python-delairstack Documentation

Release 1.7.8

Delair.ai Backend Team

Oct 07, 2020

CONTENTS

The Delair Python SDK offers to users a high-level interface to Delair.ai APIs.

CHAPTER

ONE

GETTING STARTED

1.1 First steps

1.1.1 Installation

Install the latest Delair Python package release via **pip**:

```
pip install python-delairstack
```

1.1.2 Configuration

To use the SDK, you must configure your Delair.ai credentials.

To do that, just create a `config-connection.json` configuration file:

```
{
  "user": "YOUR_EMAIL_ADDRESS",
  "password": "YOUR_PASSWORD_ON_DELAIR.AI"
}
```

Depending on your operating system, create the following directory and save the `config-connection.json` file in:

- **(Windows)** `%USERPROFILE%\Application\Application Data\Delair\python-delairstack`
- **(Linux)** `~/.local/share/python-delairstack/`
- **(MacOS)** `~/Library/Application Support/python-delairstack/`

Note: See the chapter on *Configuration* for more details about the configuration files.

1.1.3 Using the SDK

Any usage starts with the creation of a *delairstack.sdk.DelairStackSDK* instance:

```
>>> from delairstack import DelairStackSDK
>>> sdk = DelairStackSDK()
```

Note: Alternatively, if you don't want to use the SDK *with the credentials saved in the default configuration file*, you can pass your credentials directly, with:

```
>>> from delairstack import DelairStackSDK
>>> sdk = DelairStackSDK(url="https://www.delair.ai",
...                      user="YOUR_EMAIL_ADDRESS",
...                      password="YOUR_PASSWORD_ON_DELAIR.AI")
```

Get all the projects available

```
>>> projects = sdk.projects.search(name='*')
```

See the *projects.search()* documentation.

Get the missions of a project

```
>>> my_project = sdk.projects.search(name='My_project')[0]
>>> missions = sdk.missions.search(project=my_project.id)
```

See the *missions.search()* documentation.

Search for datasets related to a mission

```
>>> my_mission = missions[0]
>>> datasets = sdk.datasets.search(filter={'mission': {'$eq': my_mission.id}})
```

See the *datasets.search()* documentation.

Explore the dataset properties

Let's print some properties of a dataset:

```
>>> my_dataset = datasets[0]
>>> print("Name: {}".format(my_dataset.name))
>>> print("Type: {}".format(my_dataset.type))
>>> print("Creation date: {}".format(my_dataset.creation_date))
```

Some dataset properties depend on its type (`image`, `raster`, `mesh`, `pcl`, `vector`, `file`). You can list all the available properties for a dataset with:

```
>>> dir(my_dataset)
```

To look for the files related to a dataset, we can list the dataset components:

```
>>> print(my_dataset.components)
```

Download a dataset component

To download a dataset component in the current directory:

```
>>> component = my_dataset.components[0]
>>> sdk.datasets.download_component(dataset=my_dataset.id, component=component.get(
↪"name"))
```

See the *datasets.download_component()* documentation.

Create a new dataset

To create a new `file` dataset related to a project:

```
>>> new_dataset = sdk.datasets.create_file_dataset(name='My file dataset',
...                                                project=my_project.id)
```

See the *datasets.create_file_dataset()* documentation.

And upload a file:

```
>>> file_to_upload = "/replace/with/a/file_path.ext"
>>> sdk.datasets.upload_file(dataset=new_dataset.id,
...                          component='file',
...                          file_path=file_to_upload)
```

See the *datasets.upload_file()* documentation.

Add a tag

Let's add a tag on the dataset created.

```
>>> my_tag = sdk.tags.create(name='My tag',
...                          project=my_project.id,
...                          type='dataset',
...                          target=new_dataset.id)
```

See the *tags.create()* documentation.

This tag can be deleted with:

```
>>> sdk.tags.delete(my_tag.id)
```

See the *tags.delete()* documentation.

