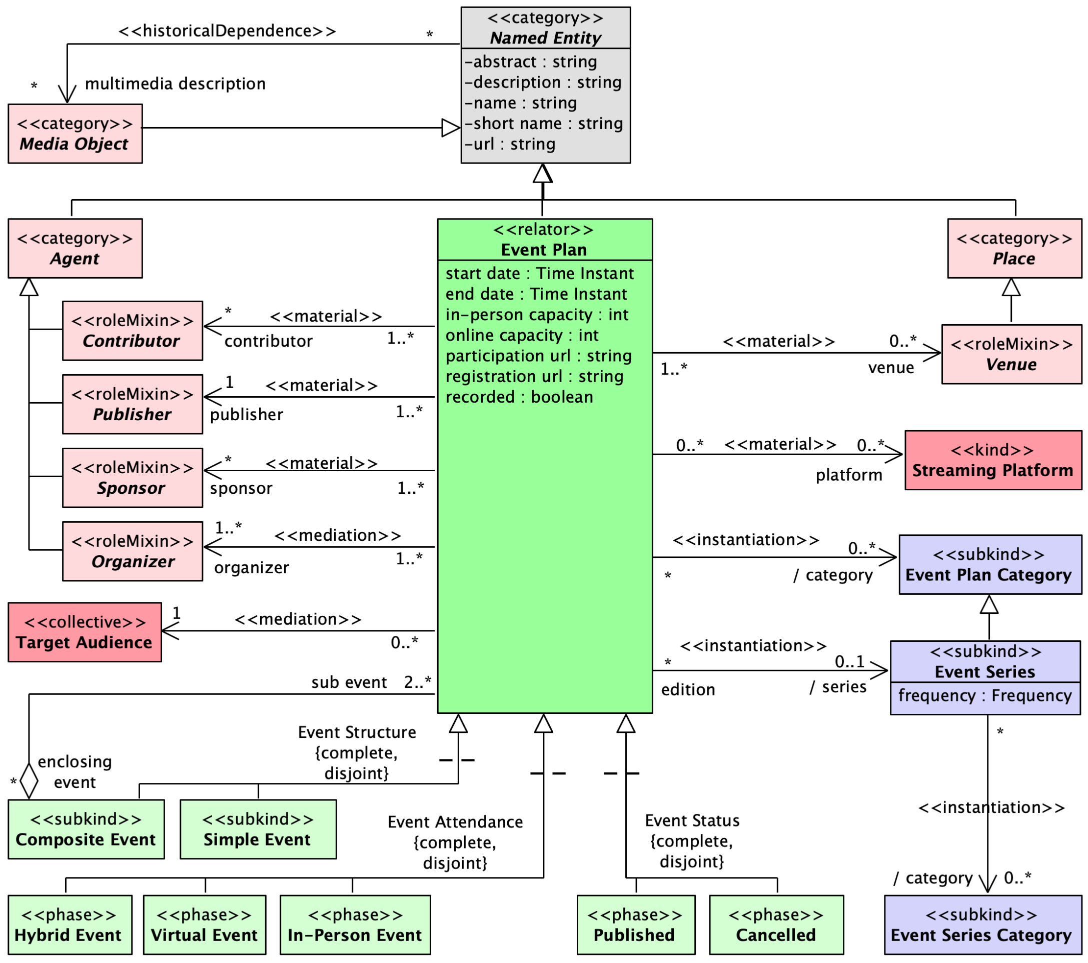
Many `Event Plan Categories` can classify `Event Plans` that share common characteristics they identify, such as "music event" or "sports event".

Each `Event Plan` may of instantiate a special type of `Event Plan Category` called `Event Series`. An `Event Series` is a kind of "template" for recurrent `Event Plans` that are referred to as its `editions`. Examples include the Olympics, which is organized every 4 years, and the Bolzano Christmas Market, which is organized every year, and the Food Truck Weekend, which is organized sporadically in Trento.

It is not possible for an `Event Plan` and one of its parts to be `editions` of the same `Event Series`. However, an `Event Plan` and its parts may be `editions` of distinct `Event Series`. For example, the "South Tyrol Jazz Festival 2021" may be an `edition` of the "South Tyrol Jazz Festival" series, while the presentation of "András Dés Rangers on 04/07/2021", its `sub-event`, may be an `edition` of the "András Dés Rangers European Tour 2021".

The frequency in which an `Event Series` is organized is captured in the ontology by the homonymous attribute.

A fragment of the ontology focused on the concepts of `Event Plan`, `Event Series`, and `Venue` is shown below:

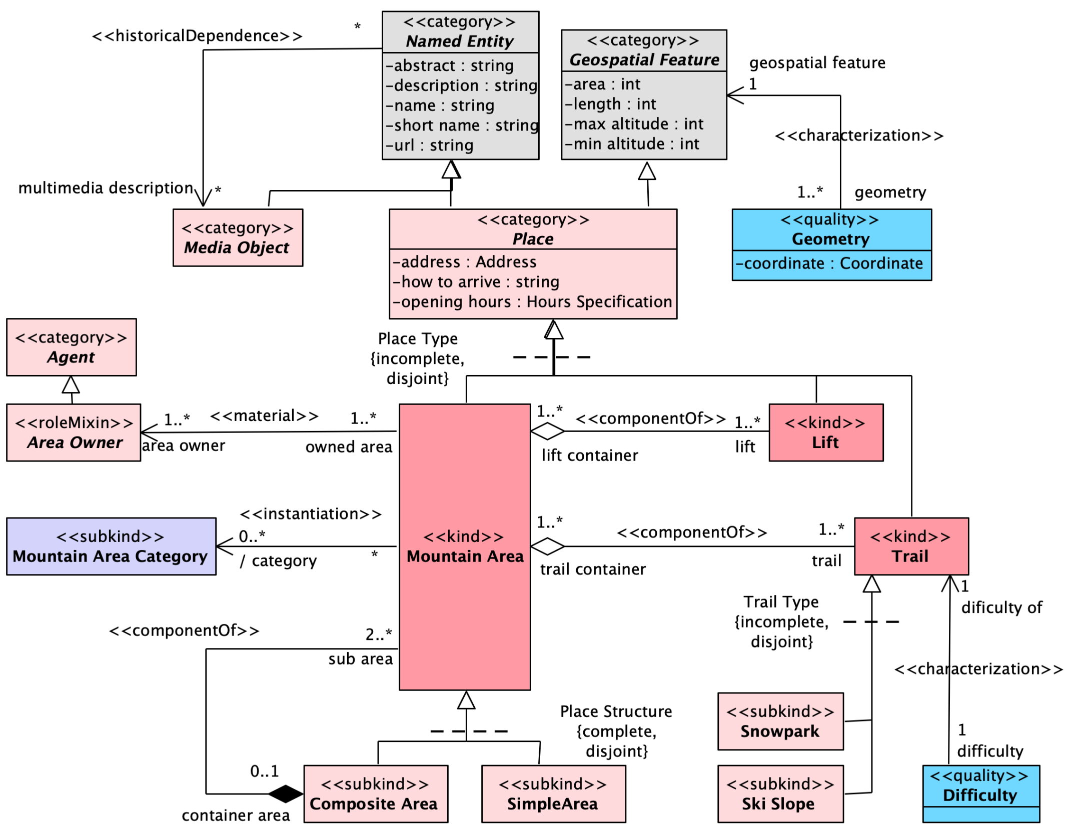


2.2.6. Mountain Areas

A `Mountain Area` is a geographical region in which alpine sports and activities can be performed, such as skiing, snowboarding, climbing, and hiking.

Examples include Dolomiti Superski, located in South Tyrol, Italy, Zermatt Matterhorn, located in Switzerland, and St. Anton Arlberg, located in Austria.

An excerpt of the ontology regarding `Mountain Area` its properties is shown below:



As a particular type of `Place`, `Mountain Area` inherits all of its attributes and relations, which includes `address` and GPS-related properties.

Instances of `Mountain Area` are alpine regions that contain other places of interest, among which instances of `Lift` and `Trail` are commonly found.

`Mountain Areas` may contain other mountain areas. This is the case of Dolomiti Superski and Ortler Skiarena, which contain several smaller areas in South Tyrol. In such cases, the broader area contains, by transitivity, every `Lift` and `Trail` within its contained areas.

Note, the composition of mountain areas is:

- *irreflexive*: an area cannot contain itself;
- *transitive*: if area X contains area Y and area Y contains area Z, then X contains area Z;
- *antisymmetric*: if area X contains area Y, area Y cannot contain area X.

A `Mountain Area` is usually owned by an organization. For instance, the Merano 2000 area is owned by the Merano 2000 Funivie Spa organization.

2.2.7. Trails and Lifts

A `Trail` is a physical path where sporting activities can be performed. Trails are classified according to the sports they afford, such as hiking trails, ski trails, and biking trails, which support hiking, skiing, and biking respectively. This version of the standard distinguishes between two types of trails, namely `Ski Slope` (aka ski trail) and `Snowpark`.

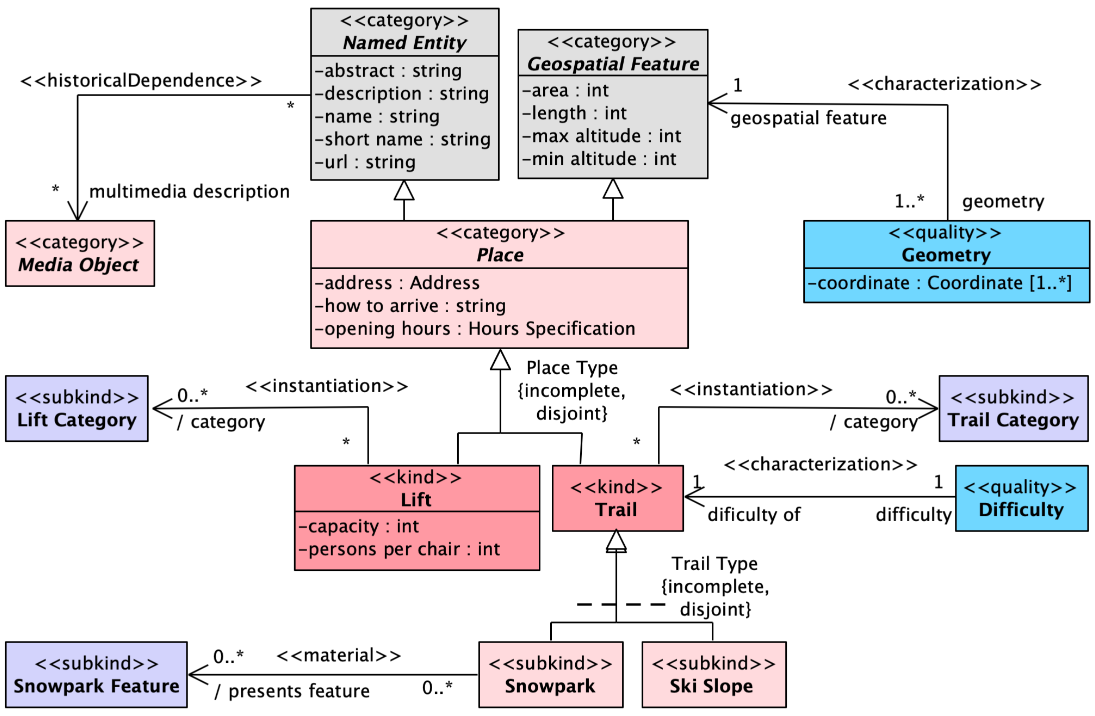
Trails are rated according to how challenging it is to practice a sport in them. This characteristic is captured by means of a general `Difficulty` quality, which can be projected in different scales. For instance, a `Ski Slope's `Difficulty` can be valued according to the european or american systems, as in the Falzeben I slope of the Merano 2000 area is rated blue (or easy) in the european system.

A `Ski Slope` is a `Trail`, usually on a hill, where people can ski and snowboard.

A `Snowpark` is a `Trail` specially designed to allow skiers and snowboarders to perform freestyle tricks by providing them with special features, such as jumps and rails.

A `Lift` is a machine designed to transport people uphill, often being used to transport skiers in mountain areas. `Lifts` may be of several types, such as chairlifts, gondolas, and cableways.

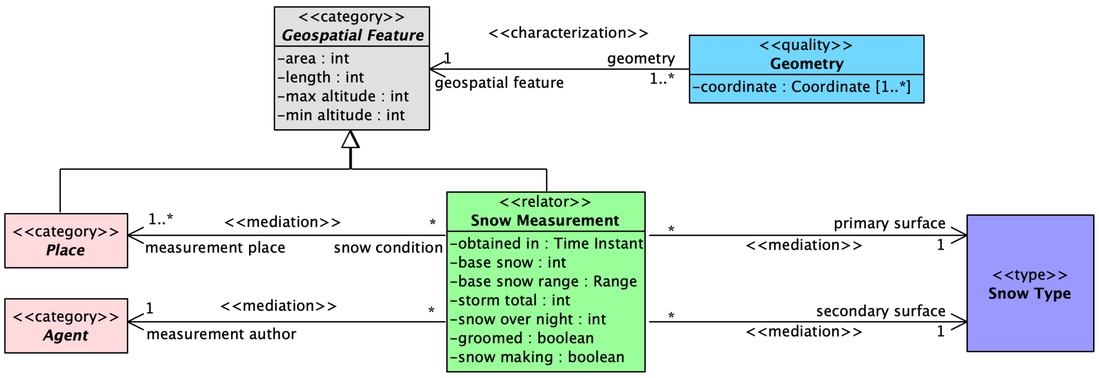
A fragment of the ontology that formalizes `Trail`, `Ski Slope`, `Snowpark`, and `Lift` is presented below:



2.2.8. Snow Measurement

A relevant piece of information for alpine tourists during the winter season is the snow condition of mountain areas and ski slopes. This is captured in the ontology by means of the `Snow Measurement` concept, a relator that describes the result of a measuring activity performed by an `Agent`, at a particular `Place`, in a given point in time (`date`).

An excerpt of the ontology describing `Snow Measurement` is shown below:



Note that `Snow Measurement` specializes `Geospatial Feature`. This means that its instances may be localized using GPS `Geometries` of any shape.

A `Snow Measurement` identifies the following descriptive properties:

- `base snow`: the height of snow in the vicinities of the measurement location
- `base snow range`: the variation of the `base snow` measurement in the vicinities of the measurement location
- `groomed`: indicates whether or not the snow has been groomed in the vicinities of the measurement location
- `primary surface`: the main type of snow found in the vicinities of the measurement location
- `secondary surface`: the secondary type of snow found in the vicinities of the measurement location
- `snow making`: indicates whether or not the snow in the vicinities of the measurement location has been artificially produced
- `snow over night`: the height of snow accumulated in the previous night in the vicinities of the measurement location
- `storm total`: the height of snow accumulated from the last snowstorm in the vicinities of the measurement location

3. API Architecture

AlpineBits® DestinationData specifies a REST API designed to support the client-server communication model, with a focus on system-to-system communication.

This API is built upon JSON:API v1.0, a specification containing best practices for the implementation of REST APIs that exchange JSON data over HTTP.

AlpineBits® DestinationData relies on its underlying transport protocol to take care of security issues. Hence **the use of HTTPS is REQUIRED**.

3.1. SERVER and CLIENT Responsibilities

A SERVER MUST be able to handle GET requests on all routes it implements (see the standard's list of routes).

A SERVER MAY support POST, PATCH, and DELETE requests in order to provide provide resource creation, update, and deletion by CLIENTS.

CLIENT and SERVER MUST observe the prescriptions of this standard when performing or responding to the aforementioned requests, using the appropriate headers, message formats.

SERVERS are RECOMMENDED to document their own implementations of the **AlpineBits® DestinationData** standard as OpenAPI-based documentations.

3.2. Messages

A message is a request sent from a CLIENT to a SERVER, or a response sent from a SERVER to a CLIENT.

CLIENTS and SERVERS MUST include the header `Content-Type: application/vnd.api+json` without any media type parameters to all messages that include a body. This requirement is designed to accommodate future versions and extensions of JSON:API.

All data exchanged between CLIENTS and SERVERS MUST be in the JSON format conforming to the JSON:API v1.0 standard.

All messages containing a body MUST be encoded in `UTF-8`.

Messages can be either request messages, success messages, or error messages. The remainder or this section describes the details of each.

3.2.1. Request Messages

Requests sent from a CLIENT to a SERVER are referred to as request messages.

A CLIENT MUST include the header `Accept: application/vnd.api+json` at least once and without any media type parameters in all requests to a SERVER to represent that the expected response MUST be conformant to this standard.

A SERVER MAY support additional media types, such as `application/json` and `application/xml`, but their adoption is not regulated by this standard.

```
GET /2022-04/events HTTP/1.1
Accept: application/vnd.api+json
```

```
GET /2022-04/events HTTP/1.1
Accept: application/vnd.api+json, application/vnd.api+json;modified-parameter=value,
application/json
```

Depending on the related request, a CLIENT MAY include a JSON body on a request message (see

Requests and Responses). This body MUST contain one of the following fields:

- `data`: an resource object the CLIENT wishes to create or update. This field MUST is present in certain successful CLIENT requests and SERVER responses (see Requests and Responses).

When present, a CLIENT MAY include the following fields to a request message:

- `jsonapi`: an object describing the JSON:API v1.0 the message is complies to. This field is OPTIONAL and, if set, it MUST be equal to the following object:

  ```
  {
    "version": "1.0"
  }
  ```

The examples below present the basic structure of a request message which varies according to the context they are used, as described in Requests and Responses.

```
GET /2022-04/events HTTP/1.1
Accept: application/vnd.api+json
```

```
POST /2022-04/events HTTP/1.1
Content-Type: application/vnd.api+json

{
  "jsonapi": {
    "version": "1.0"
  },
  "data": {
    "id": ...,
    "type": "events",
    ...
  }
}
```

A CLIENT MUST NOT include additional fields in data messages, such as `links`, `meta`, or `included`.

A SERVER MUST reject requests including a body when it is not supported (see Requests and Responses).

A SERVER SHOULD ignore additional fields sent by a CLIENT on a data message.

3.2.2. Success Messages

Successful responses sent from a SERVER to a CLIENT are referred to as success messages.

A success message header MUST contain an HTTP status code in the class 2xx Success, as in:

```
HTTP/1.1 200 OK
Content-Type: application/vnd.api+json
```

Depending on the related request, a SERVER MAY include a JSON body on a success message (see Requests and Responses). This body MUST contain one of the following fields:

- `data`: a field containing the resource objects to be sent to the CLIENT. The `data` field MUST be either:
  - if the request expects a collection of resource objects in the response, the `data` field MUST be an array that contains the expected resources or is empty
  - if the request expects a single resource object in the response, the `data` field MUST be a single resource object or `null`

For details on the different types of resources in this standard and their inner structure, refer to Resources.

- `links`: an object containing links for navigating the API. Examples of links include self-referencing links, links to resources in relationships, pagination links (see Pagination), and hypermedia links (see Hypermedia Controls).

  Besides root of a message, the `links` object MUST also be present in the root of resource objects and in relationships of resources when these are sent from a SERVER to a CLIENT. Refer to Hypermedia Controls for more details.

- `meta`: an object containing the message's metadata. Examples of metadata include the number of resources available in a certain route (see the `count` and `page` fields in Pagination).

When present, a SERVER MAY include the following fields to a success message:

- `jsonapi`: an object describing the JSON:API v1.0 the message is complies to. This field is OPTIONAL and, if set, it MUST be equal to the following object:

  ```
  {
    "version": "1.0"
  }
  ```

- `included`: an array of resource objects listed in selected relationships of the resources present in the `data` field. This field is optional but non-nullable.

  See Inclusion of Related Resources for more details on the resource inclusion feature.

Success messages SHALL NOT have fields in addition to those listed above.

The example below presents the basic structure of a success message from a SERVER to a CLIENT. The structure of messages vary according to the context they are used, as described in Requests and Responses.

```
{
  "jsonapi": {
    "version": "1.0"
  },
  "meta": { ... },
  "links": { ... },
  "data": [ ... ],
  "included": [ ... ]
}
```

A SERVER MUST reply with specific status codes in the following cases:

- `200 OK`: when the CLIENT's request has been successfully processed, with the exceptions of:
  - creation requests
  - deletion requests
- `201 Created`: when the CLIENT's creation request has been successfully processed
- `204 No Content`: when the CLIENT's deletion request has been successfully processed

3.2.3. Error Messages

An error message results from an unsuccessful request made to a SERVER, regardless of the different contexts detailed in Requests and Responses.

An error message header MUST contain an HTTP status code either in the class Client Error 4xx or the class Server Error 5xx, as in:

```
HTTP/1.1 401 Unauthorized
```

If a request message results in multiple errors, the returned status code MUST be the most general status code applicable. For example, if a request results in the errors `415 Unsupported Media Type` and `410 Gone`, the header SHOULD contain the error code `400 Bad Request`.

A SERVER MAY include a body in an error message. If it does, this body MUST contain one of the following fields:

- `errors`: an array of error objects as defined in JSON:API v1.0. This field cannot be empty.
- `links`: an object containing links (see Hypermedia Controls). This field must contain a `self` link contain the route present in the CLIENT’s.

When present, a SERVER MAY include the following fields to a error message:

- `jsonapi`: an object describing the JSON:API v1.0 the message is complies to. This field is OPTIONAL and, if set, it MUST be equal to the following object:

  ```
  {
    "version": "1.0"
  }
  ```

- `meta`: an object that contains non-standard meta-information about its container. This field is OPTIONAL but cannot be empty.

Error messages SHALL NOT have fields in addition to those listed above.

An overview of the basic structure of an error message body is depicted below:

```
{
  "jsonapi": {
    "version": "1.0"
  },
  "meta": { ... },
  "errors": [
    {
      "status": "404",
      "title": "Endpoint not available"
    }
  ],
  "links": { ... }
}
```

The following example presents an error message (header and body) containing multiple error objects.

```
HTTP/1.1 404 Not Found
Content-Type: application/vnd.api+json

{
    "jsonapi": {
        "version": "1.0"
    },
    "errors": [
        {
            "status": 404,
            "title": "Resource not found."
        },
        {
            "status": 405,
            "title": "Method not supported."
        }
    ]
}
```

A SERVER MUST reply with specific status codes in the following cases:

- `400 Bad Request`: when a CLIENT makes a request containing:
  - a method body when it is not supposed to (see Requests and Responses)
  - a query parameter not defined in this standard
  - a query parameter not supported for the specific request
  - a query parameter with an incorrect value
  - conflicting query parameters
  - a method that does not comply the standard’s definitions
  - a method that requests the creation of a resource with an `id` that is already used or malformed
  - a creation or update of a resources' relationship referring to a resource that does not exist
- `401 Unauthorized`: when a CLIENT makes a request:
  - using an authentication method, but the credentials fail to be validated by the SERVER
  - without an authentication method, but the SERVER does not accept such requests
- `404 Not Found`: when a CLIENT makes a request:
  - to an endpoint not defined in this standard
  - to a non-implemented endpoint
  - asking for a non-existing resource
  - asking for a non-existing page number
- `405 Method Not Allowed`: when a CLIENT makes a request using an HTTP method that is not supported by the SERVER.
- `406 Not Acceptable`: when a CLIENT makes a request with an `Accept` header containing a media type not supported by the SERVER or with encoding parameters in `application/vnd.api+json`.

3.3. Authentication

For authenticated requests, a SERVER MUST support authentication through the basic authentication method. A SERVER MAY support additional authentication methods, such as, OAuth, JSON Web Token, and OpenID Connect, as well as non-authenticated requests.

The following example presents the `Authorization` header of a request using basic authentication.

```
GET /2022-04/events HTTP/1.1
Accept: application/vnd.api+json
Authorization: Basic Y2hyaXM6c2VjcmV0
```

In this example, the value of `Authorization` contains the string `john:secret` encoded in base64 (`Y2hyaXM6c2VjcmV0`), as specified in the basic authentication method.

If a SERVER does support authenticated requests, it MUST respond to unauthorized requests with the `401 Unauthorized` status code (see Error Messages). If an error message body is included in the response, the SERVER MAY use an error object to differentiate between requests that lack an authorization header from those that contain invalid credentials.

```
HTTP/1.1 401 Unauthorized
Content-Type: application/vnd.api+json

{
   "error": [{
      "status": "401",
      "title": "Unauthenticated request"
   }],
   ...
}
```

```
HTTP/1.1 401 Unauthorized
Content-Type: application/vnd.api+json

{
   "error": [{
      "status": "401",
      "title": "Invalid credentials"
   }],
   ...
}
```

3.4. Hypermedia Controls

Hypermedia control is a strategy to encode resource navigation and manipulation within the messages exchanged through an API.

The **AlpineBits® DestinationData** standard adopts the hypermedia strategy defined in JSON:API v1.0, which is briefly described in the following paragraphs.

To provide hypermedia controls for CLIENTS to navigate the API, a SERVER MUST include a `links` object according to what is defined in this standard.

A SERVER MUST include a `links` object with `self` link to the root of every message representing the link used by CLIENT in the request that generates that response.

```
{
  "links": {
    "self": "www.example.com/2022-04/events?fields[events]=name,publisher",
    ...
  },
  "data": [ ... ]
  ...
}
```

A SERVER MUST include a `links` object with `self` link to the root of every resource representing its individual resource route (see Routes).

```
{
  "links": { ... },
  "data": [
    {
      "id": "123",
      "type": "events",
      "meta": { ... },
      "links": {
        "self": "www.example.com/2022-04/events/123",
        ...
      },
      "attributes": { ... },
      "relationships": { ... }
    },
    ...
  ]
  ...
}
```

A SERVER MUST include a `links` object with `related` link to every non-null relationship object (alongside the `data` field) representing the resource relationship route where the related resource(s) are available at (see Routes).

```
{
  "links": { ... },
  "data": [
    {
      "id": "123",
      "type": "events",
      "meta": { ... },
      "links": { ... },
      "attributes": { ... },
      "relationships": {
        "publisher": {
          "data": {
            "id": "456",
            "type": "agents"
          },
          "links": {
            "related": "www.example.com/2022-04/events/123/publisher"
          }
        },
        ...
      }
    },
    ...
  ]
  ...
}
```

A CLIENT SHOULD NOT include any `links` objects to its messages. A SERVER MUST ignore any `links` objects in messages from CLIENTS.