Add a comment

To add a comment on this dataset:

```
>>> my_comment = sdk.comments.create(text='This is my first dataset',
...                                  project=my_project.id,
...                                  type='dataset',
...                                  target=new_dataset.id)
```

See the *comments.create()* documentation.

We can mark all the comments of this dataset as read with:

```
>>> sdk.comments.mark_as_read(project=my_project.id,
...                           type='dataset',
...                           target=new_dataset.id)
```

See the *comments.mark_as_read()* documentation.

Add an annotation

It is also possible to add an annotation to a project. For example, let's create one whose geometry is the bounding box of the project:

```
>>> a = sdk.annotations.create(project=my_project.id,
...                            geometry=my_project.real_bbox,
...                            name='Project bounding box',
...                            description='Bounding box around the project')
```

See the *annotations.create()* documentation.

This annotation can be deleted with:

```
>>> sdk.annotations.delete(a.id)
```

See the *annotations.delete()* documentation.

1.2 Code Examples

This section describes code examples that demonstrate how to use the Python Delair.ai SDK.

Note: Before running an example, your credentials need to be configured as described in *First steps*.

1.2.1 Create a project with images

Let's create a project on Delair.ai, and upload images on it:

```
from delairstack import DelairStackSDK
import logging

logging.basicConfig(level=logging.DEBUG)

sdk = DelairStackSDK()

# === Create the project ===

my_project = sdk.projects.create(name="My project")

# === Create the survey (mission + flight) with 2 images ===

my_flight, my_mission = sdk.missions.create(
  name='My survey',
  coordinates=[[LONGITUDE1, LATITUDE1],[LONGITUDE2,LATITUDE2], ...],
  project=my_project.id,
  survey_date='2019-01-01T00:00:00.000Z',
  number_of_images=2)

# === Create the dataset for the image 1 and upload it ===

image1_dataset = sdk.datasets.create_image_dataset(
  name='Image 1',
  project=my_project.id,
  mission=my_mission.id,
  flight=my_flight.id,
  geometry={'type': 'Point', 'coordinates': [IMG1_LONGITUDE, IMG1_LATITUDE]},
  width=IMAGE_WIDTH_IN_PIXELS,
  height=IMAGE_HEIGHT_IN_PIXELS,
  )

image1_path = '/path/to/image1.jpg'
sdk.datasets.upload_file(
  dataset=image1_dataset.id,
  component='image',
  file_path=image1_path)

# === Create the dataset for the image 2 and upload it ===

image2_dataset = sdk.datasets.create_image_dataset(
  name='Image 2',
  project=my_project.id,
  mission=my_mission.id,
  flight=my_flight.id,
  geometry={'type': 'Point', 'coordinates': [IMG2_LONGITUDE, IMG2_LATITUDE]},
  width=IMAGE_WIDTH_IN_PIXELS,
  height=IMAGE_HEIGHT_IN_PIXELS,
  )

image2_path = '/path/to/image2.jpg'
sdk.datasets.upload_file(
  dataset=image2_dataset.id,
  component='image',
```

(continues on next page)

(continued from previous page)

```
  file_path=image2_path)

# === Complete the survey upload ===

sdk.missions.complete_survey_upload(flight=my_flight.id)
```

The project, along with the mission and its images are now available on Delair.ai

1.2.2 Create a project with a vector file

Let's create a project on Delair.ai, based on a vector file, such as a geojson file.

```
from delairstack import DelairStackSDK
import logging

logging.basicConfig(level=logging.DEBUG)

sdk = DelairStackSDK()

# === Create the project ===

my_project = sdk.projects.create(name="My project")

# === Create the vector dataset and upload it ===

vector_dataset = sdk.datasets.create_vector_dataset(
  name='My Vector',
  project=my_project.id,
  horizontal_srs_wkt="GEOGCS[\"WGS 84\",DATUM[\"WGS_1984\",SPHEROID[\"WGS 84\",
↪6378137,298.257223563,AUTHORITY[\"EPSG\",\"7030\"]],TOWGS84[0,0,0,0,0,0,0],
↪AUTHORITY[\"EPSG\",\"6326\"]],PRIMEM[\"Greenwich\",0,AUTHORITY[\"EPSG\",\"8901\"]],
↪UNIT[\"degree\",0.0174532925199433,AUTHORITY[\"EPSG\",\"9122\"]],AUTHORITY[\"EPSG\",
↪\"4326\"]]", dataset_format='geojson')

vector_file_to_upload = "/path/to/file.geojson"
sdk.datasets.upload_file(
  dataset=vector_dataset.id,
  component='vector',
  file_path=vector_file_to_upload)
```

The project is now available on Delair.ai

1.2.3 Projects

Create a project

```
>>> my_project = sdk.projects.create(name="My project")
```

See the *projects.create()* documentation.

Delete a project

To delete a project:

```
>>> sdk.projects.delete('5d1a14af0422ae12d537af02')
```

See the *projects.delete()* documentation.

1.2.4 Datasets

Create an image dataset

To create an `image` dataset:

```
>>> my_dataset = sdk.datasets.create_image_dataset(name='My Image',
...                                                project=PROJECT_ID,
...                                                mission=MISSION_ID,
...                                                geometry={'type': 'Point',
↪'coordinates': [LONGITUDE, LATITUDE]},
...                                                width=IMAGE_WIDTH_IN_PIXELS,
...                                                height=IMAGE_HEIGHT_IN_PIXELS)
```

See the *datasets.create_image_dataset()* documentation.

And upload the image file:

```
>>> file_to_upload = "/path/to/image/file.jpg"
>>> sdk.datasets.upload_file(dataset=my_dataset.id,
...                          component='image',
...                          file_path=file_to_upload)
```

See the *datasets.upload_file()* documentation.documentation.

Create a point cloud dataset

To create a point cloud (`pcl`) dataset:

```
>>> my_dataset = sdk.datasets.create_pcl_dataset(name='My PCL',
...                                              project=PROJECT_ID,
...                                              mission=MISSION_ID)
```

See the *datasets.create_pcl_dataset()* documentation.

And upload the pcl file:

```
>>> file_to_upload = "/path/to/pcl/file.las"
>>> sdk.datasets.upload_file(dataset=my_dataset.id,
...                          component='pcl',
...                          file_path=file_to_upload)
```

See the *datasets.upload_file()* documentation.

Create a vector dataset

To create a `vector` dataset with a specific CRS (here `UTM 32631`):

```
>>> my_dataset = sdk.datasets.create_vector_dataset(
...   name='My Vector',
...   project=PROJECT_ID,
...   mission=MISSION_ID,
...   dataset_format='shapefile',
...   is_archive=True,
...   horizontal_srs_wkt='PROJCS["WGS 84 / UTM zone 31N",GEOGCS["WGS 84",[...]')
...   # Full WKT available on http://epsg.io/32631.wkt
```

See the *datasets.create_vector_dataset()* documentation.

And upload the vector file (here a **shapefile**):

```
>>> file_to_upload = "/path/to/vector/shapefile.zip"
>>> sdk.datasets.upload_file(dataset=my_dataset.id,
...                          component='archive',
...                          file_path=file_to_upload)
```

See the *datasets.upload_file()* documentation.documentation.

Create a mesh dataset

To create a `mesh` dataset **with two texture files** (and one material file, which is the default value):

```
>>> my_dataset = sdk.datasets.create_mesh_dataset(name='My Mesh',
...                                               project=PROJECT_ID,
...                                               mission=MISSION_ID,
...                                               texture_count=2)
```

See the *datasets.create_mesh_dataset()* documentation.