3.4.1. Action Discovery

SERVERS that support additional HTTP methods SHOULD support action discovery.

To do so, a SERVER needs to:

- Support HEAD requests on all of its endpoints
- Answer HEAD requests with messages containing an `Allow` header listing the HTTP methods it supports on the requested endpoint.

The following two snippets exemplify a HEAD request made by a CLIENT and its response sent by a

SERVER.

```
HEAD /2022-04/events HTTP/1.1
Host: https://example.com/
Authorization: Basic Y2hyaXM6c2VjcmV0
```

```
HTTP/1.1 200 OK
Allow: GET,POST,PATCH,DELETE
```

A response message to a HEAD request message does not contain a body.

4. Requests and Responses

This chapter defines the different interactions between CLIENTS and SERVERS for the creation, retrieval, update, and deletion of data.

4.1. Resource Retrieval

A CLIENT MAY retrieve links and resource objects from a SERVER through an HTTP GET request. The GET request MUST be performed on the route corresponding to the desired resource(s) (see Routes).

The following example demonstrates a GET request for the retrieval of resources available on the `events` route (see Resource Routes), including also an authorization header.

```
GET /2022-04/events HTTP/1.1
Host: https://example.com
Accept: application/vnd.api+json
Authorization: Basic Y2hyaXM6c2VjcmV0
```

A CLIENT MUST NOT include a `Content-Type` header on retrieval requests. A CLIENT MUST NOT include a body on retrieval requests.

If a CLIENT's retrieval request includes a body or includes invalid headers, the SERVER MUST respond with the corresponding error code (see Error Messages).

If the route requested by a CLIENT on a retrieval request is implemented by a SERVER, it MUST respond to successful requests with data corresponding available data. The SERVER response to a successful request MUST include the status code `200 OK`. The SERVER response to a successful request MUST include the header `Content-Type`.

The example below demonstrates a valid request for the retrieval of a collection of agent resources (see events).

```
HTTP/1.1 200 OK
Content-Type: application/vnd.api+json

{
   "jsonapi": { ... },
   "meta": { ... },
   "links": { ... },
   "data": [
      {
         "type": "events",
         "id": "1",
         "meta": { ... },
         "attributes": { ... },
         "relationships": { ... },
         "links": { ... }
      }
   ]
}
```

The **AlpineBits® DestinationData** standard specifies a set of API features to be implemented by a SERVER. These features are intended to make data retrieval requests from CLIENTS to SERVERS more effective and efficient (see Resource Retrieval).

The list of features is as follows: Pagination, Sorting, Random sorting, Filtering, Searching, Sparse Fieldsets, and Inclusion of Related Resources.

4.1.1. Pagination

The pagination feature allows CLIENTS to request a subset of a resource collection.

A SERVER MUST implement page-based pagination on retrieval requests for all resource collection routes (see Routes).

The response message containing such a subset is called a **page**.

The pagination strategy adopted in this standard uses two parameters:

- `page[size]`: the number of resources included in a page.
- `page[number]`: the number of the page

The following request message exemplifies the use of pagination parameters:

```
GET /2022-04/events?page[size]=10&page[number]=2 HTTP/1.1
Host: https://example.com
Accept: application/vnd.api+json
```

The request asks for the second page (`page[number]=2`) in the collection of event resources, divided into pages containing 10 events each (`page[size]=10`).

A SERVER that implements pagination MUST:

- Be able to handle requests containing just a `page[size]` query parameter, just a `page[number]`, or both.
- Support pagination for GET requests in every base endpoint, such as `/2022-04/events` and `/2022-04/mountainAreas`.
- Respond with the first page if no pagination parameters are sent in the request.
- In paginated responses, add the following fields to the `links` object in the message body:
  - `first`: a string that represents the URL to request the first page of resources in that endpoint. Non-nullable.
  - `last`: a string that represents the URL to request the last page of resources in that endpoint. Non-nullable.
  - `self`: a string that represents the URL to request the current page of resources in that endpoint. Non-nullable.
  - `next`: a string that represents the URL to request the next page of resources in that endpoint. The `next` URL MUST be the same as `last` in case the next page is out of bounds. Non-nullable.
  - `prev`: a string that represents the URL to request the previous page of resources in that endpoint. The `prev` URL MUST be the same as `first` in case the previous page is out of bounds. Non-nullable.

  ```
  HTTP/1.1 200 OK
  Content-Type: application/vnd.api+json

  {
    "links": {
      "first": "https://example.com/2022-04/events?page[number]=1",
      "last": "https://example.com/2022-04/events?page[number]=100",
      "self": "https://example.com/2022-04/events?page[number]=2",
      "next": "https://example.com/2022-04/events?page[number]=3",
      "prev": "https://example.com/2022-04/events?page[number]=1"
    },
    ...
  }
  ```

- In paginated responses, add the following fields to the `meta` object in the message body:
  - `count`: a number that indicates the number of resources available in the endpoint. Non-nullable.
  - `pages`: a number that indicates the number of pages in the resource collection given the selected

page size. Non-nullable.

```
HTTP/1.1 200 OK
Content-Type: application/vnd.api+json

{
  "meta": {
    "count": 1000,
    "pages": 100
  },
  ...
}
```

- Respond a request for a non existing page with an error message whose status code is `404 Not Found`.

```
GET /2022-04/events?page[number]=10000 HTTP/1.1
Host: https://example.com
Accept: application/vnd.api+json
```

```
HTTP/1.1 404 Not Found
Content-Type: application/vnd.api+json
```

A SERVER that implements pagination SHOULD:

- Define a default page size.
- Respond a request for a non existing page with an error message containing a body as follows:

```
GET /2022-04/events?page[number]=10000 HTTP/1.1
Host: https://example.com
Accept: application/vnd.api+json
```

```
HTTP/1.1 404 Not Found
Content-Type: application/vnd.api+json

{
  "error": [
    {
      "status": "404",
      "title": "Page not found"
    }
  ],
  "links": {
    "self": "http://example.com/2022-04/events?page[number]=10000"
  }
}
```

4.1.2. Sorting

A SERVER SHOULD support requests to sort resource collections according to one or more criteria (**sort fields**), as defined in the JSON:API v1.0 specification.

A SERVER that implements this feature MUST:

- Interpret the values of the `sort` query parameter as sort fields:

```
GET /2022-04/events?sort=name HTTP/1.1
Accept: application/vnd.api+json
```

- Interpret a sort field to be ascending unless it is preceded by a minus (U+002D HYPHEN-MINUS, "-"), in which case it MUST interpret it to be descending:

```
GET /2022-04/events?sort=-name HTTP/1.1
Accept: application/vnd.api+json
```

- Interpret a dot-separated (U+002E FULL-STOP, ".") sort field to be a request to sort by a relationship or a nested attribute:

```
GET /2022-04/events?sort=organizer.name HTTP/1.1
Accept: application/vnd.api+json
```

- Respond with a `400 Bad Request` if a request contains a sort field that is not supported by the SERVER:

```
GET /2022-04/events?sort=hello HTTP/1.1
Accept: application/vnd.api+json
```

```
HTTP/1.1 400 Bad Request
Content-Type: application/vnd.api+json

{
  "links": {
    "self": "https://example.com/2022-04/events?sort=hello",
  },
  "errors": [{
      "title": "Invalid query parameter value.",
      "status": 400
  }]
}
```

A SERVER that implements sorting SHOULD:

- Adopt sort fields that correspond to resource attributes:

```
GET /2022-04/events?sort=name HTTP/1.1
Accept: application/vnd.api+json
```

- Support sorting by combining multiple sort fields:

```
GET /2022-04/events?sort=name,startDate HTTP/1.1
Accept: application/vnd.api+json
```

A SERVER that implements sorting MAY:

- Adopt sort fields that do not correspond to resource attributes and relationships.

```
GET /2022-04/events?sort=date HTTP/1.1
Accept: application/vnd.api+json
```

4.1.3. Random sorting

A SERVER MAY support requests to randomly sort resource collections according to a **seed** value.

A SERVER that implements this feature MUST:

- Interpret the value of the `random` query parameter as the seed to be used to sort the resource collection:

```
GET /2022-04/events?random=5 HTTP/1.1
Accept: application/vnd.api+json
```

- Always return resources in the same order for requests using a given seed value, provided everything else remains the same (e.g. the resource collection, the page size, and the page number)
- Respond with a `400 Bad Request` if a request combines a `sort` and a `random` query parameter:

```
GET /2022-04/events?random=5&sort=startDate HTTP/1.1
Accept: application/vnd.api+json
```

```
HTTP/1.1 400 Bad Request
Content-Type: application/vnd.api+json

{
  "links": {
    "self": "https://example.com/2022-04/events?random=5&sort=startDate",
  },
  "errors": [{
      "title": "Request contains conflicting queries.",
      "status": 400
  }]
}
```

- Respond with a `400 Bad Request` if a request includes a seed value that is not supported by the SERVER:

```
GET /2022-04/events?random=hello HTTP/1.1
Accept: application/vnd.api+json
```

```
HTTP/1.1 400 Bad Request
Content-Type: application/vnd.api+json

{
  "links": {
    "self": "https://example.com/2022-04/events?random=hello",
  },
  "errors": [{
      "title": "Invalid query parameter value.",
      "status": 400
  }]
}
```

A SERVER that implements random sorting SHOULD:

- Allow a CLIENT to combine `random` with every other query parameter defined in this document, except for `sort`:

```
GET /2022-
04/events?random=12&fields[events]=name,startDate&page[size]=30&page[number]=2 HTTP/1.1
Accept: application/vnd.api+json
```

4.1.4. Filtering

A SERVER SHOULD support filtering over its resources.

This standard specifies two strategies for filtering, `label-specific filters` and `simple generic` `filters`:

- `Label-specific filter`: the SERVER defines the name of each supported filter, its semantics, and the values it may receive. Label-specifics filters MUST be defined as `filter[FILTER-NAME]=VALUES`, where `VALUES` may be a list of comma-separated values.

  For instance, a filter for resources updated after 01/01/2021 could be defined as follows:

  ```
  GET /2022-04/events?filter[lastUpdate]=2021-01-01 HTTP/1.1
  Accept: application/vnd.api+json
  ```

- `Simple generic filter`: this standard defines a list of generic filter operands, their semantics, and the values it may receive. These filters are then requested over resources' fields following the pattern `filter[FIELD][OPERAND]=VALUES`.

  For instance, a request to retrieve events starting after 01/01/2021 is defined as follows.

  ```
  GET /2022-04/events?filter[startDate][gt]=2021-01-01 HTTP/1.1
  Accept: application/vnd.api+json
  ```

A SERVER is RECOMMENDED to support filtering over the following data:

- Last update
- Language
- Categories
- Date (events)
- Publisher (events)
- Location (geolocation or address based)
- Length (ski slopes, snowparks)
- Opening schedule - open at (places open on a certain date)
- Difficulty (ski slopes, snowparks)
- Snow conditions (mountain areas, snowparks)
- Area owner (mountain areas)

A SERVER MAY support multiple filters in a single request.

A SERVER MUST reply to requests with non-supported filters, field names, or values with `400 Bad Request` status code:

```
GET /2022-04/events?filter[foo]=bar HTTP/1.1
Accept: application/vnd.api+json
```

```
HTTP/1.1 400 Bad Request
Content-Type: application/vnd.api+json
{
  "links": {
    "self": "https://example.com/2022-04/events?filter[foo]=bar",
  },
  "errors": [{
      "title": "Invalid query parameter value.",
      "status": 400
  }]
}
```

A SERVER MAY support custom filters, i.e., those that are cannot be expressed as simple generic filters, as label-specific filters: For example, a filter retrieves events "active" in between two dates:

+

```
GET /2022-04/events?filter[between]=2021-03-01,2021-03-10 HTTP/1.1
Accept: application/vnd.api+json
```

A SERVER SHOULD prefer simple generic filters over label specific filters.

A SERVER MAY support simple generic filters over nested fields. For example, a filter over the countries in venues' addresses:

+

```
GET /2022-04/venues?filter[address.country][eq]=IT HTTP/1.1
Accept: application/vnd.api+json
```

The standard defines the following list of operands for simple generic filters:

- `exists`: filters resources that according to whether or not the field has values (i.e., it is not `null`). Accepts as argument a single `true` or `false` value.
  For example, filter event resources that have multimedia descriptions:

  ```
  GET /2022-04/events?filter[multimediaDescriptions][exists]=true HTTP/1.1
  Accept: application/vnd.api+json
  ```

- `eq`: filters resources that have the field assigned to a specific value. Accepts as argument a single `boolean`, `number`, or `string` value.
  For example, filter venue resources whose address has a specific country value:

  ```
  GET /2022-04/venues?filter[address.country][eq]=IT HTTP/1.1
  Accept: application/vnd.api+json
  ```

- `neq`: filters resources that have the field not assigned to a specific value. Accepts as argument a single `boolean`, `number`, or `string` value.
  For example, filter snowpark resources whose difficulty level is not `"expert"`:

  ```
  GET /2022-04/snowparks?filter[difficulty][neq]=expert HTTP/1.1
  Accept: application/vnd.api+json
  ```

- `in`: filters resources that have the field assigned to values within a desired list. Accepts a list of `number` or `string` arguments.
  For example, filter snowpark resources whose difficulty is within the list `[ "beginner", "intermediate" ]`.

  ```
  GET /2022-04/snowparks?filter[difficulty][in]=beginner,intermediate HTTP/1.1
  Accept: application/vnd.api+json
  ```

- `nin`: filters resources that have the field not assigned to values within a desired list. Accepts a list of `number` or `string` arguments.
  For example, filter snowpark resources whose difficulty is not within the list `[ "advanced", "expert" ]`.

  ```
  GET /2022-04/snowparks?filter[difficulty][nin]=advanced,expert HTTP/1.1
  Accept: application/vnd.api+json
  ```

- `any`: filters resources that have some of the field’s values within a desired list. Accepts a list of `number` or `string` arguments.
  For example, filter event resources whose categories include some of the values in the list `[ "schema:MusicEvent", "schema:SportsEvent" ]`.

  ```
  GET /2022-04/events?filter[categories][any]=schema:MusicEvent,schema:SportsEvent
  HTTP/1.1
  Accept: application/vnd.api+json
  ```

- `all`: filters resources that have all of the field’s values within a desired list. Accepts a list of `number` or `string` arguments.
  For example, filter event resources whose categories include some of the values in the list `[ "schema:Festival", "schema:MusicEvent" ]`.

  ```
  GET /2022-04/events?filter[categories][any]=schema:Festival,schema:MusicEvent HTTP/1.1
  Accept: application/vnd.api+json
  ```

- `gt`: filters resources that have the field assigned to a value greater than the desired one. Accepts as argument a single `number` or `string`.
  For example, retrieve agent resources that have their last update greater than a certain date-time:

  ```
  GET /2022-04/agents?filter[lastUpdate][gt]=2022-04-01T00:00:00+0000 HTTP/1.1
  Accept: application/vnd.api+json
  ```

- `gte`: filters resources that have the field assigned to a value greater than or equal to the desired one. Accepts as argument a single `number` or `string`.
  For example, retrieve agent resources that have their last update greater than or equal to a certain date-time:

  ```
  GET /2022-04/agents?filter[lastUpdate][gte]=2022-04-01T00:00:00+0000 HTTP/1.1
  Accept: application/vnd.api+json
  ```

- `lt`: filters resources that have the field assigned to a value lower than the desired one. Accepts as argument a single `number` or `string`.
  For example, retrieve ski slope resources that have their length lower than a certain value:

  ```
  GET /2022-04/skiSlopes?filter[length][lt]=5000 HTTP/1.1
  Accept: application/vnd.api+json
  ```

- `lte`: filters resources that have the field assigned to a value lower than or equal to the desired one. Accepts as argument a single `number` or `string`.
  For example, retrieve ski slope resources that have their length lower than or equal to a certain value:

  ```
  GET /2022-04/skiSlopes?filter[length][lte]=5000 HTTP/1.1
  Accept: application/vnd.api+json
  ```

- `near`: filters resources located within a certain distance to a pair of coordinates. Accepts a triple of `number` arguments representing a coordinate’s longitude and latitude, and a distance in meters (i.e., `LONGITUDE,LATITUDE,DISTANCE`).
  For example, retrieve lift resources whose geometries are within 10000 meters geo-located pair of longitude and latitude (11.891472,46.92275):

```
GET /2022-04/lifts?filter[geometries][near]=11.891472,46.92275,10000 HTTP/1.1
Accept: application/vnd.api+json
```

- `intersects`: filters resources whose geo-location data intersects a desired area. Accepts as argument a single `string` representing a GeoJSON object of type "Polygon" (see geometry).
  For example, filter mountain areas whose geometry intersects a specific area (e.g., the area surrounding a mountain).

```
GET /2022-
04/mountainAreas?filter[geometries][intersects]={"type":"Polygon","coordinates":
[[11.3490,46.4976],[11.3508,46.4975],[11.3510,46.4989],[11.3492,46.4990]]]} HTTP/1.1
Accept: application/vnd.api+json
```

- `within`: filters resources whose geo-location data is within a desired area. Accepts as argument a single `string` representing a GeoJSON object of type "Polygon" (see geometry).
  For example, filter venues whose geometry is within a specific area.

```
GET /2022-04/venues?filter[geometries][within]={"type":"Polygon","coordinates":
[[11.3490,46.4976],[11.3508,46.4975],[11.3510,46.4989],[11.3492,46.4990]]]} HTTP/1.1
Accept: application/vnd.api+json
```

- `starts`: filters resources whose text field starts with a desired substring. Accepts as argument a single `string`.
  For example, filter media object resources whose content type string start with a certain substring:

```
GET /2022-04/mediaObjects?filter[contentType][starts]=video HTTP/1.1
Accept: application/vnd.api+json
```

- `ends`: filters resources whose text field ends with a desired substring. Accepts as argument a single `string`.
  For example, filter media object resources whose content type string ends with a certain substring:

```
GET /2022-04/mediaObjects?filter[contentType][end]=png HTTP/1.1
Accept: application/vnd.api+json
```

- `regex`: filters resources whose text field passes a desired regular expression. Accepts as argument a single `string` representing a regular expression.
  For example, filter media object resources whose content type passes a certain regular expression:

```
GET /2022-04/mediaObjects?filter[contentType][regex]=^(audio|image|video) HTTP/1.1
Accept: application/vnd.api+json
```

4.1.5. Searching

A SERVER SHOULD support text search over its resources.

A SERVER that implements this feature MUST:

- Interpret the value of a `search[FIELD]` query parameter as the string to be used to search for resources, while the value between square brackets refers to the resources' field that should be considered in the search:

```
GET /2022-04/events?search[name]=bolzano HTTP/1.1
Accept: application/vnd.api+json
```

- Respond with a `400 Bad Request` if a request searches for a query string on a field that does not exist on the resource types returned by the queried endpoint.

```
GET /2022-04/events?search[license]=bolzano HTTP/1.1
Accept: application/vnd.api+json
```

```
HTTP/1.1 400 Bad Request
Content-Type: application/vnd.api+json

{
  "links": {
    "self": "https://example.com/2022-04/events?search[license]=bolzano",
  },
  "errors": [{
      "title": "Invalid query parameter value.",
      "status": 400
  }]
}
```

A SERVER that implements searching SHOULD:

- Support searching at least over the `name` and `description` attributes, which are defined for all resource types:

```
GET /2022-04/events?search[name]=bolzano HTTP/1.1
Accept: application/vnd.api+json
```

- Allow CLIENTS to combine a `search` query parameter with any other query parameter defined in this standard:

```
GET /2022-
04/events?search[name]=bolzano&page[size]=20&fields[events]=name,startDate&sort=-
startDate HTTP/1.1
Accept: application/vnd.api+json
```

A SERVER that implements searching MAY:

- Support searching over all fields of a resource type by accepting a `search` query parameter that is not followed by square brackets:

```
GET /2022-04/events?search=bolzano HTTP/1.1
Accept: application/vnd.api+json
```

- Choose how its search feature behaves. For instance, a SERVER MAY return resources whose fields match or contain the search string, or whose fields contain a substring that is similar to the search string.

4.1.6. Sparse Fieldsets

A SERVER MAY support requests to return only specific fields that characterize a resource on a per-type basis, as defined in the JSON:API v1.0 specification

This feature applies both to endpoints that return collections of resources, e.g. `/2022-04/events`, and those that return individual resources, e.g. `/2022-04/events/123`.

A SERVER that implements this feature MUST:

- Interpret the comma-separated (U+002C COMMA, ",") values of `field[TYPE]` query parameters as the selected fields (attributes or relationships) to be returned for a given resource type, which in turn is identified by the value between square brackets:

```
GET /2022-04/agents?fields[agents]=name,contactPoints HTTP/1.1
Host: https://example.com
Accept: application/vnd.api+json
```

```
HTTP/1.1 200 OK
Content-Type: application/vnd.api+json

{
  "links": {
      "self": "https://example.com/2022-04/agents?fields[agents]=name,contactPoints",
  },
  "data": [{
      "type": "agents",
      "id": "1",
      "meta": { ... },
      "attributes": {
        "name": {
            "eng": "Free University of Bozen-Bolzano"
        },
        "contactPoints": { ... },
      },
      "relationships": null,
      "links": { ... }
  }]
}
```

- Be able to process one `field[TYPE]` query parameter per resource type:

```
GET /2022-
04/events?fields[events]=name,startDate&fields[agents]=name,contactPoints&include=organ
izers HTTP/1.1
Host: https://example.com
Accept: application/vnd.api+json
```

- NOT include additional fields in resource objects if a CLIENT requests a restricted field set (using `fields[]`) for the respective resource types.
- Send all fields for resource objects if a CLIENT does not select a restricted field set (using `fields[]`) for the respective resource types.
- Respond with a `400 Bad Request` status code a request that selects fields that do not exist for a resource type.

```
GET /2022-04/events?fields[events]=price HTTP/1.1
Accept: application/vnd.api+json
```

- Respond with a `400 Bad Request` status code a request that submits multiple `field[]` parameters for one resource type.

```
GET /2022-04/events?fields[events]=name&fields[events]=name,startDate HTTP/1.1
Accept: application/vnd.api+json
```

- Respond with a `400 Bad Request` status code a request that selects fields on resource types that are not returned in the queried endpoint.

```
GET /2022-04/events?fields[lifts]=name HTTP/1.1
Accept: application/vnd.api+json

HTTP/1.1 400 Bad Request
Content-Type: application/vnd.api+json

{
  "links": {
    "self": "https://example.com/2022-04/events?fields[lifts]=name",
  },
  "errors": [{
      "title": "Query parameter does not have valid values.",
      "status": 400
  }]
}
```

4.1.7. Inclusion of Related Resources

The inclusion of related resources feature allows a CLIENT to request a SERVER to add, in a response message, the resources related to the base resource.

For example, a CLIENT MAY request a SERVER for event resources and to include agent resources that appear in the `organizers` field of the events returned in the response.

An example of a request asking for related resources is shown below:

```
GET /2022-04/events?include=organizers HTTP/1.1
Accept: application/vnd.api+json
```

It is RECOMMENDED that SERVERS support the inclusion of related resources.

In successful messages, a SERVER MUST add the related resources of requested relationships to the `included` array.

To ask for the inclusion of related resources, a CLIENT MUST append `includes` as a URL parameter of the request message.

The resources to be included are listed and separated by commas, as in `/2022-04/events?include=organizers,sponsors` and `/2022-04/mountainAreas?include=skiSlopes,lifts,snowparks`.

To request resources related to other resources, a dot-separated path for each relationship name can be specified, as in `/2022-04/events?include=organizers.multimediaDescriptions` and `/2022-04/mountainAreas?include=multimediaDescriptions.licenseHolder`.

If a SERVER is unable to identify a relationship or does not support the inclusion of resources from a relationship, it MUST respond with `400 Bad Request`.

The following example presents the response to a request for a collection of `mediaObjects` resources with of resources present in the `licenseHolder` relationship.

```
GET /2022-04/mediaObjects?include=licenseHolder HTTP/1.1
Host: https://example.com
Accept: application/vnd.api+json
```

```
HTTP/1.1 200 OK
Content-Type: application/vnd.api+json

{
  "jsonapi": {
    "version": "1.0"
  },
  "meta": { ... },
  "links": { ... },
  "data": [
    {
      "type": "mediaObjects",
      "id": "1",
      "meta": { ... },
      "links": { ... },
      "attributes": { ... },
      "relationships": {
        "categories": null,
        "licenseHolder": {
          "data": {
            "type": "agents",
            "id": "2"
          },
          "links": { ... }
        }
      }
    }
  ],
  "included": [
    {
      "type": "agents",
      "id": "2",
      "meta": { ... },
      "attributes": { ... },
      "relationships": { ... }
    }
  ]
}
```

4.2. Resource Creation

A SERVER MAY support resource creation requests.

If implemented, a SERVER MUST treat resource creation requests as transactions, i.e., requests MUST either fully succeed or fail entirely leaving no partial side-effects.

Resource creation requests MUST be performed through HTTP POST requests (see Messages).

Resource creation requests MUST be supported on resource collection routes (see Resource Routes) of the intended resource type.

Resource creation requests MUST include `Content-Type` and `Accept` headers.

Resource creation requests MUST include a JSON body containing a `data` field, as described in Request Messages.

Resource creation requests MUST include the resource to be created in the `data` field of the body’s message.

A SERVER MUST reject resource creation requests that do not conform to the requirements defined here.

The example below demonstrates a valid resource creation request to create an event resource.

```
POST /2022-04/events HTTP/1.1
Content-Type: application/vnd.api+json
Accept: application/vnd.api+json

{
    "data": {
        "type": "events",
        "id": "123",
        "attributes": {
            "name": {
                "eng": "Südtirol Jazz Festival 2022"
            },
            "startDate": "2022-06-29T00:00:00+00:00",
            "status": "published"
        },
        "relationships": {
            "publisher": {
                "data": {
                    "type": "agents",
                    "id": "1"
                }
            }
        }
    }
}
```

A SERVER MUST respond to successful requests with a status code `201 CREATED`.

A SERVER MUST add the `Content-Type` header on successful responses (see Messages).

A SERVER SHOULD add the `Location` header with the individual resource route of the created resource.