And upload the mesh files:

```
>>> mesh_file = "/path/to/mesh/file.obj"
>>> sdk.datasets.upload_file(dataset=my_dataset.id,
...                          component='mesh',
...                          file_path=mesh_file)

>>> first_texture = "/path/to/mesh/texture_a.jpg"
>>> sdk.datasets.upload_file(dataset=my_dataset.id,
...                          component='texture_0',
...                          file_path=first_texture)

>>> second_texture = "/path/to/mesh/texture_b.jpg"
>>> sdk.datasets.upload_file(dataset=my_dataset.id,
...                          component='texture_1',
...                          file_path=second_texture)

>>> material_file = "/path/to/mesh/material.mtl"
>>> sdk.datasets.upload_file(dataset=my_dataset.id,
...                          component='material',
...                          file_path=material_file)
```

See the *datasets.upload_file()* documentation.documentation.

Create a raster dataset

To create a `raster` dataset **with a world file and a projection file**:

```
>>> my_dataset = sdk.datasets.create_raster_dataset(name='My Raster',
...                                                 project=PROJECT_ID,
...                                                 mission=MISSION_ID,
...                                                 has_projection_file=True,
...                                                 has_worldfile=True)
```

See the *datasets.create_raster_dataset()* documentation.

And upload the raster files:

```
>>> raster_file = "/path/to/raster/file.tif"
>>> sdk.datasets.upload_file(dataset=my_dataset.id,
...                          component='raster',
...                          file_path=raster_file)

>>> world_file = "/path/to/raster/worldfile.tfw"
>>> sdk.datasets.upload_file(dataset=my_dataset.id,
...                          component='worldfile',
...                          file_path=world_file)

>>> projection_file = "/path/to/raster/projection.prj"
>>> sdk.datasets.upload_file(dataset=my_dataset.id,
...                          component='projection',
...                          file_path=projection_file)
```

See the *datasets.upload_file()* documentation.documentation.

Create a file dataset

To create a `file` dataset with several files:

```
>>> my_dataset = sdk.datasets.create_file_dataset(name='My File',
...                                               project=PROJECT_ID,
...                                               mission=MISSION_ID,
...                                               file_count=2)
```

See the *datasets.create_file_dataset()* documentation.

And upload the dataset files:

```
>>> first_file = "/path/to/pcl/file.csv"
>>> sdk.datasets.upload_file(dataset=my_dataset.id,
...                          component='file_0',
...                          file_path=first_file)

>>> second_file = "/path/to/pcl/file.pdf"
>>> sdk.datasets.upload_file(dataset=my_dataset.id,
...                          component='file_1',
...                          file_path=second_file)
```

See the *datasets.upload_file()* documentation.

Describe a dataset

```
>>> my_dataset = sdk.datasets.describe(DATASET_ID)
```

See the *datasets.describe()* documentation.

Describe a list of datasets

```
>>> my_dataset_list = sdk.datasets.describe([DATASET_ID, ANOTHER_DATASET_ID])
```

See the *datasets.describe()* documentation.

Download a preview

To download a preview of for a `raster` or `image` dataset:

```
>>> sdk.datasets.download_preview(dataset=DATASET_ID)
```

See the *datasets.download_preview()* documentation.

Delete a dataset

```
>>> sdk.datasets.delete(DATASET_ID)
```

See the *datasets.delete()* documentation.

CHAPTER

TWO

REFERENCE

2.1 Reference

2.1.1 Configuration

The Delair-Stack Python SDK has a unique entry point through the *delairstack.sdk.DelairStackSDK* class. When instantiating that class, one must provide credentials through:

- The keyword arguments *delairstack.sdk.DelairStackSDK.__init__()* function.
- A configuration file.

Keyword Arguments

Here are the configurable properties:

- `user` - The user identifier. Required if not provided through a configuration file, nor using `client_id`
- `password` - The account password. Required if not provided through a configuration file and `user` is used.
- `client_id` - An API client identifier. Required if not provided through a configuration file, nor using `user`.
- `secret` - The API client secret. Required if not provided through a configuration file and `client_id` is used.
- `access_token` - An optional API access token to use to authenticate requests as an alternative to using `user` or `client_id`.
- `url` - The public endpoint of Delair-Stack. Default to https://www.delair.ai.
- `connection` - A dictionnary providing the connection configuration.
- `proxy_url` - An optional proxy URL (will be overrided by the value of `https_proxy` or `http_proxy` environment variable if set).