In the success response the SERVER MUST send the created resource, including all fields of the `attributes`, `relationships`, `meta`, and `links` (see Hypermedia Controls) objects, equal to a retrieval request on the created resource's individual route.

A SERVER MUST set the value of `lastUpdate` in the `meta` object of the resource to the moment when the resource was created.

A SERVER MUST assign a value to the `dataProvider` field of the `meta` object of the resource using one of the following options:

- A SERVER MAY require CLIENTS to send the `dataProvider` value in the `meta` object of the resource being created
- A SERVER MAY assign the `dataProvider` value in the `meta` object of the resource using information from outside the message's body (e.g., using the authentication of the request to identify the data provider)

If a SERVER does not use CLIENT-generated values for `dataProvider`, it MUST reject creation requests that include a value for `dataProvider`.

A SERVER SHOULD ignore any additional fields included in the request (see Request Messages).

A SERVER SHOULD ignore any `links` objects present in the request (see Hypermedia Controls).

A SERVER MUST reject creation requests whose resource does not include a valid resource type, `type`, or whose resource lacks any non-nullable fields (see Resources).

A SERVER MUST set to `null` any nullable attribute or relationship that is not present in the creation request's resource (see Resources).

A SERVER MUST reject creation requests whose resource contains relationships to resources that do not exist in the SERVER.

A SERVER MUST accept CLIENT-generated values of `id` as long as the conditions below are meet. Otherwise, the SERVER MUST reject the request.

- the `id` is available in the SERVER
- the `id` conforms to the syntax used by the SERVER

The following example demonstrates the response to a successful resource creation request.

```
HTTP/1.1 201 CREATED
Content-Type: application/vnd.api+json
Location: https://example.com/2022-04/events/123

{
    "jsonapi": { ... },
    "links": { ... },
    "meta": { ... },
    "data": {
        "type": "events",
        "id": "123",
        "links": {
          "self": ...
        },
        "meta": {
            "lastUpdate": ...,
            "dataProvider": ...,
        },
        "attributes": {
            "name": {
                "eng": "Südtirol Jazz Festival 2022"
            },
            "startDate": "2022-06-29T00:00:00+00:00",
            "status": "published",
            "description": null,
            ...
        },
        "relationships": {
            "publisher": {
                "data": {
                    "type": "agents",
                    "id": "1"
                },
                "links": {
                    "related": ...
                }
            },
            "venues": null,
            ...
        }
    }
}
```

A SERVER MUST reply to resource creation requests with an error response in the following cases, using the adequate error codes (see Error Messages).

- the CLIENT does not have authorization to create a resource
- the POST HTTP method is not implemented in the requested route
- the request does not include a well-formed resource in the message’s body
- the request lacks non-nullable fields
- the request includes an invalid CLIENT-generated `id`

4.3. Resource Update

A SERVER MAY support resource update requests.

If implemented, a SERVER MUST treat resource update requests as transactions, i.e., requests MUST either fully succeed or fail entirely leaving no partial side-effects.

Resource update requests MUST be performed through HTTP PATCH requests (see Messages).

Resource update requests MUST be supported on individual resource routes (see Resource Routes) of the identified resource.

Resource update requests MUST include `Content-Type` and `Accept` headers.

Resource update requests MUST include a JSON body containing a `data` field, as described in Request Messages.

Resource update requests MUST include the resource to be update in the `data` field of the body's message.

Resource update requests MUST include the `id` and the `type` of the resource to be update and these values MUST conform to the individual resource route the request was sent to.

Resource update requests MUST include all attributes and relationships to be replaced. Attributes and relationships that are to be removed MUST be set to `null`. Non-nullable attributes and relationships MUST NOT be set to `null`.

A SERVER MUST reject resource update requests that do not conform to the requirements defined here.

The example below demonstrates a valid resource update request to update an event resource.

```
PATCH /2020-04/events/123 HTTP/1.1
Content-Type: application/vnd.api+json
Accept: application/vnd.api+json

{
    "data": {
        "type": "events",
        "id": "123",
        "attributes": {
            "status": "canceled"
        },
        "relationships": {
            "publisher": {
                "data": {
                    "type": "agents",
                    "id": "2"
                }
            },
            "sponsors": null
        }
    }
}
```

A SERVER MUST respond to successful requests with a status code `200 OK`.

A SERVER MUST add the `Content-Type` header on successful responses (see Messages).

In the success response the SERVER MUST send the created resource, including all fields of the `attributes`, `relationships`, `meta`, and `links` (see Hypermedia Controls) objects, equal to a retrieval request on the created resource's individual route.

A SERVER MUST set the value of `lastUpdate` in the `meta` object of the resource to the moment when the resource was updated.

A SERVER SHOULD ignore any additional fields included in the request (see Request Messages).

A SERVER SHOULD ignore any `links` objects present in the request (see Hypermedia Controls).

A SERVER MUST reject creation requests whose resource contains relationships to resources that do not exist in the SERVER.

A SERVER MAY update the value of the `dataProvider` field of the `meta` object of the resource using one of the following options:

- A SERVER MAY keep the `dataProvider` value of the creation assigned during the creation of the resource
- A SERVER MAY require CLIENTS to send the `dataProvider` value in the `meta` object of the resource being update
- A SERVER MAY assign the `dataProvider` value in the `meta` object of the resource using information from outside the message's body (e.g., using the authentication of the request to identify the data provider)

If a SERVER does not use CLIENT-generated values for `dataProvider`, or does not update the value of `dataProvider` after the resource's creation, it MUST reject update requests that include a value for `dataProvider`.

The following example demonstrates the response to a successful resource update request.

```
HTTP/1.1 200 OK
Content-Type: application/vnd.api+json

{
    "jsonapi": { ... },
    "links": { ... },
    "meta": { ... },
    "data": {
        "type": "events",
        "id": "123",
        "meta": { ... },
        "attributes": {
            "status": "canceled",
            ...
        },
        "relationships": {
            "publisher": {
                "data": {
                    "type": "agents",
                    "id": "2"
                },
                "links": {
                    "related": ...
                }
            },
            "sponsors": null
            ...
        }
    }
}
```

A SERVER MUST reply to resource update requests with an error response in the following cases, using the adequate error codes (see Error Messages).

- the CLIENT does not have authorization to update the resource
- the PATCH HTTP method is not implemented in the requested route
- the request does not include a well-formed resource in the message's body
- the request updates non-nullable fields to `null`

4.4. Resource Deletion

A SERVER MAY support resource deletion requests.

If implemented, a SERVER MUST treat resource deletion requests as transactions, i.e., requests MUST either fully succeed or fail entirely leaving no partial side-effects.

Resource deletion requests MUST be performed through HTTP DELETE requests (see Messages).

Resource deletion requests MUST be supported on individual resource routes (see Resource Routes) of the identified resource.

Resource deletion requests MUST NOT include a body, as described in Request Messages.

The example below demonstrates a valid resource deletion request to delete an event resource.

```
DELETE /2022-04/events/123 HTTP/1.1
```

A SERVER MUST respond to successful requests with a status code `204 No Content`.

A SERVER MUST NOT include a body message in responses to successful deletion requests.

The following example demonstrates the response to a successful resource deletion request.

```
HTTP/1.1 204 No Content
```

A SERVER MUST reply to resource deletion requests with an error response in the following cases, using the adequate error codes (see Error Messages).

- the resource identified by the route’s id does not exist
- the CLIENT does not have authorization to delete the resource

5. Routes

The **AlpineBits® DestinationData** standard defines a set of routes to be implemented by SERVERS following the REST architectural style for APIs. These routes commit to a well-defined pattern which is structured as follows:

- Base route: `/`

  The base route provides links to all versions of the **AlpineBits® DestinationData** standard supported by the SERVER. For details, refer to Base Route.

- Version routes: `/<version>`

  The version route provides links to the the collections of all resource types available in the SERVER. For details, refer to Version Routes.

- Resource collection routes: `/<version>/<resourceType>`

  The resource collection route provides the collection of all available resources of the related type. The details for each resource collection route can be found in this chapter its related subsection (see Resource Routes).

- Individual resource routes: `/<version>/<resourceType>/<resourceId>`

  The individual resource route provides the resource that is uniquely identified by that route. The details for each individual resource route can be found in this chapter its related subsection (see Resource Routes).

- Resources' relationships routes: `/<version>/<resourceType>/<resourceId>/<relationship>`

  The resources' relationship route provides all resources related to the one identified in the route through the related relationship. These can be either single resources or collections of resources, depending on the definition of the relationship. The details for each resources' relationship route can be found in this chapter its related subsection (see Resource Routes).

The definition for the fragments in the pattern above have the following interpretation:

- `<version>` is a code representing the version of the **AlpineBits® DestinationData** standard implemented in that route. This allows simultaneous support to multiple standard versions, typically necessary during migration periods. Currently, the following versions can be found:
  - `2020-04`: first release;
  - `2021-04`: second release;
  - `2022-04`: third and current release
- `<resourceType>` is a standard-defined resource type (e.g., `events`, `mountainAreas`, and `venues`). See Resources.
- `<resourceId>`: an ID that uniquely identifies a resource that is an instance of the `<resourceType>`.
- `<relationship>`: field name of a resource's relationship. Allows access to a resource's related resources, such as an event's organizer.

This chapter details all routes that are part of the **AlpineBits® DestinationData** standard version 2022-04, listing the methods and and features that SERVERS MAY be supported by SERVERS on each of them.

5.1. Base Route

To retrieve the versions of **AlpineBits® DestinationData** implemented by a SERVER, a CLIENT MAY send a GET request to that SERVER's base endpoint `/`, as in the example below.

```
GET https://example.com/ HTTP/1.1
Accept: application/vnd.api+json
```

A SERVER MUST respond to requests to the base endpoint `/` with links to each implemented versions. The `data` field in the response body MUST be set to null.

The following example demonstrates the response from a SERVER that has available implementations of the versions `2021-04` and `2022-04` of the **AlpineBits® DestinationData** standard.

```
HTTP/1.1 200 OK
Content-Type: application/vnd.api+json

{
  "links": {
    "self": "https://example.com",
    "2021-04": "https://example.com/2021-04",
    "2022-04": "https://example.com/2022-04"
  },
  "data": null
}
```

SERVERS MUST support the version standard they implement on the version route dedicated to it. For example, the implementation of the **AlpineBits® DestinationData** standard version 2022-04 MUST use the version route `/2022-04` and its sub-routes.

5.2. Version Routes

To retrieve the routes implemented by a SERVER, a CLIENT MAY perform a GET request on the desired version's endpoint (e.g., `/2022-04`), as in the example below.

```
GET https://example.com/2022-04 HTTP/1.1
Accept: application/vnd.api+json
```

A SERVER MUST respond to requests to implemented version routes with links to each implemented resource collection route. The `data` field in the response body MUST be set to null.

The following example demonstrates the response from a SERVER that has available implementations of resource collection routes of all resource types except `categories` and `features`.

```
HTTP/1.1 200 OK
Content-Type: application/vnd.api+json

{
  "links": {
    "self": "https://example.com/2022-04",
    "agents": "https://example.com/2022-04/agents",
    "events": "https://example.com/2022-04/events",
    "eventSeries": "https://example.com/2022-04/eventSeries",
    "lifts": "https://example.com/2022-04/lifts",
    "mediaObjects": "https://example.com/2022-04/mediaObjects",
    "mountainAreas": "https://example.com/2022-04/mountainAreas",
    "skiSlopes": "https://example.com/2022-04/skiSlopes",
    "snowparks": "https://example.com/2022-04/snowparks",
    "venues": "https://example.com/2022-04/venues"
  },
  "data": null
}
```

5.3. Resource Routes

Resource routes are those routes that expose the resources defined in the **AlpineBits® DestinationData** standard. This section lists all routes defined in this standard, describing what the return as response to a GET request, what HTTP methods the MAY respond to (see Requests and Responses), and what additional features a SERVER MAY support on GET requests (see resource retrieval features).

A SERVER MUST listen to HTTP requests on all routes listed in this section. A SERVER MUST respond with a `404 Not Found` status code all requests on resource routes it does not implement.

A SERVER MUST be able to provide data for at least one of the following resources routes:

- `/2022-04/events`
- `/2022-04/mountainAreas`
- `/2022-04/lifts`
- `/2022-04/skiSlopes`
- `/2022-04/snowparks`

Throughout the next sections' routes tables, the ᴹ identifies mandatory features on GET requests on related route. Other listed features that have no appended symbols are either recommended or optional for SERVERS to implement.

5.3.1. Agent Routes

Agent routes are those used to create, retrieve, update, and delete Agent resources, as well as to retrieve resources they are related to.

Route	Methods	Description	Additional GET features
/2022-04/agents	GET, POST	Returns a collection of all Agent resources	Pagination^{M}, Sparse Fieldsets, Inclusion, Sorting, Random Sorting, Filtering, Searching
/2022-04/agents/:id	GET, PATCH, DELETE	Returns the Agent resource identified by the `:id` parameter	Sparse Fieldsets, Inclusion
/2022-04/agents/:id /categories	GET	Returns the collection of Category resources that classify the agent	Pagination, Sparse Fieldsets, Inclusion, Sorting, Random Sorting, Filtering, Searching
/2022-04/agents/:id /multimediaDescriptions	GET	Returns the collection of Media Object resources that are multimedia descriptions (e.g., images or videos) of the agent	Pagination, Sparse Fieldsets, Inclusion, Sorting, Random Sorting, Filtering, Searching

5.3.2. Category Routes

Category routes are those used to create, retrieve, update, and delete Category resources, as well as to retrieve resources they are related to.

Route	Methods	Description	Additional GET features
/2022-04/categories	GET, POST	Returns the collection of all Category resources	Pagination^M, Sparse Fieldsets, Inclusion, Sorting, Random Sorting, Filtering, Searching
/2022-04/categories/:id	GET, PATCH, DELETE	Returns the Category resource identified by the `id` parameter	Sparse Fieldsets, Inclusion
/2022-04/categories/:id /children	GET	Returns the collection of Category resources that are children of the category	Pagination, Sparse Fieldsets, Inclusion, Sorting, Random Sorting, Filtering, Searching
/2022-04/categories/:id /multimediaDescriptions	GET	Returns the collection of Media Object resources that are multimedia descriptions (e.g., images or videos) of the category	Pagination, Sparse Fieldsets, Inclusion, Sorting, Random Sorting, Filtering, Searching
/2022-04/categories/:id /parents	GET	Returns the collection of Category resources that are parents of the category	Pagination, Sparse Fieldsets, Inclusion, Sorting, Random Sorting, Filtering, Searching

5.3.3. Event Routes

Event routes are those used to create, retrieve, update, and delete Event resource, as well as to retrieve resources they are related to. \

Route	Methods	Description	Additional GET features
/2022-04/events	GET, POST	Returns the collection of all Event resources	Pagination^M, Sparse Fieldsets, Inclusion, Sorting, Random Sorting, Filtering, Searching
/2022-04/events/:id	GET, PATCH, DELETE	Returns the Event resource identified by the `id` parameter	Sparse Fieldsets, Inclusion
/2022-04/events/:id /categories	GET	Returns the collection of Category resources that classify the event	Pagination, Sparse Fieldsets, Inclusion, Sorting, Random Sorting, Filtering, Searching

Route	Methods	Description	Additional GET features
/2022-04/events/:id /contributors	GET	Returns the collection of Agent resources that are contributors on the event	Pagination, Sparse Fieldsets, Inclusion, Sorting, Random Sorting, Filtering, Searching
/2022-04/events/:id /multimediaDescriptions	GET	Returns the collection of Media Object resources that are multimedia descriptions (e.g., images or videos) of the event	Pagination, Sparse Fieldsets, Inclusion, Sorting, Random Sorting, Filtering, Searching
/2022-04/events/:id /organizers	GET	Returns the collection of Agent resources that are organizers of the event	Pagination, Sparse Fieldsets, Inclusion, Sorting, Random Sorting, Filtering, Searching
/2022-04/events/:id /publisher	GET	Return the Agent resource that is the publisher of the event	Sparse Fieldsets, Inclusion
/2022-04/events/:id /series	GET	Return the Event Series resource that the event is an edition of	Sparse Fieldsets, Inclusion
/2022-04/events/:id /sponsors	GET	Returns the collection of Agent resources that are sponsors of the event	Pagination, Sparse Fieldsets, Inclusion, Sorting, Random Sorting, Filtering, Searching
/2022-04/events/:id /subEvents	GET	Returns the collection of Event resources that are part of the event	Pagination, Sparse Fieldsets, Inclusion, Sorting, Random Sorting, Filtering, Searching
/2022-04/events/:id /venues	GET	Returns the collection of Venue resources where the event will happen	Pagination, Sparse Fieldsets, Inclusion, Sorting, Random Sorting, Filtering, Searching

5.3.4. Event Series Routes

Event Series routes are those used to create, retrieve, update, and delete Event Series resources, as well as to retrieve resources they are related to.

Route	Methods	Description	Additional GET features
/2022-04/eventSeries	GET, POST	Returns the collection of all Event Series resources	Pagination^M, Sparse Fieldsets, Inclusion, Sorting, Random Sorting, Filtering, Searching
/2022-04/eventSeries/:id	GET, PATCH, DELETE	Returns the Event Series resource identified by the `id` parameter	Sparse Fieldsets, Inclusion
/2022-04/eventSeries/:id /categories	GET	Returns the collection of Category resources that classify the event series	Pagination, Sparse Fieldsets, Inclusion, Sorting, Random Sorting, Filtering, Searching
/2022-04/eventSeries/:id /editions	GET	Returns the collection of Event resources that are editions of the event series	Pagination, Sparse Fieldsets, Inclusion, Sorting, Random Sorting, Filtering, Searching
/2022-04/eventSeries/:id /multimediaDescriptions	GET	Returns the collection of Media Object resources that are multimedia descriptions (e.g., images or videos) of the event series	Pagination, Sparse Fieldsets, Inclusion, Sorting, Random Sorting, Filtering, Searching

5.3.5. Feature Routes

Feature routes are those used to create, retrieve, update, and delete Feature resources, as well as to retrieve resources they are related to.

Route	Methods	Description	Additional GET features
/2022-04/features	GET, POST	Returns the collection of all Feature resources	Pagination^M, Sparse Fieldsets, Inclusion, Sorting, Random Sorting, Filtering, Searching
/2022-04/features/:id	GET, PATCH, DELETE	Returns the Feature resource identified by the `id` parameter	Sparse Fieldsets, Inclusion
/2022-04/features/:id /children	GET	Returns the collection of Feature resources that are children of the feature	Pagination, Sparse Fieldsets, Inclusion, Sorting, Random Sorting, Filtering, Searching

Route	Methods	Description	Additional GET features
/2022-04/features/:id /multimediaDescriptions	GET	Returns the collection of Media Object resources that are multimedia descriptions (e.g., images or videos) of the feature	Pagination, Sparse Fieldsets, Inclusion, Sorting, Random Sorting, Filtering, Searching
/2022-04/features/:id /parents	GET	Returns the collection of Feature resources that are parents of the feature	Pagination, Sparse Fieldsets, Inclusion, Sorting, Random Sorting, Filtering, Searching

5.3.6. Lift Routes

Lift routes are those used to create, retrieve, update, and delete Lift resources, as well as to retrieve resources they are related to.

Route	Methods	Description	Additional GET features
/2022-04/lifts	GET, POST	Returns the collection of all Lift resources	Pagination^{M}, Sparse Fieldsets, Inclusion, Sorting, Random Sorting, Filtering, Searching
/2022-04/lifts/:id	GET, PATCH, DELETE	Returns the Lift resource identified by the `id` parameter	Sparse Fieldsets, Inclusion
/2022-04/lifts/:id /categories	GET	Returns the collection of Category resources that classify the lift	Pagination, Sparse Fieldsets, Inclusion, Sorting, Random Sorting, Filtering, Searching
/2022-04/lifts/:id /connections	GET	Returns the collection of Lift, Mountain Area, Ski Slope, and Snowpark resources that are physically accessible from the lift	Pagination, Sparse Fieldsets, Inclusion, Sorting, Random Sorting, Filtering, Searching
/2022-04/lifts/:id /multimediaDescriptions	GET	Returns the collection of Media Object resources that are multimedia descriptions (e.g., images or videos) of the lift	Pagination, Sparse Fieldsets, Inclusion, Sorting, Random Sorting, Filtering, Searching

5.3.7. Media Object Routes

Media Object routes are those used to create, retrieve, update, and delete Media Object resources, as well as to retrieve resources they are related to.

Route	Methods	Description	Additional GET features
/2022-04/mediaObjects	GET, POST	Returns the collection of all Media Object resources	Pagination^M, Sparse Fieldsets, Inclusion, Sorting, Random Sorting, Filtering, Searching
/2022-04/mediaObjects/:id	GET, PATCH, DELETE	Returns the Media Object resource identified by the `id` parameter	Sparse Fieldsets, Inclusion
/2022-04/mediaObjects/:id /categories	GET	Returns the collection of Category resources that classify the media object	Pagination, Sparse Fieldsets, Inclusion, Sorting, Random Sorting, Filtering, Searching
/2022-04/mediaObjects/:id /licenseHolder	GET	Return the Agent resource that is the license holder of the media object	Sparse Fieldsets, Inclusion

5.3.8. Mountain Area Routes

Mountain Area routes are those used to create, retrieve, update, and delete Mountain Area resources, as well as to retrieve resources they are related to.

Route	Methods	Description	Additional GET features
/2022-04/mountainAreas	GET, POST	Returns the collection of all Mountain Area resources	Pagination^M, Sparse Fieldsets, Inclusion, Sorting, Random Sorting, Filtering, Searching
/2022-04/mountainAreas/:id	GET, PATCH, DELETE	Returns the Mountain Area resource identified by the `id` parameter	Sparse Fieldsets, Inclusion
/2022-04/mountainAreas/:id /areaOwner	GET	Return the Agent resource that is the owner of the mountain area	Sparse Fieldsets, Inclusion
/2022-04/mountainAreas/:id /categories	GET	Returns the collection of Category resources that classify the mountain area	Pagination, Sparse Fieldsets, Inclusion, Sorting, Random Sorting, Filtering, Searching

Route	Methods	Description	Additional GET features
/2022-04/mountainAreas/:id /connections	GET	Returns the collection of Lift, Mountain Area, Ski Slope, and Snowpark resources that are physically accessible from the mountain area	Pagination, Sparse Fieldsets, Inclusion, Sorting, Random Sorting, Filtering, Searching
/2022-04/mountainAreas/:id /lifts	GET	Returns the collection of Lift resources that are located within the mountain area	Pagination, Sparse Fieldsets, Inclusion, Sorting, Random Sorting, Filtering, Searching
/2022-04/mountainAreas/:id /multimediaDescriptions	GET	Returns the collection of Media Object resources that are multimedia descriptions (e.g., images or videos) of the mountain area	Pagination, Sparse Fieldsets, Inclusion, Sorting, Random Sorting, Filtering, Searching
/2022-04/mountainAreas/:id /skiSlopes	GET	Returns the collection of Ski Slope resources that are located within the mountain area	Pagination, Sparse Fieldsets, Inclusion, Sorting, Random Sorting, Filtering, Searching
/2022-04/mountainAreas/:id /snowparks	GET	Returns the collection of Snowpark resources that are located within the mountain area	Pagination, Sparse Fieldsets, Inclusion, Sorting, Random Sorting, Filtering, Searching
/2022-04/mountainAreas/:id /subAreas	GET	Returns the collection of Mountain Area resources that are located within the mountain area	Pagination, Sparse Fieldsets, Inclusion, Sorting, Random Sorting, Filtering, Searching

5.3.9. Ski Slope Routes

Ski Slope routes are those used to create, retrieve, update, and delete Ski Slope resources, as well as to retrieve resources they are related to.

Route	Methods	Description	Additional GET features
/2022-04/skiSlopes	GET, POST	Returns the collection of all Ski Slope resources	Pagination^{M}, Sparse Fieldsets, Inclusion, Sorting, Random Sorting, Filtering, Searching

Route	Methods	Description	Additional GET features
/2022-04/skiSlopes/:id	GET, PATCH, DELETE	Returns the Ski Slope resource identified by the `id` parameter	Sparse Fieldsets, Inclusion
/2022-04/skiSlopes/:id /categories	GET	Returns the collection of Category resources that classify the ski slope	Pagination, Sparse Fieldsets, Inclusion, Sorting, Random Sorting, Filtering, Searching
/2022-04/skiSlopes/:id /connections	GET	Returns the collection of Lift, Mountain Area, Ski Slope, and Snowpark resources that are physically accessible from the ski slope	Pagination, Sparse Fieldsets, Inclusion, Sorting, Random Sorting, Filtering, Searching
/2022-04/skiSlopes/:id /multimediaDescriptions	GET	Returns the collection of Media Object resources that are multimedia descriptions (e.g., images or videos) of the ski slope	Pagination, Sparse Fieldsets, Inclusion, Sorting, Random Sorting, Filtering, Searching

5.3.10. Snowpark Routes

Snowpark routes are those used to create, retrieve, update, and delete Snowpark resources, as well as to retrieve resources they are related to.

Route	Methods	Description	Additional GET features
/2022-04/snowparks	GET, POST	Returns the collection of all Snowpark resources	Pagination^{M}, Sparse Fieldsets, Inclusion, Sorting, Random Sorting, Filtering, Searching
/2022-04/snowparks/:id	GET, PATCH, DELETE	Returns the Snowpark resource identified by the `id` parameter	Sparse Fieldsets, Inclusion
/2022-04/snowparks/:id /categories	GET	Returns the collection of Category resources that classify the snowpark	Pagination, Sparse Fieldsets, Inclusion, Sorting, Random Sorting, Filtering, Searching
/2022-04/snowparks/:id /connections	GET	Returns the collection of Lift, Mountain Area, Ski Slope, and Snowpark resources that are physically accessible from the snowpark	Pagination, Sparse Fieldsets, Inclusion, Sorting, Random Sorting, Filtering, Searching

Route	Methods	Description	Additional GET features
/2022-04/snowparks/:id /features	GET	Returns the collection of Feature resources that are present in the snowpark	Pagination, Sparse Fieldsets, Inclusion, Sorting, Random Sorting, Filtering, Searching
/2022-04/snowparks/:id /multimediaDescriptions	GET	Returns the collection of Media Object resources that are multimedia descriptions (e.g., images or videos) of the snowpark	Pagination, Sparse Fieldsets, Inclusion, Sorting, Random Sorting, Filtering, Searching

5.3.11. Venue Routes

Venue routes are those used to create, retrieve, update, and delete Venue resources, as well as to retrieve resources they are related to.