The connection configuration can specify the default number of retries through the key `max_retries` (the default is to retry each request 10 times with a backoff factor) and whether to disable check of SSL certificates through the key `disable_ssl_certificate` (the default is to disable such checks).

Configuration File

Configuration can be read from a file. Configuration files must be written using JSON format as a single JSON object. The supported properties are the same as the one described in the *Keyword Arguments* section.

The path of the default configuration file depends on the operating system and is documented with the *delairstack.core.config.ConnectionConfig* class.

Example Configuration File

Here is a simple configuration file:

```
{"user": "firstname.lastname@example.com",
 "password": "20h!nph-14-12394"}
```

And a more complete one, specifying both a custom URL and connection configurations:

```
{
  "user": "firstname.lastname@example.com",
  "password": "20h!nph-14-12394",
  "url": "https://www.delair.ai",
  "connection": {
    "max_retries": 3,
    "disable_ssl_certificate": true
  },
  "proxy_url": "https://my-proxy.com:8888"
}
```

Custom Configuration File

In case one has multiple accounts, it's possible to specify a custom configuration file. For example, to instantiate a *delairstack.sdk.DelairStackSDK* using the configuration found in the `~/.local/share/python-delairstack/devconf.json` file:

```
from delairstack import DelairStackSDK
sdk = DelairStackSDK(config_path='~/.local/share/python-delairstack/devconf.json')
```

2.1.2 API Reference

This section covers interfaces of Delair.ai Python SDK.

Main interface

Entry point

Delair-Stack Python SDK has a unique entry point: The class `DelairStackSDK`.

class DelairStackSDK(*, *config_path=None*, *user=None*, *password=None*, *client_id=None*, *secret=None*, *url=None*, *domain=None*, *proxy_url=None*, ***kwargs*)

Entry point providing access to resource managers.

Resource managers are availables as instance attributes. The `dir` builtin can be used to list the availables managers.

The following examples show various ways to instantiate that class:

- Using a username and a password:

```
>>> sdk = DelairStackSDK(user='admin1', password='password')
```

- Using an API client identifier and secret:

```
>>> sdk = DelairStackSDK(client_id='72a5f676-6efc-48c5-ac07-4c534c3cdccc',
                         secret='52ccd77d-17e4-499b-995e-3a2731550723')
```

- Using a configuration file:

```
>>> sdk = DelairStackSDK(config_path='/etc/python-delairstack/conf.json')
```

__init__(*, *config_path=None*, *user=None*, *password=None*, *client_id=None*, *secret=None*, *url=None*, *domain=None*, *proxy_url=None*, ***kwargs*)

Initializes Delair.ai Python SDK entry point.

Parameters

- **config_path** (Optional[str]) – Optional path to a custom configuration file.
- **user** (Optional[str]) – Optional username (email).
- **password** (Optional[str]) – Optional password (mandatory if username is defined).
- **client_id** (Optional[str]) – Optional client identifier.
- **secret** (Optional[str]) – Optional client secret (mandatory if client_id is defined).
- **url** (Optional[str]) – Optional platform URL (default https://www.delair.ai).
- **domain** (Optional[str]) – Optional platform domain.
- **proxy_url** (Optional[str]) – Optional proxy URL.
- **kwargs** – Optional keyword arguments to merge with the configuration.

Configuration

class Config(*defaults=None*, *custom=None*, ***kwargs*)

Base class handling configuration.

It merges multiple sources of configuration and makes all configured properties available as instance attributes.

Three sources of configuration are handled:

- The optional arguments given as *kwargs*.
- A custom configuration file.
- A default configuration.

__init__(*defaults=None*, *custom=None*, ***kwargs*)

Initialize self. See help(type(self)) for accurate signature.

class ConnectionConfig(*file_path=None*, ***, *user=None*, *password=None*, *client_id=None*, *secret=None*, *url=None*, *domain=None*, *proxy_url=None*, ***kwargs*)

Connection configuration.