Route	Methods	Description	Additional GET features
/2022-04/venues	GET, POST	Returns the collection of all Venue resources	Pagination^{M}, Sparse Fieldsets, Inclusion, Sorting, Random Sorting, Filtering, Searching
/2022-04/venues/:id	GET, PATCH, DELETE	Returns the Venue resource identified by the `id` parameter	Sparse Fieldsets, Inclusion
/2022-04/venues/:id /categories	GET	Returns the collection of Category resources that classify the venue	Pagination, Sparse Fieldsets, Inclusion, Sorting, Random Sorting, Filtering, Searching
/2022-04/venues/:id /multimediaDescriptions	GET	Returns the collection of Media Object resources that are multimedia descriptions (e.g., images or videos) of the venue	Pagination, Sparse Fieldsets, Inclusion, Sorting, Random Sorting, Filtering, Searching

6. Datatypes

6.1. address

An address is represented as an object containing the following fields:

- `street`: a text object containing the street name and number.
  For example: `{ "ita": "Via Bolzano 63/A", "deu": "Bozner Straße 63/A" }`
- `city`: a text object containing the city name.
  For example: `{ "ita": "Appiano sulla Strada del Vino", "deu": "Eppan an der Weinstraße" }`
- `region`: a text object containing the administrative unit in which the city is located (e.g. province, region, state). For example: `{ "ita": "Bolzano", "deu": "Bozen" }`
- `country`: a string containing a 2-letter code that identifies the country as defined in the ISO 3166-1. For example: `"IT"` identifies Italy.
- `zipcode`: a string containing the zip code. For example: `"39057"`
- `complement`: a text object containing additional information about the address, such as the building name, the floor, and/or the room number. For example: `{ "ita": "2 piano, sala 1.10", "deu": "2 etage, raum 1.10" }`
- `type`: a string containing the type of the address. For example, `"billing"`. The standard does not standardize types of addresses.

Every address object MUST have defined values for at least the `city` and `country` fields.

Here is an example of a minimum address object:

```
{
  "street": null,
  "city": {
    "ita": "Appiano sulla Strada del Vino",
    "deu": "Eppan an der Weinstraße"
  },
  "region": null,
  "country": "IT",
  "zipcode": null,
  "complement": null,
  "type": null
}
```

And here is an example of a complete address object:

```
{
  "street": {
    "ita": "Via Bolzano 63/A",
    "deu": "Bozner Straße 63/A"
  },
  "city": {
    "ita": "Appiano sulla Strada del Vino",
    "deu": "Eppan an der Weinstraße"
  },
  "region": {
    "ita": "Bolzano",
    "deu": "Bozen"
  },
  "country": "IT",
  "zipcode": "39057",
  "complement": {
    "ita": "2 piano, sala 1.10",
    "deu": "2 etage, raum 1.10"
  },
  "type": "billing"
}
```

6.2. contact point

A contact point contains data that one can use to contact an Agent. It is described by an object containing the following fields:

- `email`: an email defined for the contact point. Nullable.
  For example, "`info@alpinebits.org`".
- `telephone`: a string containing a telephone, including the country code and without spaces, defined for the contact point. Nullable.
  For example: "`+39000000000000`".
- `address`: an address defined for the contact point. Nullable.
- `availableHours`: an array of hours specification objects identifying the hours in which the Agent can be contacted through the means defined in the contact point. Nullable.

Every contact point object MUST define at least one of the following fields: `email`, `telephone`, `address`.

The three objects below illustrate the minimum information that may be defined within a contact point:

Contact point containing only an email address:

```
{
  "email": "info@alpinebits.org",
  "telephone": null,
  "address": null,
  "availableHours": null
}
```

Contact point containing only a telephone:

```
{
  "email": null,
  "telephone": "+39000000000000",
  "address": null,
  "availableHours": null
}
```

Contact point containing only an address:

```
{
  "email": null,
  "telephone": null,
  "address": {
    "street": null,
    "city": {
      "ita": "Appiano sulla Strada del Vino",
      "deu": "Eppan an der Weinstraße"
    },
    "region": null,
    "country": "IT",
    "zipcode": null
  },
  "availableHours": null
}
```

The object below illustrates the usage of all fields of a contact point object:

```
{
  "email": "info@alpinebits.org",
  "telephone": "+39 0471 066 600",
  "address": {
    "street": {
      "ita": "Via Bolzano 63/A",
      "deu": "Bozner Straße 63/A"
    },
    "city": {
      "ita": "Appiano sulla Strada del Vino",
      "deu": "Eppan an der Weinstraße"
    },
    "region": {
      "ita": "Bolzano",
      "deu": "Bozen"
    },
    "country": "IT",
    "zipcode": "39057"
  },
  "availableHours": {
    "dailySchedules": null,
    "weeklyScheules": [
      {
        "validFrom": "2020-01-01",
        "validTo": "2020-12-31",
        "monday": [
          { "opens": "08:00:00", "closes": "12:00:00" },
          { "opens": "14:00:00", "closes": "18:00:00" }
        ],
        "tuesday": [
          { "opens": "08:00:00", "closes": "12:00:00" },
          { "opens": "14:00:00", "closes": "18:00:00" }
        ],
        "wednesday": [
          { "opens": "08:00:00", "closes": "12:00:00" },
          { "opens": "14:00:00", "closes": "18:00:00" }
        ],
        "thursday": [
          { "opens": "08:00:00", "closes": "12:00:00" },
          { "opens": "14:00:00", "closes": "18:00:00" }
        ],
        "friday": [
          { "opens": "08:00:00", "closes": "12:00:00" },
          { "opens": "14:00:00", "closes": "18:00:00" }
        ],
        "saturday": null,
        "sunday": null
      }
    ]
  }
}
```

Note that the standard does not allow multiple emails, telephones, and addresses to be defined in a single contact point object. To exchange such data, multiple contact objects should be used.

6.3. date

A string that contains a year, a month, and a day, formatted as YYYY-MM-DD. Corresponds to *full-date* in RFC 3339.
Here is an example:

```
"2020-04-01"
```

6.4. date-time

A string that contains a date, a time, and a time offset (to account for time zones) and formatted as YYYY-MM-DDThh:mm:ss+hh:mm (or YYYY-MM-DDThh:mm:ss-hh:mm). Corresponds to *date-time* in RFC 3339.

The date-time that refers to April 1st, 2020 at 23 hours, 59 minutes, and 59 seconds according to the Central European Time is:

```
"2020-04-01T23:59:59+04:00"
```

6.5. email

A string containing a valid internet email address, as defined in RFC 5322, section 3.4.1.

Here is an example of a valid email:

```
info@alpinebits.org
```

6.6. geometry

The representation of geographic data structures in this standard is borrowed from the GeoJSON standard specification.

In particular, this standard adopts GeoJSON's definition of Geometry Object, which represents points, curves, and surfaces in coordinate space.

A generic geometry object contains the following fields:

- `type`: a string identifying the geometry type, whose possible values are `"Point"`, `"MultiPoint"`, `"LineString"`, `"MultiLineString"`, `"Polygon"`, and `"MultiPolygon"`. Non-nullable.
- `coordinates`: an array containing the coordinates of the geometry. The structure of the members of this array is determined by the type of geometry. Non-nullable.

Definitions for each geometry type are given in the subsequent sections, being directly extracted from the GeoJSON standard specification.

6.6.1. point

For type `Point`, the `coordinates` member is a single position.

An example of a geometry object of type `Point` is provided below:

```
{
  "type": "Point",
  "coordinates": [
    11.358447074890137,
    46.49667880447103
  ]
}
```

6.6.2. multi-point

For type `MultiPoint`, the `coordinates` member is an array of positions.

An example of a geometry object of type `MultiPoint` is provided below:

```
{
    "type": "MultiPoint",
    "coordinates": [
        [
            11.358447074890137,
            46.49667880447103
        ],
        [
            11.119945049285889,
            46.07315619530213
        ]
    ]
}
```

6.6.3. line-string

For type `LineString`, the `coordinates` member is an array of two or more positions.

An example of a geometry object of type `LineString` is provided below:

```
{
    "type": "LineString",
    "coordinates": [
        [
            11.358447074890137,
            46.49667880447103
        ],
        [
            11.119945049285889,
            46.07315619530213
        ]
    ]
}
```

6.6.4. multi-line-string

For type `MultiLineString`, the `coordinates` member is an array of LineString coordinate arrays.

An example of a geometry object of type `MultiLineString` is provided below:

```
{
    "type": "MultiLineString",
    "coordinates": [
        [
            [
                11.358447074890137,
                46.49667880447103
            ],
            [
                11.119945049285889,
                46.07315619530213
            ]
        ],
        [
            [
                11.149181127548218,
                46.673201221323815
            ],
            [
                11.40104055404663,
                47.263329885918694
            ]
        ]
    ]
}
```

6.6.5. polygon

To specify a constraint specific to the `Polygon` type, it is useful to introduce the concept of a linear ring:

- A linear ring is a closed LineString with four or more positions.
- The first and last positions are equivalent, and they MUST contain identical values; their representation SHOULD also be identical.
- A linear ring is the boundary of a surface or the boundary of a hole in a surface.
- A linear ring MUST follow the right-hand rule with respect to the area it bounds, i.e., exterior rings are counterclockwise, and holes are clockwise.

Though a linear ring is not explicitly represented as a GeoJSON geometry type, it leads to a canonical formulation of the Polygon geometry type definition as follows:

- For type `Polygon`, the `coordinates` member MUST be an array of linear ring coordinate arrays.
- For Polygons with more than one of these rings, the first MUST be the exterior ring, and any others MUST be interior rings. The exterior ring bounds the surface, and the interior rings (if present) bound holes within the surface.

An example of a geometry object of type `Polygon` is provided below:

```
{
  "type": "Polygon",
  "coordinates": [
    [
      [
        11.349220275878906,
        46.49902740460012
      ],
      [
        11.349080801010132,
        46.497631550714374
      ],
      [
        11.350893974304197,
        46.49759462287638
      ],
      [
        11.351033449172974,
        46.49890185307167
      ],
      [
        11.349220275878906,
        46.49902740460012
      ]
    ]
  ]
}
```

6.6.6. multi-polygon

For type `MultiPolygon`, the `coordinates` member is an array of Polygon coordinate arrays.

An example of a geometry object of type `MultiPolygon` is provided below:

```
{
  "type": "MultiPolygon",
  "coordinates": [
    [
      [
        [
          11.349220275878906,
          46.49902740460012
        ],
        [
          11.349080801010132,
          46.497631550714374
        ],
        [
          11.350893974304197,
          46.49759462287638
        ],
        [
          11.351033449172974,
          46.49890185307167
        ],
        [
          11.349220275878906,
          46.49902740460012
        ]
      ]
    ],
    [
      [
        [
          11.352425515651703,
          46.49758354452009
        ],
        [
          11.352377235889433,
          46.49728812085311
        ],
        [
          11.352744698524473,
          46.49723642154639
        ],
        [
          11.352425515651703,
          46.49758354452009
        ]
      ]
    ]
  ]
}
```

6.7. hours specification

An hours specification object allows the representations of schedules over periods of time. They can be used to specify when a person or organization can be contacted, when a place is open, or when a recurrent event will occur. Hours specification objects may contain daily and weekly schedules, which in turn contain arrays of opening and closing hours for a given day.

Each opening/closing interval is represented by an object containing an `opens` and a `closes` time strings. For example, the interval from 9am to 5pm may be represented as `{ "opens": "09:00:00", "closes": "17:00:00" }`. For a single day, an array of these objects represents the daily opening schedule. For example, the working hours of an office may be represented by an array with two object spaced by a lunch break, as in the following case:

```
[
  { "opens": "09:00:00", "closes": "12:00:00" },
  { "opens": "14:00:00", "closes": "17:00:00" }
]
```

The hours specification object represents daily and weekly schedules by using the aforementioned arrays to describe the schedules of each date. For that, the hours specification object MUST contain two nullable fields, `dailySchedules` and `weeklySchedules` that respectively contain daily and weekly schedules.

The `dailySchedules` field is a nullable non-empty object whose fields are date strings and represent the schedule for the listed dates. Each date field is a nullable non-empty array of opening interval objects. Each date set to `null` represents that no opening interval exists for the given date (i.e., represents a closed date). For example, a 5-days conference containing a skipped date between satellite and main events maybe be represented as follows:

```
{
  "dailySchedules": {
    "2020-04-06": [ { "opens": "08:00:00", "closes": "18:00:00" } ],
    "2020-04-07": [ { "opens": "08:00:00", "closes": "18:00:00" } ],
    "2020-04-08": null,
    "2020-04-09": [ { "opens": "08:00:00", "closes": "18:00:00" } ],
    "2020-04-10": [ { "opens": "08:00:00", "closes": "18:00:00" } ],
  },
  "weeklySchedules": null
}
```

The `weeklySchedules` field is a nullable non-empty array of objects representing recurrent weekly schedules over defined periods of time. Each object within the `weeklySchedules` array MUST contain the following fields:

- `validFrom`: a non-nullable date string that represents the first valid day of the weekly schedule.
- `validTo`: a non-nullable date string that represents the last valid day of the weekly schedule.
- `sunday`: a nullable non-empty array of opening interval objects that represents the schedule for every Sunday within the validity of the weekly schedule.
- `monday`: a nullable non-empty array of opening interval objects that represents the schedule for every Monday within the validity of the weekly schedule.
- `tuesday`: a nullable non-empty array of opening interval objects that represents the schedule for every Tuesday within the validity of the weekly schedule.
- `wednesday`: a nullable non-empty array of opening interval objects that represents the schedule for every Wednesday within the validity of the weekly schedule.
- `thursday`: a nullable non-empty array of opening interval objects that represents the schedule for every Thursday within the validity of the weekly schedule.
- `friday`: a nullable non-empty array of opening interval objects that represents the schedule for every Friday within the validity of the weekly schedule.
- `saturday`: a nullable non-empty array of opening interval objects that represents the schedule for every Saturday within the validity of the weekly schedule.

The `weeklySchedules` may represent, for example, a shopping mall that opens beyond regular hours on Saturdays but closes on Sundays, as follows:

```
{
  "dailySchedules": null,
  "weeklySchedules": [
    {
      "validFrom": "2020-01-01",
      "validTo": "2020-12-31",
      "monday": [ { "opens": "08:00:00", "closes": "20:00:00" } ],
      "tuesday": [ { "opens": "08:00:00", "closes": "20:00:00" } ],
      "wednesday": [ { "opens": "08:00:00", "closes": "20:00:00" } ],
      "thursday": [ { "opens": "08:00:00", "closes": "20:00:00" } ],
      "friday": [ { "opens": "08:00:00", "closes": "20:00:00" } ],
      "saturday": [ { "opens": "08:00:00", "closes": "22:00:00" } ],
      "sunday": null
    }
  ],
}
```

The following constraints apply to hours specification objects:

- within a single date, opening interval objects MUST NOT overlap
- within the `weeklySchedules` array, the validity window (between `validFrom` to `validTo`) of any two objects MUST NOT overlap
- schedules within `dailySchedules` have precedence over schedules within `weeklySchedules` for the same date
- the closing time of an opening interval object MAY go beyond midnight into the following day

To exemplify the combination daily and weekly schedules, we can exceptions to the shopping mall example, adding that it could be closed in during the first week of the year and that it includes exceptional schedules during the holidays.

```
{
  "dailySchedules": {
    "2020-12-24": [ { "opens": "08:00:00", "closes": "02:00:00" } ],
    "2020-12-25": [ { "opens": "08:00:00", "closes": "16:00:00" } ],
    "2020-12-26": null,
  },
  "weeklySchedules": [
    {
      "validFrom": "2020-01-01",
      "validTo": "2020-01-07",
      "monday": null,
      "tuesday": null,
      "wednesday": null,
      "thursday": null,
      "friday": null,
      "saturday": null,
      "sunday": null
    },
    {
      "validFrom": "2020-01-08",
      "validTo": "2020-12-31",
      "monday": [ { "opens": "08:00:00", "closes": "20:00:00" } ],
      "tuesday": [ { "opens": "08:00:00", "closes": "20:00:00" } ],
      "wednesday": [ { "opens": "08:00:00", "closes": "20:00:00" } ],
      "thursday": [ { "opens": "08:00:00", "closes": "20:00:00" } ],
      "friday": [ { "opens": "08:00:00", "closes": "20:00:00" } ],
      "saturday": [ { "opens": "08:00:00", "closes": "22:00:00" } ],
      "sunday": null
    }
  ],
}
```

6.8. snow condition

A snow condition object allows the representation of the conditions of a ski slope, a snowpark, or a mountain area at a given point in time.

The object is described by the following fields:

- `baseSnow`: a number representing the depth of base snow in a location. Measured in centimeters. Non-nullable.
- `baseSnowRange`: an object that identifies the variation in the depth of snow in a location. Nullable. This object contains two fields:
    - `lower`: a number that identifies the smallest depth of snow found in a location. Measured in centimeters. Non-nullable.
    - `upper`: a number that identifies the biggest depth of snow found in a location. Measured in centimeters. Non-nullable.
- `groomed`: a boolean flag indicating whether or not the snow in a location has been groomed. Nullable.
- `latestStorm`: a number representing the amount of snow from the latest storm cycle in a location. Measured in centimeters. Nullable.
- `obtainedIn`: a date or date-time string that identifies when the data contained in the snow condition objects was obtained. Nullable.
- `primarySurface`: a string that indicates the primary type of snow found in a location. Non-nullable. It is RECOMMENDED the usage of snow types defined in MTN.XML: `"packed-powder"`, `"powder"`, `"hard-pack"`, `"loose-granular"`, `"frozen-granular""`, `"wet-packed"`, `"wet-granular"`, `"wet-snow"`, `"spring-conditions"`, `"windblown"`, `"corn-snow"`, `"icy"`, `"variable"`
- `secondarySurface`: a string that indicates the secondary type of snow found in a location. Nullable. Its recommended values are those listed for `primarySurface`.
- `snowMaking`: a boolean flag indicating whether or not the snow in a location has been artificially produced. Nullable.
- `snowOverNight`: a number representing the amount of snow accumulated in a location throughout the night before the measurement. Measured in centimeters. Nullable.

The following example illustrates the minimal information required for a snow condition object:

```
{
  "obtainedIn": null,
  "primarySurface": "powder",
  "secondarySurface": null,
  "baseSnow": 50,
  "baseSnowRange": null,
  "latestStorm": null,
  "snowOverNight": null,
  "groomed": null,
  "snowMaking": null
}
```

The following example illustrates all fields of snow condition object:

```
{
  "obtainedIn": "2020-01-18T08:00:00+04:00",
  "primarySurface": "powder",
  "secondarySurface": "packed-powder",
  "baseSnow": 50,
  "baseSnowRange": {
    "lower": 40,
    "upper": 60
  },
  "latestStorm": 40,
  "snowOverNight": 5,
  "groomed": true,
  "snowMaking": false
}
```

6.9. text

Every field used to exchange textual data (e.g. name, description) supports the use of multiple languages. Such fields have, like their values, objects containing language codes as keys (e.g. `eng`, `ita`, `deu`), and the actual text in each language as values.

The example below illustrates how to represent the field `name` in German, English, and Italian.

```
{
  ...
  "name": {
    "deu": "Mein name",
    "eng": "My Name",
    "ita": "Il mio nome"
  },
  ...
}
```

The language codes used in the aforementioned objects are those defined in ISO 639-3, which can be found at https://iso639-3.sil.org/code_tables/639/.

The table below contains some examples of languages and their respective ISO 639-3 codes:

Language	ISO-639-3
German	`deu`
English	`eng`
Italian	`ita`
Ladin	`lld`
French	`fra`
Chinese	`zho`
Spanish	`spa`

Note that a language-specific field should only be added if there is available data. For instance, if only the English version of a resource’s name is available, it should be represented as:

```
{
  ...
  "name": {
    "eng": "My Name"
  },
  ...
}
```

6.10. time

A string that contains hours, minutes, and seconds, formatted as hh:mm:ss. Corresponds to *full-time* in RFC 3339.

Here is an example:

```
"23:59:59"
```

6.11. url

A string containing a valid universal resource identifier (URI), as defined in RFC 3986.

Here are some valid examples provided in the RFC 3986:

```
ftp://ftp.is.co.za/rfc/rfc1808.txt

http://www.ietf.org/rfc/rfc2396.txt

ldap://[2001:db8::7]/c=GB?objectClass?one

mailto:John.Doe@example.com

news:comp.infosystems.www.servers.unix

tel:+1-816-555-1212

telnet://192.0.2.16:80/

urn:oasis:names:specification:docbook:dtd:xml:4.1.2
```

7. Resources

7.1. Basic Resource Schema

7.1.1. Basic Fields

Every resource MUST have the following fields:

- `type`: a string that identifies the resource’s type within an enumeration of possible values. Non-nullable. Enumeration.
  Its possible values are `"agents"`, `"events"`, `"eventSeries"`, `"lifts"`, `"mediaObjects"`, `"mountainAreas"`, `"snowparks"`, `"skiSlopes"`, and `"venues"`.
- `id`: a string that uniquely and persistently identifies the resource within a SERVER. Non-nullable. Non-empty. Unique.
- `attributes`: an object containing the resource’s data that does not refer to other resources (e.g. the name of an event, the length of a slope).
- `relationships`: an object containing the resource’s data referring to other resources (e.g. the organizers of an event, the slopes within a mountain area).

In addition, resources MAY have the following fields in certain contexts (see Requests and Responses):

- `meta`: an object containing metadata of the resource (e.g., the URL representing the resource’s data provider or a date-time string of the instant of the resource’s last update). Non-empty.

  The `meta` object MAY NOT be present in the resource of creation and update requests when the data provider is not required by the server.

- `links`: an object containing the links related to access related resources. Examples include, a link to the agent resources representing organizers and sponsors of an event, a link to the media object resources representing multimedia descriptions of a mountain area resource, or a link to the resource itself, also referred to as `self` (see Messages and Hypermedia Controls). Non-empty.

  The `meta` object SHALL NOT be present in the resource of creation and update requests.

In the following example, we present an event resource (i.e. a resource of type `events`) might be represented as follows:

```
{
  "type": "events",
  "id": "1",
  "meta": {
    "lastUpdate": "2019-11-05T08:15:30-05:00",
    ...
  },
  "attributes": {
    "name": {
      "eng": "My event",
      "ita": "Il mio evento"
    },
    "startDate": "2020-04-01T09:30:00+00:00",
    ...
  },
  "relationships": {
    "organizers": {
      "data": [
        {
          "type": "agents",
          "id": "1"
        }
      ],
      "links": {
        "related": "https://example.com/2022-04/events/1/organizer"
      }
    },
    ...
  },
  "links": {
    "self": "https://example.com/2022-04/events/1"
  }
}
```

Note that each reference to another resource within the `data` in the `relationships` object consists only of a `type` and an `id`. The `relationships` object SHALL NOT contain the actual data of these resources, but only their references. We refer to the section Relationships Object for the mechanisms on how to access the actual referenced resource.

Each resource type defines a set of fields that must be present in each of these objects. However, a specific SERVER, when implementing the **AlpineBits® DestinationData**, MAY include additional fields, but it MUST comply to the JSON:API v1.0 standard (e.g., additional metadata fields within the `meta` object, or additional resource's attributes within the `attributes` object). It is also RECOMMENDED that the SERVER follow the style of the **AlpineBits® DestinationData** standard when including additional fields.

7.1.2. Meta Object

A `meta` object MUST contain the following fields:

- `lastUpdate`: a date-time that identifies when a resource was last modified in the SERVER. If no modifications were made after the resource's creation, the value in this field will be the creation date. Nullable.
- `dataProvider`: a url that identifies the organization responsible for provide the resource's data to the SERVER. The data provider field is intended to offer basic data traceability, for example, allowing users to contact providers regarding questions or issues with a resource's data. The data provider field should not be interpreted as the data author, as these two may differ. The data provider may be the the organization providing the SERVER itself. Non-nullable.

  For example, the URL `"http://tourism.opendatahub.bz.it/"` would be used to represent that a resource has been modified by the OpenDataHub platform.

An example of `meta` object is presented below:

```
{
  "dataProvider": "http://tourism.opendatahub.bz.it/",
  "lastUpdate": "2019-05-01T08:15:30-05:00"
}
```

7.1.3. Attributes Object

The `attributes` object of every resource in this specification MUST contain the following fields:

- `abstract`: a text object containing a short textual description of the resource. The size of an abstract in any given language SHOULD be at most 200 characters long. This field SHOULD NOT contain the same data as in the `description` field. Nullable.
- `description`: a text object containing the complete textual description of the resource. If `abstract` is not null, this field MUST NOT be null.
- `name`: a text object containing the complete name of the resource. This field MUST NOT be null if `shortName` is not null or if otherwise specified for by a particular resource type.
- `shortName`: a text object containing a short name of the resource. The size of a short name in any given language SHOULD be at most 36 characters long. This field SHOULD NOT contain the same data as in the `name` field. Nullable.
- `url`: a url string or a multilingual url object containing language-specific url strings. Nullable if not otherwise specified.

An example of a URL string:

```
"https://www.alpinebits.org/events/1"
```

An example of a multilingual URL object:

```
{
  "deu": "https://www.alpinebits.org/de/events/1",
  "eng": "https://www.alpinebits.org/en/events/1",
  "ita": "https://www.alpinebits.org/it/events/1"
}
```

The following object exemplifies the use of the basic attributes defined above for an event resource:

```
{
  "type": "events",
  "id": "1",
  "meta": { ... },
  "attributes": {
    "abstract": {
      "eng": "An event for those who want to implement AlpineBits at their companies.",
      "ita": "Un evento per coloro che vogliono implementare AlpineBits nelle loro
aziende."
    },
    "description": {
      "eng": "A 1-day workshop in which we present and explain the standard, give
implementation suggestions...",
      "ita": "Un workshop da 1 giorno in cui presentiamo e spieghiamo lo standard,
facciamo suggerimenti di implementazione..."
    },
    "name": {
      "eng": "AlpineBits DestinationData workshop for newcomers",
      "ita": "AlpineBits DestinationData workshop per i nuovi membri"
    },
    "shortName": {
      "eng": "DestinationData Workshop",
      "ita": "DestinationData Workshop"
    },
    "url": "https://www.alpinebits.org/events/1"
    ...
  },
  "relationships": { ... },
  "links": { ... }
}
```

The object below exemplifies an event which only has a name in English and no short name:

```
{
  "type": "events",
  "id": "1",
  "meta": { ... },
  "attributes": {
    "name": {
      "eng": "AlpineBits DestinationData workshop for newcomers."
    },
    "shortName": null,
    ...
  },
  "relationships": { ... },
  "links": { ... }
}
```

7.1.4. Relationships Object

The `relationships` object contains reference objects to one or many resources in every field allowing for resources to be connected through domain relationship in a modular way (see Relationship in JSON:API v1.0). Every relationship (i.e., relationship field) within the `relationships` object MUST contain either a reference object or `null`. Every resource type MUST define the relationships within the `relationships` object as either Reference to One objects or Reference to Many objects depending on the semantic of the relationship.

Reference to One

A reference object to one resource MUST contain the following fields:

- `data`: an object containing the `type` string and the `id` string of the referred resource.
- `links`: an object containing the `related` field with a string representing the endpoint on which the referred resource is available.

In case the resource does not support the endpoint of the related resource, the presence of the field `relates` is OPTIONAL.

For example, the Media Objects resource type defines a relationship `licenseHolder` (see Relationships) to the agent resource representing the media object's owner:

```
{
  "type": "mediaObjects",
  "id": "1",
  "meta": { ... },
  "attributes": { ... },
  "relationships": {
    "licenseHolder": {
      "data": {
        "type": "agents",
        "id": "1"
      },
      "links": {
        "related": "https://example.com/2022-04/mediaObjects/1/licenseHolder"
      }
    }
  },
  "links": { ... }
}
```

In case a resource has no related resource, the reference can be set to `null` according to the requirements defined in its resource type. For example, a media object resource that does not refer to an agent resource as its license holder can be presented as follows:

```
{
  "type": "mediaObjects",
  "id": "1",
  "meta": { ... },
  "attributes": { ... },
  "relationships": {
    "licenseHolder": null
  },
  "links": { ... }
}
```

Reference to Many

A reference object to many resources MUST contain the following fields:

- `"data"`: an array of objects containing the `type` string and the `id` string of each referred resource.
- `"links"`: an object containing the `related` field with a string representing the endpoint on which the referred resources are available.
  In case the resource does not support the endpoint of the related resource, the presence of the field `relates` is OPTIONAL.

For example, the event resource type defines a relationship `organizers` (see Relationships) to the agent resources representing the event's owner:

```
{
  "type": "events",
  "id": "1",
  "meta": { ... },
  "attributes": { ... },
  "relationships": {
    "organizers": {
      "data": [
        {
          "type": "agents",
          "id": "4"
        },
        {
          "type": "agents",
          "id": "1"
        },
        {
          "type": "agents",
          "id": "9"
        }
      ],
      "links": {
        "related": "https://example.com/2022-04/mediaObjects/1/licenseHolder"
      }
    },
    ...
  },
  "links": { ... }
}
```

In case a resource has no related resource, the reference can be set to `null` according to the requirements defined in its resource type. For example, a media object resource that does not refer to an agent resource as its license holder can be presented as follows:

```
{
  "type": "events",
  "id": "1",
  "meta": { ... },
  "attributes": { ... },
  "relationships": {
    "organizers": null,
    ...
  },
  "links": { ... }
}
```

Every resource, with the exception of instances of categories, features, and media objects, MUST contain the Reference to Many bellow within the `relationships` object. Of these, categories and features MUST contain the Reference to Many multimediaDescriptions, and media objects MUST contain the Reference to Many categories

- `categories`: a Reference to Many category resources identifying the categories instantiated by the resource. The values proposed by the standard for each resource type are listed in each resource section. Non-nullable. Non-empty.

- `multimediaDescriptions`: a Reference to Many media object resources that illustrates a resource. Media objects be images, videos, audio, data documents (e.g. csv, json), textual documents (e.g. doc, pdf), html documents, and so on.

Here is an example of how to use the `multimediaDescriptions` relationship:

```
{
  "type": "events",
  "id": "1",
  "attributes": { ... },
  "relationships": {
    "categories": {
      "data": [
        {
          "type": "categories",
          "id": "alpinebits:inPersonEvent"
        }
      ],
      "links": {
        "related": "https://example.com/2022-04/events/1/categories"
      }
    },
    "multimediaDescriptions": {
      "data": [
        {
          "type": "mediaObjects",
          "id": "1"
        },
        {
          "type": "mediaObjects",
          "id": "2"
        },
        {
          "type": "mediaObjects",
          "id": "3"
        },
      ],
      "links": {
        "related": "https://example.com/2022-04/events/1/multimediaDescriptions"
      }
    }
    ...
  },
  "links": { ... }
}
```

7.1.5. Links Object

The `links` object of instances of every resource type MUST contain the following fields:

- `self`: a string representing the endpoint on which the container resource is available.
  For example, an event resource may be presented as follows:

  ```
  {
    "type": "events",
    "id": "1",
    "attributes": { ... },
    "relationships": { ... },
    "links": {
      "self": "https://example.com/2022-04/events/1"
    }
  }
  ```

7.2. Resource Schemas

7.2.1. Agents

A resource that implements the concept of Agent defined in the **AlpineBits® DestinationData Ontology**.

A JSON object representing such a resource MUST contain the following fields:

- `type`: the constant `"agents"` that identifies the resource as being of the type agent.
- `id`: a string that uniquely and persistently identifies the agent within a SERVER. See the definition in Basic Fields.
- `attributes`: an object containing the attributes of the agent.
- `relationships`: an object containing the relationships of the agent to other resources.
- `links`: an object containing the links related to the agent.

Agent resources are structured in the following way:

```
{
  "type": "agents",
  "id": "1",
  "meta": { ... },
  "attributes": { ... },
  "relationships": { ... },
  "links": { ... }
}
```

Meta

See the definition of the `meta` object in Meta Object.

Attributes

The `attributes` object of the agent resource MUST contain the following fields:

- `abstract`: a text object containing a brief description for the agent. Nullable. See the definition in Attributes Object.
- `contactPoints`: an array of contact point objects. Nullable. Non-empty.
- `description`: a text object containing a description of the agent. Nullable. Conditional Assignment. See the definition in Attributes Object.
- `name`: a text object containing the complete name of the agent. Non-nullable. Conditional Assignment. See the definition in Attributes Object.
- `shortName`: a text object containing a short name of the agent. Nullable. See the definition in Attributes Object.
- `url`: a url object or string describing the agent, such as a website or a Wikipedia page. Nullable. See the definition in Attributes Object.

A summary of the attributes is presented in the table below:

Field	Type	Constraints
abstract	text	Nullable
contactPoints	Array of contact point	Nullable, Non-empty
description	text	Nullable, Conditional Assignment
name	url	Non-nullable, Conditional Assignment
shortName	text	Nullable
url	url	Nullable

Relationships

The `relationships` object of the agent resource MUST contain the following fields:

- `categories`: a Reference to Many category resources that are instantiated by the agent. See Section Categories. Nullable. Non-empty.

The standard defines the following disjoint categories for agents:

  - `"alpinebits:person"`: an agent resource representing a particular person. For instance, an artist who sings at a concert or a photographer who owns the license of a photo.
  - `"alpinebits:organization"`: an agent resource representing a public or a private organization. For instance, a company organizing an event or the owner of a mountain area.

- `multimediaDescriptions`: a Reference to Many media object resources (see Media Objects) that are related to the agent. See Section multimediaDescriptions. Nullable. Non-empty.

A summary of the relationships is presented in the table below:

Field	Type	Constraints
categories	Reference to Many object to Categories	Nullable, Non-empty
multimediaDescriptions	Reference to Many object to Media Objects	Nullable, Non-empty

Links

See the definition of the `links` object in Links Object.

Examples

The following example presents an object containing the minimal information required of an agent resource:

```
{
  "type": "agents",
  "id": "1",
  "meta": {
    "dataProvider": null,
    "lastUpdate": null
  },
  "attributes": {
    "abstract": null,
    "contactPoints": null,
    "description": null,
    "name": {
      "eng": "Free University of Bozen-Bolzano"
    },
    "shortName": null,
    "url": null
  },
  "relationships": {
    "categories": null,
    "multimediaDescriptions": null
  },
  "links": {
    "self": "https://example.com/2022-04/agents/1"
  }
}
```

asciidoc/examples/agent.min.json

The following example presents an object representing an agent resource:

```
{
  "type": "agents",
  "id": "1",
  "meta": {
    "dataProvider": "https://example.com",
    "lastUpdate": "2020-04-01T08:00:00+02:00"
  },
  "attributes": {
    "abstract": {
      "ita": "La Libera Università di Bolzano sorge in una delle più attraenti regioni 
europee...",
      "deu": "Die Freie Universität Bozen befindet sich in einer der attraktivsten 
Regionen Europas...",
      "eng": "The Free University of Bozen-Bolzano is located in one of the most 
fascinating European regions..."
    },
    "contactPoints": [
      {
        "email": "info@noi.bz.it",
        "telephone": "+39 0471 066 600",
        "address": {
          "street": {
            "ita": "Piazza Università 1"
          },
          "city": {
            "ita": "Bolzano"
          },
          "region": {
            "ita": "Trentino-Alto Adige"
          },
          "country": "IT",
          "zipcode": "39100",
          "complement": {
            "ita": "Ufficio 3.07"
          },
          "categories": [
            "example:main"
          ]
        },
        "availableHours": {
          "dailySchedules": {
            "2020-12-23": [
              {
                "opens": "08:00:00",
                "closes": "12:00:00"
              }
            ],
            "2020-12-25": null
          },
          "weeklySchedules": [
            {
              "validFrom": "2020-01-01",
              "validTo": "2020-12-31",
              "monday": [
                {
                  "opens": "08:00:00+01:00",
                  "closes": "12:00:00+01:00"
                },
                {
                  "opens": "14:00:00+01:00",
                  "closes": "18:00:00+01:00"
                }
              ],
              "tuesday": [
                {
                  "opens": "08:00:00+01:00",
                  "closes": "12:00:00+01:00"
                },
                {
                  "opens": "14:00:00+01:00",
                  "closes": "18:00:00+01:00"
                }
              ],
              "wednesday": [
                {
                  "opens": "08:00:00+01:00",
                  "closes": "12:00:00+01:00"
                },
                {
                  "opens": "14:00:00+01:00",
                  "closes": "18:00:00+01:00"
                }
              ],
              "thursday": [
                {
                  "opens": "08:00:00+01:00",
                  "closes": "12:00:00+01:00"
                },
                {
                  "opens": "14:00:00+01:00",
                  "closes": "18:00:00+01:00"
                }
              ],
              "friday": [
                {
                  "opens": "08:00:00+01:00",
                  "closes": "12:00:00+01:00"
                },
                {
                  "opens": "14:00:00+01:00",
                  "closes": "18:00:00+01:00"
                }
              ],
              "saturday": null,
              "sunday": null
            }
          ]
        }
      }
    ],
    "description": {
      "ita": "La Libera Università di Bolzano sorge in una delle più attraenti regioni
europee, al crocevia tra il mondo economico e culturale tedesco e italiano.",
      "deu": "Die Freie Universität Bozen befindet sich in einer der attraktivsten
Regionen Europas an der Schnittstelle zwischen dem deutschsprachigen und italienischen
Kultur- und Wirtschaftsraum.",
      "eng": "The Free University of Bozen-Bolzano is located in one of the most
fascinating European regions, at the crossroads between the German-speaking and Italian
economies and cultures."
    },
    "name": {
      "ita": "Libera Univeristà di Bolzano",
      "deu": "Freie Universität Bozen",
      "eng": "Free University of Bozen-Bolzano"
    },
    "shortName": {
      "eng": "Unibz"
    },
    "url": "https://www.unibz.it"
  },
  "relationships": {
    "categories": {
      "data": [
        {
          "type": "categories",
          "id": "alpinebits:organization"
        }
      ],
      "links": {
        "related": "https://example.com/2022-04/agents/1/categories"
      }
    },
    "multimediaDescriptions": {
      "data": [
        {
          "type": "mediaObjects",
          "id": "1"
        }
      ],
      "links": {
        "related": "https://example.com/2022-04/agents/1/multimediaDescriptions"
      }
    }
  },
  "links": {
    "self": "https://example.com/2022-04/agents/1"
  }
}
```

asciidoc/examples/agent.full.json

7.2.2. Categories

A resource that implements the concept of Category defined in the **AlpineBits® DestinationData Ontology**.

A JSON object representing such a resource MUST contain the following fields:

- `type`: the constant `"categories"` that identifies the resource as being of the type category.
- `id`: a string that uniquely and persistently identifies the category within a SERVER. This id MUST be a human-readable string composed of a namespace, identifying the context defining the category (e.g., the data provider or the standard), and the category name, these two separated by a colon (i.e., `"<namespace>:<categoryname>"`).

  Category id strings MUST be valid against the regular expression below. Examples of categories listed in this standard include `"alpinebits:person"` and `"alpinebits:organization"`, as categories of agents, and "schema:BusinessEvent" as a category of events.

  ```
  "^([a-z]|[A-Z]|[0-9])+:([a-z]|[A-Z]|[0-9])+$"
  ```

- `attributes`: an object containing the attributes of the category.
- `relationships`: an object containing the relationships of the category to other resources.
- `links`: an object containing the links related to the category.

Category resources are structured in the following way:

```
{
  "type": "categories",
  "id": "example:soccerEvent",
  "meta": { ... },
  "attributes": { ... },
  "relationships": { ... },
  "links": { ... }
}
```

Meta

See the definition of the `meta` object in Meta Object.

Attributes

The `attributes` object of the agent resource MUST contain the following fields:

- `abstract`: a text object containing a brief description for the category. Nullable. See the definition in Attributes Object.
- `description`: a text object containing a description of the category. Nullable. Conditional Assignment. See the definition in Attributes Object.

- `name`: a text object containing the complete name of the category. Differently from the category's `id`, the category's `name` serves no identification purpose, but only description. Therefore, the category's name SHOULD be a human-readable text friendly to end users. Non-nullable. Conditional Assignment. See the definition in Attributes Object.
- `namespace`: a string containing the namespace that prefixes the category's id. The namespace identifies the context where a category is defined. For instance, the `namespace` for the category `"alpinebits:person"`, which is defined in this standard, is `"alpinebits"`. Non-nullable. Non-empty.

  A SERVER SHALL NOT use `alpinebits` as the namespace of a category it defines.

  A SERVER SHOULD adopt a single namespace for all categories it defines.

  An example of a non-standardized category is `"unibz:academic"`, which could represent a category defined by the Free University of Bozen-Bolzano (acting as a SERVER) that classifies events that its students can attend.

- `resourceTypes`: an array of strings containing the resource types that the category is applicable to. For example, the value of `resourceTypes` for the category `"alpinebits:person"` is `[ "agents" ]`, since this category exclusively applies to resources with the type `"agents"`. Non-nullable. Non-empty.
- `shortName`: a text object containing a short name of the category. Nullable. See the definition in Attributes Object.
- `url`: a url object or string describing the category, such as a website or a Wikipedia page. Nullable. See the definition in Attributes Object.

A summary of the attributes is presented in the table below:

Field	Type	Constraints
abstract	text	Nullable
description	text	Nullable, Conditional Assignment
name	text	Non-nullable, Conditional Assignment
namespace	text	Non-nullable, Conditional Assignment
resourceTypes	Array of string	Non-nullable, Non-empty
shortName	text	Nullable
url	url	Nullable

Relationships

The `relationships` object of the agent resource MUST contain the following fields:

- `children`: a Reference to Many category resources that specialize the category. Every instance of a child category is also an instance of the referring category. Every category listed as child MUST refer back to the referring category as its parent category (i.e., inverse relations). Conditional assignment. Nullable. Non-empty.

  For example, if the category `"example:sportsEvent"` refers to the category `"example:soccerEvent"` as its child, every resource that instantiates `"example:soccerEvent"` also instantiates the category `"example:sportsEvent"`.

- `multimediaDescriptions`: a Reference to Many media object resources (see Media Objects) that are related to the category. See Section multimediaDescriptions. Nullable. Non-empty.
- `parents`: a Reference to Many category resources that are specialized by the category. Every instance of the referring category is also an instance of the parent category. Every category listed as

parent MUST refer back to the referring category as its child category (i.e., inverse relations). Conditional assignment. Nullable. Non-empty.

For example, if the category `"example:modernArtEvent"` refers to the category `"example:artEvent"` as its parent, every resource that instantiates `"example:modernArtEvent"` also instantiates the category `"example:artEvent"`.

A summary of the relationships is presented in the table below:

Field	Type	Constraints
children	Reference to Many object to Categories	Nullable, Non-empty
multimediaDescriptions	Reference to Many object to Media Objects	Nullable, Non-empty
parents	Reference to Many object to Categories	Nullable, Non-empty

Links

The `links` object of category resources MUST contain the following fields:

- `self`: a string representing the endpoint on which the container resource is available. Non-nullable.
- `resources`: a object where the key-value pairs contains URLs for retrieving resources that instantiate the category. The object MUST include key for each string in the `resourceTypes` attribute array, and the corresponding value in the pair is the endpoint for that resource type. Whenever supported, a filter for the category MUST be included (see example below). Non-nullable.

For example, the links object of a category resource may be presented as follows:

```
{
  "type": "categories",
  "id": "alpinebits:person",
  "attributes": {
    "resourceTypes": [ "agents" ],
    ...
  },
  "relationships": { ... },
  "links": {
    "self": "https://example.com/2022-04/categories/alpinebits:person",
    "resources": {
      "agents": "https://example.com/2022-
04/agents?filter[category][any]=alpinebits:person"
    }
  }
}
```

Examples

The following example presents an object containing the minimal information required of a category resource:

```
{
  "type": "categories",
  "id": "example:temporarilyRestricted",
  "meta": {
    "dataProvider": null,
    "lastUpdate": null
  },
  "attributes": {
    "abstract": null,
    "description": null,
    "name": {
      "eng": "Temporarily Restricted"
    },
    "namespace": "example",
    "resourceTypes": [ "venues" ],
    "shortName": null,
    "url": null
  },
  "relationships": {
    "children": null,
    "multimediaDescriptions": null,
    "parents": null
  },
  "links": {
    "self": "https://example.com/2022-04/categories/example:temporarilyRestricted",
    "resources": {
      "venues": "https://example.com/2022-
04/venues?filter[categories][any]=example:temporarilyRestricted"
    }
  }
}
```

asciidoc/examples/category.min.json

The following example presents an object representing a category resource:

```
{
  "type": "categories",
  "id": "example:soccerEvent",
  "meta": {
    "dataProvider": "https://example.com",
    "lastUpdate": "2022-04-01T08:00:00+02:00"
  },
  "attributes": {
    "abstract": {
      "ita": "Un evento sportivo legato al calcio.",
      "deu": "Ein Sportereignis, das mit Fußball zu tun hat.",
      "eng": "A sports event related to soccer."
    },
    "description": {
      "ita": "Un evento sportivo legato al calcio, per esempio una partita di calcio o un
incontro con i giocatori.",
      "deu": "Ein Sportereignis mit Bezug zum Fußball, z. B. ein Fußballspiel oder ein
Meet-and-Greet mit Spielern.",
      "eng": "A sports event related to soccer, for example a soccer match or a meet-and-
greet with players."
    },
    "name": {
      "ita": "Evento di Calcio",
      "deu": "Fußball-Event",
      "eng": "Soccer Event"
    },
    "namespace": "example",
    "resourceTypes": [ "events" ],
    "shortName": {
      "eng": "Soccer Event"
    },
    "url": "https://en.wikipedia.org/wiki/Association_football"
  },
  "relationships": {
    "children": {
      "data": [
        {
          "type": "categories",
          "id": "example:sub19SoccerMatch"
        }
      ],
      "links": {
        "related": "https://example.com/2022-04/categories/example:soccerEvent/children"
      }
    },
    "multimediaDescriptions": {
      "data": [
        {
          "type": "mediaObjects",
          "id": "1"
        }
      ],
      "links": {
        "related": "https://example.com/2022-
04/categories/example:soccerEvent/multimediaDescriptions"
      }
    },
    "parents": {
      "data": [
        {
          "type": "categories",
          "id": "schema:SportsEvent"
        }
      ],
      "links": {
        "related": "https://example.com/2022-04/categories/example:soccerEvent/parents"
      }
    }
  },
  "links": {
    "self": "https://example.com/2022-04/categories/example:soccerEvent",
    "resources": {
      "events": "https://example.com/2022-
04/events?filter[categories][any]=example:soccerEvent"
    }
  }
}
```

asciidoc/examples/category.full.json

7.2.3. Events

A resource that implements the concept of Event defined in the **AlpineBits® DestinationData Ontology**.

A JSON object representing such a resource MUST contain the following fields:

- `type`: the constant `"events"` that identifies the resource as being of the type event.
- `id`: a string that uniquely and persistently identifies the event within a SERVER. See the definition in Basic Fields.
- `attributes`: an object containing the attributes of the event.
- `relationships`: an object containing the relationships of the event to other resources.
- `links`: an object containing the links related to the event.

An event resource is structured as follows:

```
{
  "type": "events",
  "id": "1",
  "meta": { ... },
  "attributes": { ... },
  "relationships": { ... },
  "links": { ... }
}
```

Meta

See the definition of the `meta` object in Meta Object.

Attributes

The `attributes` object of an event resource MUST contain the following fields:

- `abstract`: a text object containing a brief description of the event. Nullable. See the definition in Attributes Object.
- `description`: a text object containing a description of the event. Nullable. Conditional Assignment. See the definition in Attributes Object.
- `endDate`: a string, formatted as date or date-time, representing when the event is planned to end. This field can only be `null` if `startDate` is defined.
- `inPersonCapacity`: an integer representing the total number of individuals that can attend the event in-person. This field can only be assigned in events having the one of the following categories: `"alpinebits:inPersonEvent"`, or `"alpinebits:hybridEvent"`. Nullable. Positive integer. Conditional assignment.
- `name`: a text object containing the complete name of the event. Non-nullable. Conditional Assignment. See the definition in Attributes Object.
- `onlineCapacity`: an integer representing the total number of individuals that can attend the event virtually. This field can only be assigned in events having the one of the following categories: `"alpinebits:virtualEvent"`, or `"alpinebits:hybridEvent"`. Nullable. Positive integer. Conditional assignment.
- `participationUrl`: a url object or string containing the URL for virtual event attendance. This field can only be assigned in events having the one of the following categories: `"alpinebits:virtualEvent"`, or `"alpinebits:hybridEvent"`. Nullable. Conditional assignment.
- `recorded`: a boolean value indicated whether the event will be recorded. Nullable.
- `registrationUrl`: a url object or string containing the URL for event registration. Nullable.
- `shortName`: a text object containing a short name of the event. Nullable. See the definition in Attributes Object.
- `startDate`: a string, formatted as date or date-time, representing when the event is planned to start. This field can only be `null` if `endDate` is defined.
- `status`: a string representing the current status of the event. Nullable. The possible values for this field are the following enumerated values:
  - `published`: the event has been published. It may or may not have happened.
  - `canceled`: the event has been canceled. Events can only be canceled before they have started.
- `url`: a url object or string describing the event, such as a website or a Wikipedia page. Nullable. See the definition in Attributes Object.

A summary of the attributes is presented in the table below:

Field	Type	Constraints
abstract	text	Nullable
description	text	Nullable
endDate	date or date-time	Nullable, Conditional Assignment
inPersonCapacity	integer	Nullable, Greater than Zero, Conditional Assignment
name	text	Non-Nullable
onlineCapacity	integer	Nullable, Greater than Zero, Conditional Assignment
participationUrl	url	Nullable, Conditional Assignment
recorded	boolean	Nullable
registrationUrl	url	Nullable
shortName	text	Nullable
startDate	date or date-time	Nullable, Conditional Assignment
status	string	Nullable, Enumeration
url	url	Nullable

Relationships

The `relationships` object of an event resource MUST contain the following fields:

- `categories`: a Reference to Many category resources that are instantiated by the event. See Section Categories. Non-nullable. Non-empty.

  The standard recommends the use of event types defined in Schema.org, which are identified by the `schema` namespace: `"schema:BusinessEvent"`, `"schema:ChildrensEvent"`, `"schema:ComedyEvent"`, `"schema:CourseInstance"`, `"schema:DanceEvent"`, `"schema:DeliveryEvent"`, `"schema:EducationEvent"`, `"schema:EventSeries"`, `"schema:ExhibitionEvent"`, `"schema:Festival"`, `"schema:FoodEvent"`, `"schema:Hackathon"`, `"schema:LiteraryEvent"`, `"schema:MusicEvent"`, `"schema:PublicationEvent"`, `"schema:SaleEvent"`, `"schema:ScreeningEvent"`, `"schema:SocialEvent"`, `"schema:SportsEvent"`, `"schema:TheaterEvent"`, `"schema:VisualArtsEvent"`.

  Additionally, the standard defines the three disjoint categories for events where every event MUST be assigned with exactly one of them:

  - `"alpinebits:inPersonEvent"`: an event resource representing a event planned for exclusive in-person attendance.
  - `"alpinebits:virtualEvent"`: an event resource representing a event planned for exclusive virtual attendance.
  - `"alpinebits:hybridEvent"`: an event resource representing a event planned for both in-person and virtual attendance.
- `contributors`: a Reference to Many agent resources (see Agents), which identifies the agents expected to participate in the event, such as a speaker who will give a talk or a musician who will perform at a concert. Contributors do not include those attending the event. Nullable. Non-empty.
- `multimediaDescriptions`: a Reference to Many media object resources (see Media Objects) that are related to the event. See Section multimediaDescriptions. Nullable. Non-empty.
- `organizers`: a Reference to Many agent resources (see Agents) identifying the persons or organizations responsible for organizing the event. Non-nullable. Non-empty.
- `publisher`: a Reference to One agent resource (see Agents) who originally provided the data about the event. The publisher is not the organization who manages the system where the data was

originally inserted, but the organization actually provided the data. Non-nullable.