__init__ (*file_path=None*, ***, *user=None*, *password=None*, *client_id=None*, *secret=None*, *url=None*, *domain=None*, *proxy_url=None*, ***kwargs*)

Initializes a connection configuration.

Parameters

- **file_path** (Optional[str]) – Optional path to a custom configuration file.
- **user** (Optional[str]) – Optional username (email).
- **password** (Optional[str]) – Optional password (mandatory if username is defined).
- **client_id** (Optional[str]) – Optional client identifier.
- **secret** (Optional[str]) – Optional client secret (mandatory if client_id is defined).
- **url** (Optional[str]) – Optional platform URL (default https://www.delair.ai).
- **domain** (Optional[str]) – Optional domain.
- **proxy_url** (Optional[str]) – Optional proxy URL.
- **kwargs** –

  Optional keyword arguments to merge with the configuration.

  kwargs : Optional arguments.

Three sources of configuration are merged:

- The optional arguments *kwargs*.
- The file at path *file_path*.
- The default configuration.

The configuration file is expected to be written in JSON.

Resources management

class ResourcesManagerBase (***, *provider*, ***kwargs*)

Base class implementing resources management.

It provides default implementations of the following operations:

- Resource creation
- Search of resources
- Retrieval of a resource by its identifier
- Update of a resource
- Deletion of a resource

create (***kwargs*)

Creates a resource.

Parameters ****kwargs** – Optional keyword arguments used as the description of the resource to create.

Returns The created resource.

Return type *Resource*

delete(*, *resource*)
Delete the given resource.

Parameters **resource** (Resource) – The resource to delete.

Returns It always returns True.

Return type bool

get(*, *id*)
Get the resource with given identifier.

Parameters **id** (*str*) – The resource identifire.

Returns The resource with identifier equal to id.

Return type *Resource*

search(*, *query*)
Search for resources matching the given query.

Parameters **query** – Query that resources must match.

Returns List of resources matching the search criteria.

Return type [*Resource*]

The argument query is expected to be JSON serializable.

update(*, *resource*)
Update the resource.

Parameters **resource** (Resource) – The resource to update.

Returns The updated resource.

Return type *Resource*

class Resource(*id*, *, *desc*, *manager=None*)

__init__(*id*, *, *desc*, *manager=None*)
Resource class.

Parameters

- **id** (str) – Resource identifier.
- **desc** (dict) – Resource description.
- **manager** (Optional[object]) – Resource manager.

Returns Resource created.

Return type *Resource*

Errors

Package errors.

exception BoundingBoxError(*msg=''*)

exception ConfigError(*msg=''*)

exception DownloadError(*msg=''*)

exception FileError(*msg=''*)

exception ImmutableAttribute(*name*)

exception MissingCredentialsError(*msg=''*)

exception ParameterError(*msg=''*)

exception QueryError(*msg=''*)

exception ResponseError(*msg=''*)

exception SearchError(*msg=''*)

exception TokenRenewalError(*msg=''*)

exception UnsupportedOperationError

exception UnsupportedResourceError(*resource_name*)

exception UploadError(*msg=''*)

Annotations

class AnnotationsImpl(*annotations_api*, *sdk*, ***kwargs*)

class Icons(*value*)
An enumeration.

add_attachments(*annotation*, ***, *attachments=None*, *file_paths=None*, ***kwargs*)
Attach datasets to the annotation.

An attachment is a reference to a dataset handled by the Data Management API.

Items of the `file_paths` argument are interpreted as file paths on the host file system. For each item, a dataset is created and the file at the given path is uploaded. The dataset will be attached to the created annotation.

The created dataset has the following properties:

- It's type is equal to `file` or `image` depending on the local file MIME type.
- It belongs to the same project as the annotation.
- It's named is equal to the basename of the local file.

For fine control of the dataset type, mission, published status, etc. or when the dataset has multiple components, one must create the dataset separately and use the `attachment` argument.