- `series`: a Reference to One event series resource (see Event Series) of which the event is an edition of. For instance, the Südtirol Jazz Festival 2020 is an edition of the Südtirol Jazz Festival. Nullable.

  Not that sub-events do not share the same series as the whole, but they may be editions of unrelated event series. For example, the "South Tyrol Jazz Festival 2021" may be an edition of the "South Tyrol Jazz Festival" series, while the presentation of "András Dés Rangers on 04/07/2021", its sub-event, may be an edition of the "András Dés Rangers European Tour 2021".

- `sponsors`: a Reference to Many agent resources (see Agents) identifying the persons or organizations who are sponsoring the event. Nullable. Non-empty.
- `subEvents`: a Reference to Many event resources (see Events) identifying the parts of the event. For instance, a festival may consist of several concerts and a conference may consist of several presentations and a keynote. Nullable. Non-empty.

  There SHALL NOT be a circular composition of events. For instance, if X is a sub-event of Y, and Y is a sub-event of Z, Z SHALL NOT be a sub-event of X.

  Note that sub-events are not restricted to the temporal boundaries of their parent events. For instance, a conference that starts on 15/03/2020 and ends on 20/03/2020 may have, as a sub-event, a pre-conference meeting that is scheduled for the 14/03/2020.

- `venues`: a Reference to Many venue resources (see Venues) identifying where the event will happen. This field is non-nullable in events having the one of the following categories: `"alpinebits:inPersonEvent"`, or `"alpinebits:hybridEvent"`. Nullable. Non-empty. Conditional Assignment.

A summary of the relationships is presented in the table below:

Field	**Type**	**Constraints**
categories	Reference to Many object to Categories	Non-Nullable, Non-empty
contributors	Reference to Many object to Agents	Nullable, Non-empty
multimediaDescriptions	Reference to Many object to Media Objects	Nullable, Non-empty
organizers	Reference to Many object to Agents	Non-Nullable, Non-empty
publisher	Reference to One object to Agents	Non-Nullable
series	Reference to One object to Event Series	Nullable
sponsors	Reference to Many object to Agents	Nullable, Non-empty
subEvents	Reference to Many object to Events	Nullable, Non-empty
venues	Reference to Many object to Venues	Nullable, Non-empty, Conditional Assignment

Links

See the definition of the `links` object in Links Object.

Examples

The following example contains the minimal information required for an event resource:

```
{
  "type": "events",
  "id": "1",
  "meta": {
    "dataProvider": null,
    "lastUpdate": null
  },
  "attributes": {
    "name": {
      "eng": "Südtirol Jazz Festival 2018"
    },
    "shortName": null,
    "description": null,
    "abstract": null,
    "startDate": "2018-06-29T00:00:00+00:00",
    "endDate": null,
    "url": null,
    "status": null,
    "inPersonCapacity": null,
    "onlineCapacity": null,
    "participationUrl": null,
    "recorded": null,
    "registrationUrl": null
  },
  "relationships": {
    "series": null,
    "publisher": {
      "data": {
        "type": "agents",
        "id": "1"
      },
      "links": {
        "related": "https://example.com/2022-04/events/1/publisher"
      }
    },
    "organizers": {
      "data": [
        {
          "type": "agents",
          "id": "2"
        }
      ],
      "links": {
        "related": "https://example.com/2022-04/events/1/organizers"
      }
    },
    "sponsors": null,
    "contributors": null,
    "categories": {
      "data": {
        "type": "categories",
        "id": "alpinebits:inPersonEvent"
      },
      "links": {
        "related": "https://example.com/2022-04/events/1/categories"
      }
    },
    "multimediaDescriptions": null,
    "venues": {
      "data": [
        {
          "type": "venues",
          "id": "1"
        }
      ],
      "links": {
        "related": "https://example.com/2022-04/events/1/venues"
      }
    },
    "subEvents": null
  },
  "links": {
    "self": "https://example.com/2022-04/events/1"
  }
}
```

examples/event.min.json

The following example illustrates the fields defined for event resources:

```
{
  "type": "events",
  "id": "1",
  "meta": {
    "dataProvider": "https://example.com",
    "lastUpdate": "2020-04-01T08:00:00+02:00"
  },
  "attributes": {
    "name": {
      "eng": "Südtirol Jazz Festival 2020",
      "deu": "Südtirol Jazz Festival 2020",
      "ita": "Südtirol Jazz Festival 2020"
    },
    "shortName": {
      "eng": "Südtirol Jazz Festival 2020"
    },
    "description": {
      "deu": "Beim Südtirol Jazzfestival Alto Adige wagen sich Solisten und Ensembles
nicht nur in schwindelerregende musikalische „Höhen“ vor, sondern besteigen, mit ihren
Instrumenten im Gepäck, sogar „echte“ Berge und bespielen dabei eine atemberaubende alpine
Landschaft. Das Festival selbst beschreitet oft ganz neue Wege: selbstbewusst und sich
mitunter über zerklüftete Spalten herantastend, versucht es in neue Klanglandschaften
vorzudringen. Mainstream und Konzertsäle sind beim Südtirol Jazzfestival Alto Adige vom
Aussterben bedroht...."
    },
    "abstract": {
      "deu": "Das Südtirol Jazzfestival Alto Adige ist ein in Südtirol stattfindendes
Musikfestival für Jazz und experimentelle Musik."
    },
    "startDate": "2020-06-26T21:00:00+00:00",
    "endDate": "2020-07-07T23:30:00+00:00",
    "url": "https://www.suedtiroljazzfestival.com/",
    "status": "published",
    "inPersonCapacity": 1000,
    "onlineCapacity": 10000,
    "participationUrl": "https://www.example-streaming-site.com/asdeZCawdeGfdg44fas",
    "recorded": true,
    "registrationUrl": "https://www.example-tickets-site.com/asdeZCawdeGfdg44fas"
  },
  "relationships": {
    "series": {
      "data": {
        "type": "eventSeries",
        "id": "1"
      },
      "links": {
        "related": "https://example.com/2022-04/events/1/eventSeries"
      }
    },
    "publisher": {
      "data": {
        "type": "agents",
        "id": "1"
      },
      "links": {
        "related": "https://example.com/2022-04/events/1/publisher"
      }
    },
    "organizers": {
      "data": [
        {
          "type": "agents",
          "id": "2"
        }
      ],
      "links": {
        "related": "https://example.com/2022-04/events/1/organizers"
      }
    },
    "sponsors": {
      "data": [
        {
          "type": "agents",
          "id": "3"
        },
        {
          "type": "agents",
          "id": "4"
        }
      ],
      "links": {
        "related": "https://example.com/2022-04/events/1/sponsors"
      }
    },
    "contributors": {
      "data": [
        {
          "type": "agents",
          "id": "5"
        }
      ],
      "links": {
        "related": "https://example.com/2022-04/events/1/contributors"
      }
    },
    "categories": {
      "data": [
        {
          "type": "categories",
          "id": "alpinebits:hybridEvent"
        },
        {
          "type": "categories",
          "id": "schema:MusicEvent"
        }
      ],
      "links": {
        "related": "https://example.com/2022-04/events/1/categories"
      }
    },
    "multimediaDescriptions": {
      "data": [
        {
          "type": "mediaObjects",
          "id": "1"
        },
        {
          "type": "mediaObjects",
          "id": "2"
        }
      ],
      "links": {
        "related": "https://example.com/2022-04/events/1/multimediaDescriptions"
      }
    },
    "venues": {
      "data": [
        {
          "type": "venues",
          "id": "1"
        }
      ],
      "links": {
        "related": "https://example.com/2022-04/events/1/venues"
      }
    },
    "subEvents": {
      "data": [
        {
          "type": "events",
          "id": "2"
        },
        {
          "type": "events",
          "id": "3"
        }
      ],
      "links": {
        "related": "https://example.com/2022-04/events/1/subEvents"
      }
    }
  },
  "links": {
    "self": "https://example.com/2022-04/events/1"
  }
}
```

examples/event.full.json

7.2.4. Event Series

A resource that implements the concept of Event Series defined in the **AlpineBits® DestinationData Ontology**.

A JSON object representing such a resource MUST contain the following fields:

- `type`: the constant `"eventSeries"` that identifies the resource as being of the type event series.
- `id`: a string that uniquely and persistently identifies the event series within a SERVER. See the definition in Basic Fields.
- `attributes`: an object containing the attributes of the event series.
- `relationships`: an object containing the relationships of the event series to other resources.
- `links`: an object containing the links related to the event series.

Event series resources are structured in the following way:

```
{
  "type": "eventSeries",
  "id": "1",
  "meta": { ... },
  "attributes": { ... },
  "relationships": { ... },
  "links": { ... }
}
```

Meta

See the definition of the `meta` object in Meta Object.

Attributes

The `attributes` object of the event resource MUST contain the following fields:

- `abstract`: a text object containing a brief description for the event series. Nullable. See the definition in Attributes Object.
- `description`: a text object containing a description of the event series. Nullable. Conditional Assignment. See the definition in Attributes Object.
- `frequency`: a string representing how often editions of the event series are organized according to an enumeration of possible values. Nullable. Enumeration.
  For example, event series with the `frequency` field set to `"weekly"` may include street markets and guided city tours, while event series with the `frequency` field set to `"yearly"` may include Südtirol Jazz Festival or Flower Festival.
  If set, the `frequency` field MUST be assigned one of the following values:
  - `daily`: when editions of the event series are organized **every day**, such as the daily street market of Bolzano.
  - `weekly`: when editions of the event series are organized **every week**, such as the weekly farmers' market of Bolzano.
  - `monthly`: when editions of the event series are organized **every month**, such as a flea market organized every month.
  - `bimonthly`: when editions of the event series are organized **every two months**.
  - `quarterly`: when editions of the event series are organized **every three months**.
  - `annual`: when editions of the event series are organized **every year**, such as the Südtirol Jazz Festival or Bolzano's Christmas Market.
  - `biennial`: when editions of the event series are organized **every two years**.
  - `triennial`: when editions of the event series are organized **every three years**.
- `name`: a text object containing the complete name of the event series. Non-nullable. Conditional Assignment. See the definition in Attributes Object.
- `shortName`: a text object containing a short name of the event series. Nullable. See the definition in Attributes Object.
- `url`: a url object or string describing the event series, such as a website or a Wikipedia page. Nullable. See the definition in Attributes Object.

A summary of the attributes is presented in the table below:

Field	Type	Constraints
abstract	text	Nullable
description	text	Nullable, Conditional Assignment
frequency	string	Nullable, Enumeration
name	url	Non-nullable, Conditional Assignment
shortName	text	Nullable
url	url	Nullable

Relationships

The `relationships` object of an event series resource MUST contain the following fields:

- `categories`: a Reference to Many category resources that are instantiated by the event series. See Section Categories. Nullable. Non-empty.

  No category is pre-defined by the standard.

- `editions`: a Reference to Many event resources (see Events) that are editions of the event series. Nullable. Non-empty.

  Note that no two editions of an event series can be a sub-event of the other.

- `multimediaDescriptions`: a Reference to Many media object resources (see Media Objects) that are related to the event series. See Section multimediaDescriptions. Nullable. Non-empty.

A summary of the relationships is presented in the table below:

Field	Type	Constraints
categories	Reference to Many object to Categories	Nullable, Non-empty
editions	Reference to Many object to Events	Nullable, Non-empty
multimediaDescriptions	Reference to Many object to Media Objects	Nullable, Non-empty

Links

See the definition of the `links` object in Links Object.

Examples

The following example presents an object containing the minimal information required of an event series resource:

```
{
  "type": "eventSeries",
  "id": "1",
  "meta": {
    "dataProvider": null,
    "lastUpdate": null
  },
  "attributes": {
    "abstract": null,
    "description": null,
    "frequency": null,
    "name": {
      "eng": "Südtirol Jazz Festival"
    },
    "shortName": null,
    "url": null
  },
  "relationships": {
    "categories": null,
    "multimediaDescriptions": null,
    "editions": null
  },
  "links": {
    "self": "https://example.com/2022-04/eventSeries/1"
  }
}
```

asciidoc/examples/eventseries.min.json

The following example presents an object representing an event series resource:

```
{
  "type": "eventSeries",
  "id": "1",
  "meta": {
    "dataProvider": "https://example.com",
    "lastUpdate": "2020-04-01T08:00:00+02:00"
  },
  "attributes": {
    "abstract": {
      "eng": "The Südtirol Jazzfestival Alto Adige was held for the first time in 1982
under the name of “Jazz Summer”..."
```

```
    },
    "description": {
      "eng": "The Südtirol Jazzfestival Alto Adige was held for the first time in 1982
under the name of “Jazz Summer”, which went on to become “Jazz & Other”. While in the
early years the concerts were played only in Bolzano itself, today the festival stretches
throughout the whole of South Tyrol and beyond."
    },
    "frequency": "annual",
    "name": {
      "eng": "Südtirol Jazz Festival",
      "ita": "Südtirol Jazz Festival",
      "deu": "Südtirol Jazz Festival"
    },
    "shortName": {
      "eng": "Südtirol Jazz Festival"
    },
    "url": "https://www.suedtiroljazzfestival.com/"
  },
  "relationships": {
    "categories": {
      "data": [
        {
          "type": "categories",
          "id": "example:festival"
        }
      ],
      "links": {
        "related": "https://example.com/2022-04/eventSeries/1/categories"
      }
    },
    "multimediaDescriptions": {
      "data": [
        {
          "type": "mediaObjects",
          "id": "1"
        }
      ],
      "links": {
        "related": "https://example.com/2022-04/eventSeries/1/multimediaDescriptions"
      }
    },
    "editions": {
      "data": [
        {
          "type": "events",
          "id": "1"
        },
        {
          "type": "events",
          "id": "2"
        }
      ],
      "links": {
        "related": "https://example.com/2022-04/eventSeries/1/editions"
      }
    }
  },
  "links": {
    "self": "https://example.com/2022-04/eventSeries/1"
  }
}
```

asciidoc/examples/eventseries.full.json

7.2.5. Features

A resource that implements the concept of Feature defined in the **AlpineBits® DestinationData Ontology**.

A JSON object representing such a resource MUST contain the following fields:

- `type`: the constant `"features"` that identifies the resource as being of the type feature.
- `id`: a string that uniquely and persistently identifies the feature within a SERVER. This id MUST be a human-readable string composed of a namespace, identifying the context defining the feature (e.g., the SERVER or the standard), and the feature name, these two separated by a colon (i.e., `"<namespace>:<featurename>"`).

  Feature id strings MUST be valid against the regular expression below.

  ```
  "^([a-z]|[A-Z]|[0-9])+:([a-z]|[A-Z]|[0-9])+$"
  ```

- `attributes`: an object containing the attributes of the feature.
- `relationships`: an object containing the relationships of the features to other resources.
- `links`: an object containing the links related to the feature.

Feature resources are structured in the following way:

```
{
  "type": "features",
  "id": "example:snowparkRamp",
  "meta": { ... },
  "attributes": { ... },
  "relationships": { ... },
  "links": { ... }
}
```

Meta

See the definition of the `meta` object in Meta Object.

Attributes

The `attributes` object of the agent resource MUST contain the following fields:

- `abstract`: a text object containing a brief description for the feature. Nullable. See the definition in Attributes Object.
- `description`: a text object containing a description of the feature. Nullable. Conditional Assignment. See the definition in Attributes Object.
- `name`: a text object containing the complete name of the feature. Differently from the feature's `id`, its `name` serves no identification purpose, but only description. Therefore, the `name` SHOULD be a human-readable text friendly to end users. Non-nullable. Conditional Assignment. See the definition in Attributes Object.
- `namespace`: a string containing the namespace that prefixes the feature's id. The namespace identifies the context where a feature is defined. For instance, a SERVER may define the features `"example:snowparkRamp"` and `"example:snowparkRail"` under the `namespace` `"example"`. Non-nullable. Non-empty.

  A SERVER SHALL NOT use `alpinebits` as the namespace of a feature it defines.

  A SERVER SHOULD adopt a single namespace for all features it defines.

- `resourceTypes`: an array of strings containing the resource types that can present the feature. For example, the value of `resourceTypes` for the feature `"example:snowparkRamp"` is `[ "snowpark" ]`. At the moment, the standard only accounts for features of snowparks. Non-nullable. Non-empty.
- `shortName`: a text object containing a short name of the feature. Nullable. See the definition in Attributes Object.

- `url`: a url object or string describing the feature, such as a website or a Wikipedia page. Nullable. See the definition in Attributes Object.

A summary of the attributes is presented in the table below:

Field	Type	Constraints
abstract	text	Nullable
description	text	Nullable, Conditional Assignment
name	text	Non-nullable, Conditional Assignment
namespace	text	Non-nullable, Conditional Assignment
resourceTypes	Array of string	Non-nullable, Non-empty
shortName	text	Nullable
url	url	Nullable

Relationships

The `relationships` object of the agent resource MUST contain the following fields:

- `children`: a Reference to Many feature resources that specialize the feature. Every resource presenting a child feature also must also present the referring feature. Every feature listed as child MUST refer back to the referring feature as its parent feature (i.e., inverse relations). Conditional assignment. Nullable. Non-empty.

  For example, if the feature `"example:snowparkRamp"` refers to the feature `"example:snowboardingRamp"` as its child, every resource that presents `"example:snowboardingRamp"` also presents the feature `"example:snowparkRamp"`.

- `multimediaDescriptions`: a Reference to Many media object resources (see Media Objects) that are related to the feature. See Section multimediaDescriptions. Nullable. Non-empty.
- `parents`: a Reference to Many feature resources that are specialized by the feature. Every resource presenting the referring feature must also present the parent feature. Every feature listed as parent MUST refer back to the referring feature as its child feature (i.e., inverse relations). Conditional assignment. Nullable. Non-empty.

  For example, if the feature `"example:snowboardingJib"` refers to the feature `"example:snowparkRail"` as its parent, every resource that presents `"example:snowboardingJib"` also presents the feature `"example:snowparkRail"`.

A summary of the relationships is presented in the table below:

Field	Type	Constraints
children	Reference to Many object to Features	Nullable, Non-empty
multimediaDescriptions	Reference to Many object to Media Objects	Nullable, Non-empty
parents	Reference to Many object to Features	Nullable, Non-empty

Links

The `links` object of feature resources MUST contain the following fields:

- `self`: a string representing the endpoint on which the container resource is available. Non-nullable.
- `resources`: a object where the key-value pairs contains URLs for retrieving resources that present

the feature. The object MUST include key for each string in the `resourceTypes` attribute array, and the corresponding value in the pair is the endpoint for that resource type. Whenever supported, a filter for the feature MUST be included (see example below). Non-nullable.

For example, the links object of a feature resource may be presented as follows:

```
{
  "type": "features",
  "id": "example:snowparkRail",
  "attributes": {
    "resourceTypes": [ "snowparks" ],
    ...
  },
  "relationships": { ... },
  "links": {
    "self": "https://example.com/2022-04/features/example:snowparkRail",
    "resources": {
      "snowparks": "https://example.com/2022-
04/snowparks?filter[features][any]=example:snowparkRail"
    }
  }
}
```

Examples

The following example presents an object containing the minimal information required of a feature resource:

```
{
  "type": "features",
  "id": "example:snowparkRamp",
  "meta": {
    "dataProvider": null,
    "lastUpdate": null
  },
  "attributes": {
    "abstract": null,
    "description": null,
    "name": {
      "eng": "Snowpark Ramp"
    },
    "namespace": "example",
    "resourceTypes": [ "snowparks" ],
    "shortName": null,
    "url": null
  },
  "relationships": {
    "children": null,
    "multimediaDescriptions": null,
    "parents": null
  },
  "links": {
    "self": "https://example.com/2022-04/features/example:snowparkRamp",
    "resources": {
      "snowparks": "https://example.com/2022-
04/snowparks?filter[features][any]=example:snowparkRamp"
    }
  }
}
```

asciidoc/examples/feature.min.json

The following example presents an object representing a feature resource:

```
{
  "type": "features",
  "id": "example:snowparkRamp",
  "meta": {
    "dataProvider": "https://example.com",
    "lastUpdate": "2022-04-01T08:00:00+02:00"
  },
  "attributes": {
    "abstract": {
      "ita": "Una rampa di snowpark è una struttura per la pratica di manovre aeree negli
sport invernali radicali.",
      "deu": "Eine Snowpark-Rampe ist eine Struktur zum Üben von Flugmanövern im radikalen
Wintersport.",
      "eng": "A snowpark ramp is a structure for the practice of aerial maneuvers in
radical winter sports."
    },
    "description": {
      "ita": "Una rampa da snowpark è una struttura presente negli snowpark progettata per
supportare l'esecuzione di manovre aeree negli sport invernali radicali.",
      "deu": "Eine Snowpark-Rampe ist eine Einrichtung in Snowparks, die dazu dient, die
Ausführung von Flugmanövern bei radikalen Wintersportarten zu unterstützen.",
      "eng": "A snowpark ramp is a feature present in snowparks designed to support the
execution of aerial maneuvers in radical winter sports."
    },
    "name": {
      "ita": "Snowpark Ramp",
      "deu": "Snowpark Rampe",
      "eng": "Snowpark Ramp"
    },
    "namespace": "example",
    "resourceTypes": [ "snowparks" ],
    "shortName": {
      "eng": "Snowpark Ramp"
    },
    "url": "https://en.wikipedia.org/wiki/Terrain_park"
  },
  "relationships": {
    "children": {
      "data": [
        {
          "type": "features",
          "id": "example:snowboardRamp"
        }
      ],
      "links": {
        "related": "https://example.com/2022-04/features/example:snowparkRamp/children"
      }
    },
    "multimediaDescriptions": {
      "data": [
        {
          "type": "mediaObjects",
          "id": "1"
        }
      ],
      "links": {
        "related": "https://example.com/2022-
04/features/example:snowparkRamp/multimediaDescriptions"
      }
    },
    "parents": {
      "data": [
        {
          "type": "features",
          "id": "example:radicalSportsRamp"
        }
      ],
      "links": {
        "related": "https://example.com/2022-04/features/example:snowparkRamp/parents"
      }
    }
  },
  "links": {
    "self": "https://example.com/2022-04/features/example:snowparkRamp",
    "resources": {
      "snowparks": "https://example.com/2022-
04/snowparks?filter[features][any]=example:snowparkRamp"
    }
  }
}
```

`asciidoc/examples/feature.full.json`

7.2.6. Lifts

A resource that implements the concept of Lift defined in the **AlpineBits® DestinationData Ontology**.

A JSON object representing such a resource MUST contain the following fields:

- `type`: the constant `"lifts"` that identifies the resource as being a lift.
- `id`: a string that uniquely and persistently identifies the lift within a SERVER. See the definition in Basic Fields.
- `attributes`: an object containing the attributes of the lift.
- `relationships`: an object containing the relationships of the lift to other resources.
- `links`: an object containing the links related to the lift.

A lift resource is structured as follows:

```
{
  "type": "lifts",
  "id": "1",
  "meta": { ... },
  "attributes": { ... },
  "relationships": { ... },
  "links": { ... }
}
```

Meta

See the definition of the `meta` object in Meta Object.

Attributes

The `attributes` object of a lift resource MUST contain the following fields:

- `abstract`: a text object containing a brief description for the lift. Nullable. See the definition in Attributes Object.
- `address`: an address object representing the address of the lift. The address should only be set if the lift is directly accessible from a public road. In such cases, the address should be that of the base station. Nullable.
- `capacity`: a number representing how many persons the lift can transport hourly on average. Nullable. Positive value.
- `description`: a text object containing a description of the lift. Nullable. Conditional Assignment. See the definition in Attributes Object.
- `geometries`: an array of geometry objects each of which represents the location of the lift in terms of GPS coordinates. There should be at most one geometry object of each type (e.g. Point, LineString). Nullable. Non-empty.
- `howToArrive`: a text object containing instructions on how to arrive at the lift. Nullable.
- `length`: a number representing the length of the lift in meters. Nullable.

- `maxAltitude`: a number representing the highest elevation point of the lift in meters above sea level. Nullable.
- `minAltitude`: a number representing the lowest elevation point of the lift in meters above sea level. Nullable.
- `name`: a text object containing the complete name of the lift. Non-nullable. Conditional Assignment. See the definition in Attributes Object.
- `openingHours`: an hours specification object representing the hours in which the lift is open to the public. Nullable.
- `personsPerChair`: an integer representing the number of persons that fit in a single chair/cabin of a lift. Nullable. Positive value.
- `shortName`: a text object containing a short name of the lift. Nullable. See the definition in Attributes Object.
- `url`: a url object or string describing the lift, such as a website or a Wikipedia page. Nullable. See the definition in Attributes Object.

A summary of the attributes is presented in the table below:

Field	Type	Constraints
abstract	text	Nullable
address	address	Nullable
capacity	number	Nullable, Greater than Zero
description	text	Nullable, Conditional Assignment
geometries	Array of geometry	Nullable, Non-empty
howToArrive	text	Nullable
length	number	Nullable, Unit of Measure, Greater than Zero
maxAltitude	number	Nullable, Unit of Measure
minAltitude	number	Nullable, Unit of Measure
name	url	Non-nullable, Conditional Assignment
openingHours	hours specification	Nullable
personsPerChair	integer	Nullable, Greater than Zero
shortName	text	Nullable
url	url	Nullable

Relationships

The `relationships` object of a lift resource MUST contain the following fields:

- `categories`: a Reference to Many category resources that are instantiated by the lift. See Section Categories. Nullable. Non-empty.

  The standard recommends the following categories for lifts:, `"alpinebits:chairlift"`, `"alpinebits:gondola"`, `"alpinebits:skilift"`, `"alpinebits:cablecar`, `"alpinebits:funicular"`, `"alpinebits:magic-carpet"`, `"alpinebits:skibus"`, `"alpinebits:train"`.

- `connections`: a Reference to Many place resources that identify the places that are physically accessible from the lift, which may include Mountain Areas, other Lifts, Snowparks, and Ski Slopes. Nullable. Non-empty.

  Notice that connections between place resources may not be symmetrical (i.e., bidirectional). For example, a place like a lift may give access to a snowpark, but the snowpark may not give access back to it.

- `multimediaDescriptions`: a Reference to Many media object resources (see Media Objects) that are related to the lift. See Section multimediaDescriptions. Nullable. Non-empty.

A summary of the relationships is presented in the table below:

Field	Type	Constraints
categories	Reference to Many object to Categories	Nullable, Non-empty
connections	Reference to Many object to Mountain Areas, Ski Slopes, Lifts, and Snowparks	Nullable, Non-empty
multimediaDescriptions	Reference to Many object to Media Objects	Nullable, Non-empty

Links

See the definition of the `links` object in Links Object.

Examples

The following example contains the minimal information required for a lift resource:

```
{
  "type": "lifts",
  "id": "merano2000-l1",
  "meta": {
    "dataProvider": null,
    "lastUpdate": null
  },
  "attributes": {
    "name": {
      "eng": "Ropeway Merano 2000"
    },
    "shortName": null,
    "description": null,
    "abstract": null,
    "url": null,
    "length": null,
    "minAltitude": null,
    "maxAltitude": null,
    "capacity": null,
    "personsPerChair": null,
    "openingHours": null,
    "address": {
      "street": null,
      "city": {
        "eng": "Merano"
      },
      "region": null,
      "country": "IT",
      "zipcode": null,
      "categories": null,
      "complement": null
    },
    "geometries": null,
    "howToArrive": null
  },
  "relationships": {
    "connections": null,
    "categories": null,
    "multimediaDescriptions": null
  },
  "links": {
    "self": "https://example.com/2022-04/lifts/1"
  }
}
```

asciidoc/examples/lift.min.json

The following example illustrates the fields defined for lift resources:

```
{
  "type": "lifts",
  "id": "1",
  "meta": {
    "dataProvider": "https://example.com",
    "lastUpdate": "2020-04-01T08:00:00+02:00"
  },
  "attributes": {
    "name": {
      "deu": "Bergbahn Meran 2000",
      "eng": "Ropeway Merano 2000",
      "ita": "Funivia Merano 2000"
    },
    "shortName": {
      "deu": "Meran 2000",
      "eng": "Merano 2000",
      "ita": "Merano 2000"
    },
    "description": {
      "ita": "Situata a pochi minuti dalla città di Merano, la Funivia conduce all’area
sciistica ed escursionistica Merano 2000, un luogo ideale in cui passare una vacanza
all’insegna del movimento e della buona cucina con tutta la famiglia. Grazie al suo clima
mite, l’area offre infinite possibilità per il tempo libero in ogni stagione: impianti di
risalita, sentieri escursionistici facili e numerosi rifugi in cui sostare per assaporare
l’ottima cucina locale in un paesaggio ricco e dominato dalla tranquillità."
    },
    "abstract": {
      "ita": "Situata a pochi minuti dalla città di Merano, la Funivia conduce all’area
sciistica ed escursionistica Merano 2000, un luogo ideale in cui passare una vacanza
all’insegna del movimento e della buona cucina con tutta la famiglia..."
    },
    "url": "https://example.com",
    "length": 3650,
    "minAltitude": 1000,
    "maxAltitude": 2350,
    "capacity": 200,
    "personsPerChair": 10,
    "openingHours": {
      "dailySchedules": {
        "2020-12-25": null
      },
      "weeklySchedules": [
        {
          "validFrom": "2020-01-01",
          "validTo": "2020-12-31",
          "monday": [
            {
              "opens": "08:00:00+01:00",
              "closes": "18:00:00+01:00"
            }
          ],
          "tuesday": [
            {
              "opens": "08:00:00+01:00",
              "closes": "18:00:00+01:00"
            }
          ],
          "wednesday": [
            {
              "opens": "08:00:00+01:00",
              "closes": "18:00:00+01:00"
            }
          ],
          "thursday": [
            {
              "opens": "08:00:00+01:00",
              "closes": "18:00:00+01:00"
            }
          ],
          "friday": [
            {
              "opens": "08:00:00+01:00",
              "closes": "18:00:00+01:00"
            }
          ],
          "saturday": [
            {
              "opens": "08:00:00+01:00",
              "closes": "18:00:00+01:00"
            }
          ],
          "sunday": [
            {
              "opens": "08:00:00+01:00",
              "closes": "18:00:00+01:00"
            }
          ]
        }
        ]
      },
      "address": {
        "street": {
          "deu": "Naifweg 37",
          "eng": "37 Val di Nova Street",
          "ita": "Via Val di Nova, 37"
        },
        "city": {
          "deu": "Meran",
          "eng": "Merano",
          "ita": "Merano"
        },
        "region": {
          "deu": "Trentino-Südtirol",
          "eng": "Trentino-Alto Adige",
          "ita": "Trentino-Alto Adige"
        },
        "country": "IT",
        "zipcode": "39012",
        "categories": null,
        "complement": null
      },
      "geometries": [
        {
          "type": "LineString",
          "coordinates": [
            [
              11.305682659149168,
              46.66705018437341
            ],
            [
              11.30692720413208,
              46.667182709603225
            ],
            [
              11.308064460754393,
              46.667491933875965
            ]
          ]
        }
      ],
      "howToArrive": {
        "ita": "Fino alla stazione a valle della Funivia Merano 2000: dalla stazione
ferroviaria di Merano si raggiunge in pochi minuti la stazione a valle Val di Nova con la
linea urbana 1A. Da Scena c'è il bus vacanze che porta fino alla Val di Nova.",
        "deu": "Zur Talstation der Bergbahn Meran 2000: vom Zugbahnhof Meran erreicht man in
wenigen Minuten die Talstation Naif mit der Buslinie 1A. Von Schenna aus fährt der
Gästebus bis zur Talstation Naif."
      }
    },
    "relationships": {
      "connections": {
        "data": [
          {
            "type": "skiSlopes",
            "id": "1"
          },
          {
            "type": "skiSlopes",
            "id": "2"
          }
        ],
        "links": {
          "related": "https://example.com/2022-04/lifts/1/connections/"
        }
      },
      "categories": {
        "data": [
          {
            "type": "categories",
            "id": "alpinebits:cablecar"
          }
        ],
        "links": {
          "related": "https://example.com/2022-04/lifts/1/categories"
        }
      },
      "multimediaDescriptions": {
        "data": [
          {
            "type": "mediaObjects",
            "id": "1"
          }
        ],
        "links": {
          "related": "https://example.com/2022-04/lifts/1/multimediaDescriptions/"
        }
      }
    },
    "links": {
      "self": "https://example.com/2022-04/lifts/1"
    }
  }
```

asciidoc/examples/lift.full.json

7.2.7. Media Objects

A resource that implements the concept of Media Object defined in the **AlpineBits® DestinationData Ontology**.

A JSON object representing such a resource MUST contain the following fields: * `type`: the constant `"mediaObjects"` that identifies the resource as being of the type media object.

- `id`: a string that uniquely and persistently identifies the media object within a SERVER. See the definition in Basic Fields.
- `attributes`: an object containing the attributes of the media object.
- `relationships`: an object containing the relationships of the media object to other resources.
- `links`: an object containing the links related to the media object.

A media object resource is structured as follows:

```
{
  "type": "mediaObjects",
  "id": "1",
  "meta": { ... },
  "attributes": { ... },
  "relationships": { ... },
  "links": { ... }
}
```

Meta

See the definition of the `meta` object in Meta Object.

Attributes

The `attributes` object of the media object resource MUST contain the following fields:

- `abstract`: a text object containing a brief description of the media object. Nullable. See the definition in Attributes Object.
- `author`: a string that identifies the author of a media object, like an image, a video, or a song.

Nullable.

When assigned a value, the `author` SHOULD follow the example below representing the author's name and e-mail contact:

```
"author": "\"Mary Jane\" <mary.jane@example.com>"
```

- `contentType`: a string that represents the media type (formerly known as MIME type) of the media object. Non-nullable. Regular Expression.
  For example, possible values of `contentType` include yet are not limited to: `"video/mp4"`, for videos, `"image/png"`, for images, and "audio/mpeg3", for audio.
  The allowed media types are defined by IANA. The following regular expression constraints the valid pattern for the `contentType` string.

```
"^(application|audio|font|example|image|message|model|multipart|text|video)/[a-zA-Z0-9-
.+]+$"
```

- `description`: a text object containing a description of the media object. Nullable. Conditional Assignment. See the definition in Attributes Object.
- `duration`: a number representing the duration of an audio or a video in seconds. Nullable. Positive value.
  If the media object is neither an audio or a video object, i.e., `contentType` contains a substring following the pattern `"^(application|font|example|image|message|model|multipart|text)"`, `duration` MUST be set to null.
  For example, a short video shared in social platforms may have its `duration` set to `45` seconds, while a documentary of an event may have its `duration` set to `5774` seconds (i.e., 1 hour, 36 minutes and 14 seconds).
- `height`: a number representing the height of an image or a video in pixels. Nullable. Positive value.
  If the media object is neither an image or a video, i.e. `contentType` contains a substring following the pattern `"^(application|audio|font|example|message|model|multipart|text)"`, `height` MUST be set to null.
  For example, the `height` of a Full-HD video is expected to be `1080` pixels, while the `height` of a 4K video is expected to be `2160` pixels.
- `license`: a string that represents the license applied to the media object. The value of this field SHOULD be a valid license identifier as defined in SPDX License List (e.g. CC-BY-4.0, FreeImage). Nullable.
- `name`: a text object containing the complete name of the media object. Non-nullable. Conditional Assignment. See the definition in Attributes Object.
- `shortName`: a text object containing a short name of the media object. Nullable. See the definition in Attributes Object.
- `url`: a url string representing the media object's source. Non-nullable. See the definition in Attributes Object.
  The `url` string may be used for the exchange of HTML data with the condition that whenever this HTML data contains some textual description of another resource (i.e., `abstract`, `description`, `name`, or `shortName`), the resource MUST have the equivalent text field assigned with non-HTML (sanitized) data.
  In case the `url` field is used with this purpose, `contentType` MUST be set to `"text/html"` and the URI must be properly encoded. For example:
  - HTML data:

```
<div>This is the solution for sending HTML we were looking for!<div/>
```

  ◦ Encoded string:

    ```
    "url":
    "data:text/html;charset:utf8,%3Cdiv%3EThis%20is%20the%20solution%20for%20sending%20H
    TML%20we%20were%20looking%20for%21%3Cdiv%2F%3E"
    ```

- `width`: a number representing the width of an image or a video in pixels. Nullable. Positive value.
  If the media object is neither an image or a video, i.e., if `contentType` contains a substring following the pattern `"^(application|audio|font|example|message|model|multipart|text)"`, `width` MUST be set to null.
  For example, the `width` of a Full-HD video is expected to be `1920` pixels, while the `width` of a 4K video is expected to be `3840` pixels.

A summary of the `attributes` object is presented in the table below:

Field	Type	Constraints
abstract	text	Nullable
author	string	Nullable
contentType	string	Non-nullable, Regular Expression
description	text	Nullable, Conditional Assignment
duration	number	Nullable, Conditional Assignment, Unit of Measure, Greater than Zero
height	number	Nullable, Conditional Assignment, Unit of Measure, Greater than Zero
license	string	Nullable
name	text	Nullable, Conditional Assignment
shortName	text	Nullable
url	url	Non-nullable
width	number	Nullable, Conditional Assignment, Unit of Measure, Greater than Zero

Relationships

The `relationships` object of a media object resource MUST contain the following fields:

- `categories`: a Reference to Many category resources that are instantiated by the media object. See Section Categories. Nullable. Non-empty.

  No category is pre-defined by the standard.

- `licenseHolder`: a Reference to One agent resource (see agent) who holds the rights over the media object. Nullable.

A summary of the `relationships` object is presented in the table below:

Field	Type	Constraints
categories	Reference to Many object to Categories	Nullable, Non-empty
licenseHolder	Reference to One object to Agents	Nullable

Links

See the definition of the `links` object in Links Object.

Examples

The following example presents an object containing the minimal information required of a media object resource:

```json
{
  "type": "mediaObjects",
  "id": "1",
  "meta": {
    "dataProvider": null,
    "lastUpdate": null
  },
  "attributes": {
    "abstract": null,
    "author": null,
    "contentType": "audio/mpeg3",
    "description": null,
    "duration": null,
    "height": null,
    "license": null,
    "name": null,
    "shortName": null,
    "url": "https://example.com/audio.mp3",
    "width": null
  },
  "relationships": {
    "categories": null,
    "licenseHolder": null
  },
  "links": {
    "self": "https://example.com/2022-04/mediaObjects/1"
  }
}
```

`asciidoc/examples/mediaobject.min.json`

The following example presents an object representing a media object resource:

```
{
  "type": "mediaObjects",
  "id": "1",
  "meta": {
    "dataProvider": "https://example.com",
    "lastUpdate": "2020-04-01T08:00:00+02:00"
  },
  "attributes": {
    "abstract": {
      "eng": "My first image sent as an example for the API."
    },
    "author": "Leonardo da Vinci",
    "contentType": "image/png",
    "description": {
      "eng": "My first image sent as an example for the API."
    },
    "duration": null,
    "height": 300,
    "license": "FreeImage",
    "name": {
      "ita": "La mia immagine",
      "deu": "Mein bild",
      "eng": "My image"
    },
    "shortName": {
      "eng": "My image"
    },
    "url": "https://example.com/image.png",
    "width": 400
  },
  "relationships": {
    "categories": {
      "data": [
        {
          "type": "categories",
          "id": "example:panoramic-photo"
        }
      ],
      "links": {
        "related": "https://example.com/2022-04/mediaObjects/1/categories"
      }
    },
    "licenseHolder": {
      "data": {
        "type": "agents",
        "id": "0"
      },
      "links": {
        "related": "https://example.com/2022-04/mediaObjects/1/copyrgihtOwner/"
      }
    }
  },
  "links": {
    "self": "https://example.com/2022-04/mediaObjects/1"
  }
}
```

asciidoc/examples/mediaobject.full.json

7.2.8. Mountain Areas

A resource that implements the concept of Mountain Area defined in the **AlpineBits® DestinationData Ontology**.

A JSON object representing such a resource MUST contain the following fields:

- `type`: the constant `"mountainAreas"` that identifies the resource as being a mountain area.

- `id`: a string that uniquely and persistently identifies the mountain area within a SERVER. See the definition in Basic Fields.
- `attributes`: an object containing the attributes of the mountain area.
- `relationships`: an object containing the relationships of the mountain area to other resources.
- `links`: an object containing the links related to the mountain area.

A mountain area resource is structured as follows:

```
{
  "type": "mountainAreas",
  "id": "1",
  "meta": { ... },
  "attributes": { ... },
  "relationships": { ... },
  "links": { ... }
}
```

Meta

See the definition of the `meta` object in Meta Object.

Attributes

The `attributes` object of a mountain area resource MUST contain the following fields:

- `abstract`: a text object containing a brief description of the mountain area. Nullable. See the definition in Attributes Object.
- `area`: a number representing the total area, in square meters, of the mountain area. Nullable.
- `description`: * `description`: a text object containing a description of the mountain area. Nullable. Conditional Assignment. See the definition in Attributes Object.
- `geometries`: an array of geometry objects each of which represents the location of the mountain area in terms of GPS coordinates. There should be at most one geometry object of each type (e.g. Point, LineString). Nullable. Non-empty.
- `howToArrive`: a text object containing instructions on how to arrive at the mountain area. Nullable.
- `maxAltitude`: a number representing the highest elevation point of the mountain area in meters above sea level. Nullable.
- `minAltitude`: a number representing the lowest elevation point of the mountain area in meters above sea level. Nullable.
- `name`: a text object containing the complete name of the mountain area. Non-nullable. Conditional Assignment. See the definition in Attributes Object.
- `openingHours`: an hours specification object representing the hours in which the mountain area is open to the public. Nullable.
- `shortName`: a text object containing a short name of the mountain area. Nullable. See the definition in Attributes Object.
- `snowCondition`: a snow condition object containing the latest reported condition of the snow in the mountain area. Nullable.
- `totalParkLength`: an integer representing the total length, in meters, of all snowparks located within the mountain area. Nullable.
- `totalSlopeLength`: an integer representing the total length, in meters, of all ski slopes located within the mountain area. Nullable.
- `url`: a url object or string describing the mountain area, such as a website or a Wikipedia page. Nullable. See the definition in Attributes Object.

A summary of the `attributes` object is presented in the table below:

Field	Type	Constraints
abstract	text	Nullable
area	number	Nullable, Unit of Measure, Greater than Zero
description	text	Nullable
geometries	Array of geometry	Nullable, Non-empty
howToArrive	text	Nullable
maxAltitude	number	Nullable, Unit of Measure
minAltitude	number	Nullable, Unit of Measure
name	url	Non-nullable
openingHours	hours specification	Nullable
shortName	text	Nullable
snowCondition	snow condition	Nullable
totalParkArea	number	Nullable, Unit of Measure, Greater than Zero
totalTrailLength	number	Nullable, Unit of Measure, Greater than Zero
url	url	Nullable

Relationships

The `relationships` object of a mountain area resource MUST contain the following fields:

- `areaOwner`: a Reference to One agent resource (see Agents) who owns the mountain area. Nullable.
- `categories`: a Reference to Many category resources that are instantiated by the mountain area. See Section Categories. Nullable. Non-empty.

  No category is pre-defined by the standard.
- `connections`: a Reference to Many place resources that identify the places that are physically accessible from the mountain area, which may include other Mountain Areas, Lifts, Snowparks, and Ski Slopes. Nullable. Non-empty.

  Notice that connections between place resources may not be symmetrical (i.e., bidirectional). For example, a place like a lift may give access to a snowpark, but the snowpark may not give access back to it.
- `lifts`: a Reference to Many lift resources (see Lifts) that identify the lifts located within the mountain area. Nullable. Non-empty.
- `multimediaDescriptions`: a Reference to Many media object resources (see Media Objects) that are related to the mountain area. See Section multimediaDescriptions. Nullable. Non-empty.
- `skiSlopes`: a Reference to Many ski slope resources (see Ski Slopes) that identify the slopes located within the mountain area. Nullable. Non-empty.
- `snowparks`: a Reference to Many snowpark resources (see Snowparks) that identify the snowparks located within the mountain area. Nullable. Non-empty.
- `subAreas`: a Reference to Many mountain area resources (see Mountain Areas) that identify the mountain areas located within the mountain area. Nullable. Non-empty.

A summary of the relationships is presented in the table below:

Field	Type	Constraints
areaOwner	Reference to One object to Agents	Nullable, Non-empty
categories	Reference to Many object to Categories	Nullable, Non-empty
connections	Reference to Many object to Mountain Areas, Ski Slopes, Lifts, and Snowparks	Nullable, Non-empty
lifts	Reference to Many object to Lifts	Nullable, Non-empty
multimediaDescriptions	Reference to Many object to Media Objects	Nullable, Non-empty
snowparks	Reference to Many object to Snowparks	Nullable, Non-empty
subAreas	Reference to Many object to Mountain Areas	Nullable, Non-empty
skiSlopes	Reference to Many object to Ski Slopes	Nullable, Non-empty

Links

See the definition of the `links` object in Links Object.

Examples

The following example contains the minimal information required for a mountain area resource:

```json
{
  "id": "1",
  "type": "mountainAreas",
  "meta": {
    "dataProvider": null,
    "lastUpdate": null
  },
  "attributes": {
    "name": {
      "eng": "Meran 2000"
    },
    "shortName": null,
    "description": null,
    "abstract": null,
    "url": null,
    "geometries": null,
    "howToArrive": null,
    "openingHours": null,
    "area": null,
    "minAltitude": null,
    "maxAltitude": null,
    "totalTrailLength": null,
    "totalParkArea": null,
    "totalParkLength": null,
    "snowCondition": null
  },
  "relationships": {
    "areaOwner": null,
    "connections": null,
    "categories": null,
    "multimediaDescriptions": null,
    "lifts": null,
    "skiSlopes": null,
    "snowparks": null,
    "subAreas": null
  },
  "links": {
    "self": "https://example.com/2022-04/mountainAreas/1"
  }
}
```

asciidoc/examples/mountain.min.json

The following example illustrates the fields defined for mountain area resources:

```json
{
  "id": "1",
  "type": "mountainAreas",
  "meta": {
    "dataProvider": "https://example.com",
    "lastUpdate": "2020-04-01T08:00:00+02:00"
  },
  "attributes": {
    "name": {
      "ita": "Merano 2000",
      "deu": "Meran 2000",
      "eng": "Meran 2000"
    },
    "shortName": {
      "eng": "Meran 2000"
    },
    "description": {
      "deu": "Das Skigebiet Meran 2000 liegt unter dem Berg Ifinger im Burggrafenamt auf
einem Hochplateau oberhalb Meran am Tschögglberg in Südtirol. Es hat 45 km Alpin-Pisten
und reicht von 1670 bis 2300 m Höhe. Von Meran aus ist das Gebiet direkt über die Ifinger-
Seilbahn ab Naif oder durch eine Umlaufseilbahn ab Falzeben erreichbar. Das Skigebiet
erstreckt sich hauptsächlich auf dem Gemeindegebiet von Hafling, berührt aber auch zu den
```

```
Gemeinden Schenna und Sarntal gehörende Flächen."
    },
    "abstract": {
      "deu": "Das Skigebiet Meran 2000 liegt unter dem Berg Ifinger im Burggrafenamt auf
einem Hochplateau oberhalb Meran am Tschögglberg in Südtirol. Es hat 45 km Alpin-Pisten
und reicht von 1670 bis 2300 m Höhe."
    },
    "url": "https://www.meran2000.com",
    "geometries": [
      {
        "type": "Polygon",
        "coordinates": [
          [
            [
              11.310853958129883,
              46.66958283253642
            ],
            [
              11.304588317871094,
              46.668817160723044
            ],
            [
              11.301412582397461,
              46.666696782172096
            ],
            [
              11.305532455444336,
              46.66457632044435
            ],
            [
              11.31265640258789,
              46.66646117942096
            ],
            [
              11.314373016357422,
              46.66869936409677
            ],
            [
              11.310853958129883,
              46.66958283253642
            ]
          ]
        ]
      }
    ],
    "howToArrive": {
      "ita": "L'area sciistica ed escursionistica di Merano è situata ai piedi della
montagna Picco Ivigna ed è raggiungibile in pochi minuti dalle destinazioni di Merano,
Avelengo, Scena e Tirolo. La cima di Merano 2000 è raggiungibile con due impianti di
risalita diversi e ha dunque due stazioni a valle, una presso Merano con la Funivia e una
ad Avelengo con la Cabinovia Falzeben.",
      "deu": "Die Sonnenterrasse Merans liegt am Fuße des Ifingers und ist von den
Ferienorten Meran, Hafling, Schenna und Dorf Tirol in wenigen Minuten leicht erreichbar.
Die Bergstation von Meran 2000 kann man mit zwei verschiedenen Aufstiegsanlagen erreichen:
von Meran aus mit der Bergbahn und von Hafling aus mit der Umlaufbahn Falzeben.",
      "eng": "The skiing and hiking area of Merano 2000 is best located next to the
biggest vacation hotspots of South Tyrol and so reachable within few minutes from Merano,
Avelengo, Scena and Tirolo. Two lifts can bring you to the mountain station of Merano
2000: the Ropeway in Merano or the Gondola Falzeben in Avelengo."
    },
    "openingHours": {
      "dailySchedules": {
        "2020-12-25": null
      },
      "weeklySchedules": [
        {
          "validFrom": "2020-01-01",
          "validTo": "2020-12-31",
          "monday": [
            {
              "opens": "08:00:00+01:00",
              "closes": "18:00:00+01:00"
            }
          ],
          "tuesday": [
            {
              "opens": "08:00:00+01:00",
              "closes": "18:00:00+01:00"
            }
          ],
          "wednesday": [
            {
              "opens": "08:00:00+01:00",
              "closes": "18:00:00+01:00"
            }
          ],
          "thursday": [
            {
              "opens": "08:00:00+01:00",
              "closes": "18:00:00+01:00"
            }
          ],
          "friday": [
            {
              "opens": "08:00:00+01:00",
              "closes": "18:00:00+01:00"
            }
          ],
          "saturday": [
            {
              "opens": "08:00:00+01:00",
              "closes": "18:00:00+01:00"
            }
          ],
          "sunday": [
            {
              "opens": "08:00:00+01:00",
              "closes": "18:00:00+01:00"
            }
          ]
        }
      ]
    },
    "area": 36000,
    "minAltitude": 1200,
    "maxAltitude": 2000,
    "totalTrailLength": 4000,
    "totalParkArea": 20000,
    "totalParkLength": 1000,
    "snowCondition": {
      "primarySurface": "powder",
      "secondarySurface": "packed-powder",
      "baseSnow": 50,
      "baseSnowRange": {
        "lower": 40,
        "upper": 60
      },
      "latestStorm": 40,
      "obtainedIn": "2019-12-20",
      "snowOverNight": 5,
      "groomed": true,
      "snowMaking": false
    }
  },
  "relationships": {
    "areaOwner": {
      "data": {
        "type": "agents",
        "id": "1"
      },
        "links": {
          "related": "https://example.com/2022-04/mountainAreas/1/areaOwner/"
        }
      },
      "connections": {
        "data": [
          {
            "type": "lifts",
            "id": "1"
          },
          {
            "type": "skiSlopes",
            "id": "1"
          }
        ],
        "links": {
          "related": "https://example.com/2022-04/mountainAreas/1/connections/"
        }
      },
      "categories": {
        "data": [
          {
            "type": "categories",
            "id": "example:skiarea"
          }
        ],
        "links": {
          "related": "https://example.com/2022-04/mountainAreas/1/categories"
        }
      },
      "multimediaDescriptions": {
        "data": [
          {
            "type": "mediaObjects",
            "id": "1"
          }
        ],
        "links": {
          "related": "https://example.com/2022-04/mountainAreas/1/multimediaDescriptions/"
        }
      },
      "lifts": {
        "data": [
          {
            "type": "lifts",
            "id": "1"
          }
        ],
        "links": {
          "related": "https://example.com/2022-04/mountainAreas/1/lifts/"
        }
      },
      "skiSlopes": {
        "data": [
          {
            "type": "skiSlopes",
            "id": "1"
          }
        ],
        "links": {
          "related": "https://example.com/2022-04/mountainAreas/1/skiSlopes/"
        }
      },
      "snowparks": {
        "data": [
          {
            "type": "snowparks",
            "id": "1"
          }
        ],
      "links": {
        "related": "https://example.com/2022-04/mountainAreas/1/snowparks/"
      }
    },
    "subAreas": {
      "data": [
        {
          "type": "mountainAreas",
          "id": "2"
        },
        {
          "type": "mountainAreas",
          "id": "3"
        }
      ],
      "links": {
        "related": "https://example.com/2022-04/mountainAreas/1/subAreas/"
      }
    }
  },
  "links": {
    "self": "https://example.com/2022-04/mountainAreas/1"
  }
}
```

asciidoc/examples/mountain.full.json

7.2.9. Ski Slopes

A resource that implements the concept of Ski Slope defined in the **AlpineBits® DestinationData Ontology**.

A JSON object representing such a resource MUST contain the following fields:

- `type`: the constant `"skiSlopes"` that identifies the resource as being of the type ski slope.
- `id`: a string that uniquely and persistently identifies the ski slope within a SERVER. See the definition in Basic Fields.
- `attributes`: an object containing the attributes of the ski slope.
- `relationships`: an object containing the relationships of the ski slope to other resources.
- `links`: an object containing the links related to the ski slope.

A ski slope resource is structured as follows:

```
{
  "type": "skiSlopes",
  "id": "1",
  "meta": { ... },
  "attributes": { ... },
  "relationships": { ... },
  "links": { ... }
}
```

Meta

See the definition of the `meta` object in Meta Object.

Attributes

The `attributes` object of a ski slope resource MUST contain the following fields:

- `abstract`: a text object containing a brief description for the ski slope. Nullable. See the definition in Attributes Object.

- `address`: an address object representing the main address of the ski slope. The address should only be set if the ski slope is directly accessible from a public road. Nullable.
- `description`: a text object containing a description of the ski slope. Nullable. Conditional Assignment. See the definition in Attributes Object.
- `difficulty`: an object containing the difficulty level of a ski slope, as in `{ "eu": "novice", "us": "beginner" }`. Nullable.

  The `"eu"` field refers to the European classification of ski slopes. Its possible values are `"novice"`, `"beginner"`, `"intermediate"`, and `"expert"`. The `"us"` field refers to the North American classification of ski slopes. Its possible values are `"beginner"`, `"beginner-intermediate"`, `"intermediate"`, `"intermediate-advanced"`, and `"expert"`.

  Every such object MUST contain both fields and at least one of them MUST NOT be null. The field itself, however, is nullable.

- `geometries`: an array of geometry objects each of which represents the location of the ski slope in terms of GPS coordinates. There should be at most one geometry object of each type (e.g. Point, LineString). Nullable. Non-empty.
- `howToArrive`: a text object containing instructions on how to arrive at the ski slope. Nullable.
- `length`: a number representing the total length of the ski slope in meters. The computation of the length (in the case of branching paths, for instance) is determined by the data provider. Nullable.
- `maxAltitude`: a number representing the highest elevation point of the ski slope in meters above sea level. Nullable.
- `minAltitude`: a number representing the lowest elevation point of the ski slope in meters above sea level. Nullable.
- `name`: a text object containing the complete name of the ski slope. Non-nullable. See the definition in Attributes Object.
- `openingHours`: an hours specification object representing the hours in which the ski slope is open to the public. Nullable.
- `shortName`: a text object containing a short name of the ski slope. Nullable. See the definition in Attributes Object.
- `snowCondition`: a snow condition object containing the latest reported condition of the snow in the ski slope.
- `url`: a url object or string describing the ski slope, such as a website or a Wikipedia page. Nullable. See the definition in Attributes Object.

A summary of the attributes is presented in the table below:

Field	Type	Constraints
abstract	text	Nullable
address	address	Nullable
description	text	Nullable, Conditional Assignment
difficulty	object	Nullable
geometries	Array of geometry	Nullable, Non-empty
howToArrive	text	Nullable
length	number	Nullable, Unit of Measure, Greater than Zero
maxAltitude	number	Nullable, Unit of Measure
minAltitude	number	Nullable, Unit of Measure
name	text	Non-nullable, Conditional Assignment

Field	Type	Constraints
openingHours	hours specification	Nullable
shortName	text	Nullable
snowCondition	snow condition	Nullable
url	url	Nullable

Relationships

The `relationships` object of a ski slope resource MUST contain the following fields:

- `categories`: a Reference to Many category resources that are instantiated by the ski slope. See Section Categories. Nullable. Non-empty.

  The standard recommends the following categories for ski slopes:, `"alpinebits:standard-ski-slope"`, `"alpinebits:sledge-slope"`, and `"alpinebits:cross-country"`

- `connections`: a Reference to Many place resources that identify the places that are physically accessible from the mountain area, which may include other Ski Slopes, Lifts, Snowparks, and Mountain Areas. Nullable. Non-empty.

  Notice that connections between place resources may not be symmetrical (i.e., bidirectional). For example, a place like a lift may give access to a snowpark, but the snowpark may not give access back to it.

- `multimediaDescriptions`: a Reference to Many media object resources (see Media Objects) that are related to the ski slope. See Section multimediaDescriptions. Nullable. Non-empty.

A summary of the relationships is presented in the table below:

Field	Type	Constraints
categories	Reference to Many object to Categories	Nullable, Non-empty
connections	Reference to Many object to Mountain Areas, Ski Slopes, Lifts, and Snowparks	Nullable, Non-empty
multimediaDescriptions	Reference to Many object to Media Objects	Nullable, Non-empty

Links

See the definition of the `links` object in Links Object.

Examples

The following example contains the minimal information required for a ski slope resource:

```
{
  "type": "skiSlopes",
  "id": "1",
  "meta": {
    "dataProvider": null,
    "lastUpdate": null
  },
  "attributes": {
    "name": {
      "eng": "Falzeben I"
    },
    "shortName": null,
    "description": null,
    "abstract": null,
    "url": null,
    "length": null,
    "minAltitude": null,
    "maxAltitude": null,
    "difficulty": null,
    "address": {
      "street": null,
      "city": {
        "eng": "Merano"
      },
      "region": null,
      "country": "IT",
      "zipcode": null,
      "categories": null,
      "complement": null
    },
    "geometries": null,
    "howToArrive": null,
    "openingHours": null,
    "snowCondition": null
  },
  "relationships": {
    "connections": null,
    "categories": null,
    "multimediaDescriptions": null
  },
  "links": {
    "self": "https://example.com/2022-04/skiSlopes/1"
  }
}
```

asciidoc/examples/skislope.min.json

The following example illustrates the fields defined for ski slope resources:

```
{
  "type": "skiSlopes",
  "id": "1",
  "meta": {
    "dataProvider": "https://example.com",
    "lastUpdate": "2020-04-01T08:00:00+02:00"
  },
  "attributes": {
    "name": {
      "ita": "Falzeben I",
      "deu": "Falzeben I",
      "eng": "Falzeben I"
    },
    "shortName": {
      "ita": "Falzeben I"
    },
    "description": {
      "ita": "Falzeben I -  Discesa tra i boschi facile, immersa in un panorama da sogno
```

```
ed ideale per bambini e principianti. Altezza/Lunghezza: 1900 altezza, 3500 m."
    },
    "abstract": {
      "ita": "Falzeben I -  Discesa tra i boschi facile, immersa in un panorama da sogno
ed ideale per bambini e principianti."
    },
    "url": "https://example.com",
    "length": 2000,
    "minAltitude": 1500,
    "maxAltitude": 2500,
    "difficulty": {
      "eu": "novice",
      "us": "beginner"
    },
    "address": {
      "street": null,
      "city": {
        "eng": "Merano"
      },
      "region": null,
      "country": "IT",
      "zipcode": null,
      "categories": null,
      "complement": null
    },
    "howToArrive": {
      "ita": " Le fermate più vicine a Falzeben sono: Falzeben è a 41 metri di distanza a
piedi e ci si arriva in 1 minuti di cammino; Villa Schäfer è a 808 metri di distanza a
piedi e ci si arriva in 11 minuti di cammino."
    },
    "geometries": [
      {
        "type": "LineString",
        "coordinates": [
          [
            11.305682659149168,
            46.66705018437341
          ],
          [
            11.30692720413208,
            46.667182709603225
          ],
          [
            11.308064460754393,
            46.667491933875965
          ]
        ]
      }
    ],
    "openingHours": {
      "dailySchedules": {
        "2020-12-25": null
      },
      "weeklySchedules": [
        {
          "validFrom": "2020-01-01",
          "validTo": "2020-12-31",
          "monday": [
            {
              "opens": "08:00:00+01:00",
              "closes": "18:00:00+01:00"
            }
          ],
          "tuesday": [
            {
              "opens": "08:00:00+01:00",
              "closes": "18:00:00+01:00"
            }
          ],
          "wednesday": [
          {
            "opens": "08:00:00+01:00",
            "closes": "18:00:00+01:00"
          }
        ],
        "thursday": [
          {
            "opens": "08:00:00+01:00",
            "closes": "18:00:00+01:00"
          }
        ],
        "friday": [
          {
            "opens": "08:00:00+01:00",
            "closes": "18:00:00+01:00"
          }
        ],
        "saturday": [
          {
            "opens": "08:00:00+01:00",
            "closes": "18:00:00+01:00"
          }
        ],
        "sunday": [
          {
            "opens": "08:00:00+01:00",
            "closes": "18:00:00+01:00"
          }
        ]
      }
    ]
  },
  "snowCondition": {
    "primarySurface": "powder",
    "secondarySurface": "packed-powder",
    "baseSnow": 50,
    "baseSnowRange": {
      "lower": 40,
      "upper": 60
    },
    "latestStorm": 40,
    "obtainedIn": "2020-02-01",
    "snowOverNight": 5,
    "groomed": true,
    "snowMaking": false
  }
},
"relationships": {
  "connections": {
    "data": [
      {
        "type": "skiSlopes",
        "id": "2"
      },
      {
        "type": "skiSlopes",
        "id": "3"
      },
      {
        "type": "skiSlopes",
        "id": "4"
      },
      {
        "type": "lifts",
        "id": "1"
      }
    ],
    "links": {
      "related": "https://example.com/2022-04/skiSlopes/1/connections/"
    }
```

```
      },
      "categories": {
        "data": [
          {
            "type": "categories",
            "id": "alpinebits:standard-ski-slope"
          }
        ],
        "links": {
          "related": "https://example.com/2022-04/skiSlopes/1/categories"
        }
      },
      "multimediaDescriptions": {
        "data": [
          {
            "type": "mediaObjects",
            "id": "2"
          }
        ],
        "links": {
          "related": "https://example.com/2022-04/skiSlopes/1/multimediaDescriptions/"
        }
      }
    },
    "links": {
      "self": "https://example.com/2022-04/skiSlopes/1"
    }
  }
```

asciidoc/examples/skislope.full.json

7.2.10. Snowparks

A resource that implements the concept of Snowpark defined in the **AlpineBits® DestinationData Ontology**.

A JSON object representing such a resource MUST contain the following fields:

- `type`: the constant `"snowparks"` that identifies the resource as being of the type snowpark.
- `id`: a string that uniquely and persistently identifies the snowpark within a SERVER. See the definition in Basic Fields.
- `attributes`: an object containing the attributes of the snowpark.
- `relationships`: an object containing the relationships of the snowpark to other resources.
- `links`: an object containing the links related to the snowpark.

A snowpark resource is structured as follows:

```
{
  "type": "snowparks",
  "id": "1",
  "meta": { ... },
  "attributes": { ... },
  "relationships": { ... },
  "links": { ... }
}
```

Meta

See the definition of the `meta` object in Meta Object.

Attributes

The `attributes` object of a snowpark resource MUST contain the following fields:

- `abstract`: a text object containing a brief description for the snowpark. Nullable. See the definition in Attributes Object.
- `address`: an address object representing the address of the snowpark. The address should only be set if the snowpark is directly accessible from a public road. Nullable.
- `description`: a text object containing a description of the snowpark. Nullable. Conditional Assignment. See the definition in Attributes Object.
- `difficulty`: a string describing the difficulty level of a snowpark according to the following enumerated values:
    - `"beginner"`
    - `"intermediate"`
    - `"advanced"`
    - `"expert"`
- `geometries`: an array of geometry objects each of which represents the location of the snowpark in terms of GPS coordinates. There should be at most one geometry object of each type (e.g. Point, LineString). Nullable. Non-empty.
- `howToArrive`: a text object containing instructions on how to arrive at the snowpark. Nullable.
- `length`: a number representing the total length of the snowpark in meters. The computation of the length (in the case of branching paths, for instance) is determined by the data provider. Nullable.
- `maxAltitude`: a number representing the highest elevation point of the snowpark in meters above sea level. Nullable.
- `minAltitude`: a number representing the lowest elevation point of the snowpark in meters above sea level. Nullable.
- `name`: a text object containing the complete name of the snowpark. Non-nullable. Conditional Assignment. See the definition in Attributes Object.
- `openingHours`: an hours specification object representing the hours in which the snowpark is open to the public. Nullable.
- `shortName`: a text object containing a short name of the snowpark. Nullable. See the definition in Attributes Object.
- `snowCondition`: a snow condition object containing the latest reported condition of the snow in the snowpark.
- `url`: a url object or string describing the snowpark, such as a website or a Wikipedia page. Nullable. See the definition in Attributes Object.

A summary of the attributes is presented in the table below:

Field	Type	Constraints
abstract	text	Nullable
address	address	Nullable
description	text	Nullable, Conditional Assignment
difficulty	string	Enumeration, Nullable
geometries	Array of geometry	Nullable, Non-empty
howToArrive	string	Nullable
length	number	Nullable, Unit of Measure, Greater than Zero
maxAltitude	number	Nullable, Unit of Measure
minAltitude	number	Nullable, Unit of Measure
name	text	Non-nullable, Conditional Assignment

Field	Type	Constraints
openingHours	hours specification	Nullable
shortName	text	Nullable
snowCondition	snow condition	Nullable
url	url	Nullable

Relationships

The `relationships` object of a snowpark resource MUST contain the following fields:

- `categories`: a Reference to Many category resources that are instantiated by the snowpark. See Section Categories. Nullable. Non-empty.

  No category is pre-defined by the standard.

- `connections`: a Reference to Many place resources that identify the places that are physically accessible from the mountain area, which may include other Snowparks, Lifts, Mountain Areas, and Ski Slopes. Nullable. Non-empty.

  Notice that connections between place resources may not be symmetrical (i.e., bidirectional). For example, a place like a lift may give access to a snowpark, but the snowpark may not give access back to it.

- `features`: a Reference to Many feature resources present in a snowpark, such as a ramp or a rail. See Section Features. Nullable. Non-empty.

  No feature is pre-defined by the standard.

- `multimediaDescriptions`: a Reference to Many media object resources (see Media Objects) that are related to the snowpark. See Section multimediaDescriptions. Nullable. Non-empty.

A summary of the relationships is presented in the table below:

Field	Type	Constraints
categories	Reference to Many object to Categories	Nullable, Non-empty
connections	Reference to Many object to Mountain Areas, Ski Slopes, Lifts, and Snowparks	Nullable, Non-empty
features	Reference to Many object to Features	Nullable, Non-empty
multimediaDescriptions	Reference to Many object to Media Objects	Nullable, Non-empty

Links

See the definition of the `links` object in Links Object.

Examples

The following example contains the minimal information required for a snowpark resource:

```
{
  "type": "snowparks",
  "id": "1",
  "meta": {
    "dataProvider": null,
    "lastUpdate": null
  },
  "attributes": {
    "name": {
      "eng": "Snowpark Merano 2000"
    },
    "shortName": null,
    "description": null,
    "abstract": null,
    "url": null,
    "length": null,
    "minAltitude": null,
    "maxAltitude": null,
    "address": {
      "street": null,
      "city": {
        "eng": "Merano"
      },
      "region": null,
      "country": "IT",
      "zipcode": null,
      "categories": null,
      "complement": null
    },
    "howToArrive": null,
    "difficulty": null,
    "geometries": null,
    "openingHours": null,
    "snowCondition": null
  },
  "relationships": {
    "connections": null,
    "features": null,
    "categories": null,
    "multimediaDescriptions": null
  },
  "links": {
    "self": "https://example.com/2022-04/snowparks/1"
  }
}
```

asciidoc/examples/snowpark.min.json

The following example illustrates the fields defined for snowpark resources:

```
{
  "type": "snowparks",
  "id": "1",
  "meta": {
    "dataProvider": "https://example.com",
    "lastUpdate": "2020-04-01T08:00:00+02:00"
  },
  "attributes": {
    "name": {
      "ita": "Snowpark Merano 2000",
      "deu": "Snowpark Merano 2000",
      "eng": "Snowpark Merano 2000"
    },
    "shortName": {
      "eng": "Snowpark Merano 2000"
    },
    "description": {
      "eng": "Located in the rear part of the skiing area on the Oswald-Slope, close to
the Waidmann Alpine Cottage, the SNOWPARK MERANO 2000 awaits brave Snowboarders and
Freestylers. An ambitious project with many kicks, rails, tubes and boxes for newcomers
and pros alike. You can reach the Jump Zone from the mountain station of the Ropeway by
using the chairlift Piffing."
    },
    "abstract": {
      "eng": "Located in the rear part of the skiing area on the Oswald-Slope, close to
the Waidmann Alpine Cottage, the SNOWPARK MERANO 2000 awaits brave Snowboarders and
Freestylers..."
    },
    "url": "https://example.com",
    "length": 1300,
    "minAltitude": 1500,
    "maxAltitude": 2500,
    "address": {
      "street": null,
      "city": {
        "eng": "Merano"
      },
      "region": null,
      "country": "IT",
      "zipcode": null,
      "categories": null,
      "complement": null
    },
    "howToArrive": null,
    "difficulty": "intermediate",
    "geometries": [
      {
        "type": "LineString",
        "coordinates": [
          [
            11.305682659149168,
            46.66705018437341
          ],
          [
            11.30692720413208,
            46.667182709603225
          ],
          [
            11.308064460754393,
            46.667491933875965
          ]
        ]
      }
    ],
    "openingHours": {
      "dailySchedules": {
        "2020-12-25": null
      },
      "weeklySchedules": [
        {
          "validFrom": "2020-01-01",
          "validTo": "2020-12-31",
          "monday": [
            {
              "opens": "08:00:00+01:00",
              "closes": "18:00:00+01:00"
            }
          ],
          "tuesday": [
            {
              "opens": "08:00:00+01:00",
              "closes": "18:00:00+01:00"
            }
          ],
          "wednesday": [
            {
              "opens": "08:00:00+01:00",
              "closes": "18:00:00+01:00"
            }
          ],
          "thursday": [
            {
              "opens": "08:00:00+01:00",
              "closes": "18:00:00+01:00"
            }
          ],
          "friday": [
            {
              "opens": "08:00:00+01:00",
              "closes": "18:00:00+01:00"
            }
          ],
          "saturday": [
            {
              "opens": "08:00:00+01:00",
              "closes": "18:00:00+01:00"
            }
          ],
          "sunday": [
            {
              "opens": "08:00:00+01:00",
              "closes": "18:00:00+01:00"
            }
          ]
        }
      ]
    },
    "snowCondition": {
      "primarySurface": "frozen-granular",
      "secondarySurface": "packed-powder",
      "baseSnow": 30,
      "baseSnowRange": {
        "lower": 25,
        "upper": 40
      },
      "latestStorm": 5,
      "obtainedIn": "2020-01-14",
      "snowOverNight": 0,
      "groomed": true,
      "snowMaking": false
    }
  },
  "relationships": {
    "connections": {
      "data": [
        {
          "type": "skiSlopes",
          "id": "2"
        },
        {
          "type": "skiSlopes",
          "id": "3"
        }
      ],
      "links": {
        "related": "https://example.com/2022-04/snowparks/1/connections/"
      }
    },
    "features": {
      "data": [
        {
          "type": "features",
          "id": "example:jib"
        },
        {
          "type": "features",
          "id": "example:pipe"
        }
      ],
      "links": {
        "related": "https://example.com/2022-04/snowparks/1/features"
      }
    },
    "categories": {
      "data": [
        {
          "type": "categories",
          "id": "example:free-style"
        }
      ],
      "links": {
        "related": "https://example.com/2022-04/snowparks/1/categories"
      }
    },
    "multimediaDescriptions": {
      "data": [
        {
          "type": "mediaObjects",
          "id": "2"
        }
      ],
      "links": {
        "related": "https://example.com/2022-04/snowparks/1/multimediaDescriptions/"
      }
    }
  },
  "links": {
    "self": "https://example.com/2022-04/snowparks/1"
  }
}
```

asciidoc/examples/snowpark.full.json

7.2.11. Venues

A resource that implements the concept of Venue defined in the **AlpineBits® DestinationData Ontology**.

A JSON object representing such a resource MUST contain the following fields:

- `type`: the constant `"venues"` that identifies the resource as being of the type venue.
- `id`: a string that uniquely and persistently identifies the venue within a SERVER. See the definition in Basic Fields.
- `attributes`: an object containing the attributes of the venue.
- `relationships`: an object containing the relationships of the venue to other resources.
- `links`: an object containing the links related to the venue.

Venue resources are structured in the following way:

```
{
  "type": "venues",
  "id": "1",
  "meta": { ... },
  "attributes": { ... },
  "relationships": { ... },
  "links": { ... }
}
```

Meta

See the definition of the `meta` object in Meta Object.

Attributes

The `attributes` object of the venue resource MUST contain the following fields:

- `abstract`: a text object containing a brief description of the venue. Nullable. See the definition in Attributes Object.
- `address`: an address object containing the address of the venue. Nullable.
- `description`: a text object containing a description of the venue. Nullable. Conditional Assignment. See the definition in Attributes Object.
- `geometries`: an array of geometry objects each of which represents the location of the venue in terms of GPS coordinates. There should be at most one geometry object of each type (e.g. Point, LineString). Nullable. Non-empty.
- `howToArrive`: a text object containing instructions on how to arrive at the venue. Nullable.
- `name`: a text object containing the complete name of the venue. Non-nullable. Conditional Assignment. See the definition in Attributes Object.
- `shortName`: a text object containing a short name of the venue. Nullable. See the definition in Attributes Object.
- `url`: a url object or string describing the venue, such as a website or a Wikipedia page. Nullable. See the definition in Attributes Object.

A summary of the `attributes` object is presented in the table below:

Field	Type	Constraints
abstract	text	Nullable
address	address	Nullable
description	text	Nullable, Conditional Assignment
geometries	Array of geometry	Nullable, Non-empty
howToArrive	string	Nullable
name	text	Non-nullable, Conditional Assignment
shortName	text	Nullable
url	url	Nullable

Relationships

The `relationships` object of a mountain area resource MUST contain the following fields:

- `categories`: a Reference to Many category resources that are instantiated by the venue. See Section Categories. Nullable. Non-empty.

  No category is pre-defined by the standard.

- `multimediaDescriptions`: a Reference to Many media object resources (see Media Objects) that are related to the venue. See Section multimediaDescriptions. Nullable. Non-empty.

A summary of the relationships is presented in the table below:

Field	Type	Constraints
categories	Reference to Many object to Categories	Nullable, Non-empty

Field	Type	Constraints
multimediaDescriptions	Reference to Many object to Media Objects	Nullable, Non-empty

Links

See the definition of the `links` object in Links Object.

Examples

he following example presents an object containing the minimal information required of a venue resource:

```
{
  "type": "venues",
  "id": "1",
  "meta": {
    "dataProvider": null,
    "lastUpdate": null
  },
  "attributes": {
    "abstract": null,
    "address": null,
    "description": null,
    "geometries": null,
    "howToArrive": null,
    "name": {
      "eng": "Auditorium 1"
    },
    "shortName": null,
    "url": null
  },
  "relationships": {
    "categories": null,
    "multimediaDescriptions": null
  },
  "links": {
    "self": "https://example.com/2022-04/venues/1"
  }
}
```

`asciidoc/examples/venue.min.json`

The following example presents an object representing a venue resource:

```
{
  "type": "venues",
  "id": "1",
  "meta": {
    "dataProvider": "https://example.com",
    "lastUpdate": "2020-04-01T08:00:00+02:00"
  },
  "attributes": {
    "abstract": {
      "eng": "The Auditorium 1 of the Free University of Bozen-Bolzano provides a great
space for keynotes, lectures and presentations."
    },
    "address": {
      "street": {
        "ita": "Piazza Università, 1",
        "deu": "Universitätsplatz 1"
      },
      "city": {
        "deu": "Bozen"
      },
      "region": {
        "deu": "Trentino-Südtirol"
      },
      "country": "IT",
      "zipcode": "39100",
      "complement": {
        "deu": "Hauptgebäude"
      },
      "categories": [
        "example:building"
      ]
    },
    "description": {
      "eng": "The Auditorium 1 of the Free University of Bozen-Bolzano provides a great
space for keynotes, lectures and presentations, being available to host events related
academic, provincial and cultural topics."
    },
    "geometries": [
      {
        "type": "Point",
        "coordinates": [
          11.35087251663208,
          46.49873937419277
        ]
      }
    ],
    "howToArrive": {
      "eng": "From the train station, the Free University of Bozen-Bolzano is accessible
in a 5 minutes walk into the historical city center."
    },
    "name": {
      "eng": "Auditorium 1 - Free University of Bozen-Bolzano"
    },
    "shortName": {
      "eng": "Auditorium 1"
    },
    "url": "https://example.com/auditorium-1"
  },
  "relationships": {
    "categories": {
      "data": [
        {
          "type": "categories",
          "id": "example:auditorium"
        }
      ],
      "links": {
        "related": "https://example.com/2022-04/venues/1/categories"
      }
    },
    "multimediaDescriptions": {
      "data": [
        {
          "type": "mediaObjects",
          "id": "1"
        }
      ],
      "links": {
        "related": "https://example.com/2022-04/venues/1/multimediaDescriptions"
      }
    }
  },
  "links": {
    "self": "https://example.com/2022-04/venues/1"
  }
}
```

asciidoc/examples/venue.full.json

Appendix A: AlpineBits® DestinationData developer resources

The **AlpineBits® DestinationData** development home page is at https://www.alpinebits.org/destinationdata/. There are resources linked from that page that help test one's implementation.

Public repositories with schema files and example code snippets are available online at https://gitlab.com/alpinebits/destinationdata. Contributions are welcome (any programming language).