Parameters

- **annotation** (`NewType()(ResourceId, str)`) – Identifier of the annotation to attach to.

- **attachments** (Optional[List[NewType()(ResourceId, str)]]) – Identifiers of dataset to attach to the annotation.
- **file_paths** (Optional[List[AnyStr]]) – List of file path to upload and attach to the annotation.
- ****kwargs** – Optional keyword arguments. Those arguments are passed as is to the API provider.

create(*, *project*, *mission=None*, *geometry=None*, *stroke=None*, *stroke_dasharray=None*, *icon=None*, *stroke_width=None*, *stroke_opacity=None*, *fill=None*, *fill_opacity=None*, *type='2d'*, *target=None*, *name=None*, *description=None*, *followers=None*, *attachments=None*, *file_paths=None*, *normals=None*, *feature=None*, *flight=None*, *image=None*, *dataset=None*, ***kwargs*)

Create an annotation.

Items of the `file_paths` argument are interpreted as file paths on the host file system. For each item, a dataset is created and the file at the given path is uploaded. The dataset will be attached to the created annotation.

Refer to `add_attachments()` for details on the properties of the created datasets.

Parameters

- **project** (NewType()(ResourceId, str)) – Identifier of project to annotate.
- **mission** (Optional[NewType()(ResourceId, str)]) – Identifier of mission to annotate.
- **type** (AnyStr) – Annotation type (must be one of `2d`, `3d` and `image`).
- **geometry** (Optional[dict]) – Geojson geometry of the annotation.
- **stroke** (Optional[List[int]]) – Color used as annotation stroke list of integer `[r, g,b]` or `[r,g,b,a]`.
- **stroke_dasharray** (Optional[List[int]]) – List of integer for dasharray display (specify intervals of line and break).
- **icon** (Union[*Icons*, str, None]) – Icon string or enum. Used for `point` annotations. Enum can be retrieved through `sdk.annotations.Icons` (default: `sdk.annotations.Icons.ANNOTATE`).
- **stroke_width** (Optional[float]) – Width of stroke.
- **stroke_opacity** (Optional[float]) – Opacity of stroke between 0 and 1.
- **fill** (Optional[List[int]]) – Color used as fill for annotation list of integer [r,g,b] or [r,g,b,a]
- **fill_opacity** (Optional[float]) – Opacity of fill between 0 and 1.
- **target** (Optional[NewType()(ResourceId, str)]) – Identifier of the dataset to annotate. Using values such as `2d`, `3d` or `photo` is deprecated.
- **name** (Optional[AnyStr]) – Annotation name.
- **description** (Optional[AnyStr]) – Annotation description.
- **followers** (Optional[List[NewType()(ResourceId, str)]]) – Identifiers of users following the annotation.
- **attachments** (Optional[List[NewType()(ResourceId, str)]]) – Identifiers of datasets to attach to the annotation.

- **file_paths** (Optional[List[AnyStr]]) – List of file paths to upload and attach to the annotation.
- **normals** (Optional[List]) – Transformation vector used to transform the geometry on the front-end (for 3D datasets).
- **feature** (Optional[dict]) – *Deprecated* Converted to geometry property if set.
- **image** (Optional[NewType()(ResourceId, str)]) – *Deprecated* Used as target.id if set.
- **flight** (Optional[NewType()(ResourceId, str)]) – *Deprecated* Not used.
- **dataset** (Optional[NewType()(ResourceId, str)]) – *Deprecated* Used as target.id if set.
- ****kwargs** – Optional keyword arguments. Those arguments are passed as is to the API provider.

Returns The created annotation.

Return type *Resource*

Examples

```
>>> sdk.annotations.create(
...     project='5d63cf9743d61400078efaf8',
...     geometry={
...         "type": "Point",
...         "coordinates": [1440.8495575221236, 1144.8259587020648]
...     },
...     name='My point annotation',
...     type='image',
...     target='5d63cf972fb3880011e57e34',
...     icon=sdk.annotations.Icons.CONVEYOR,
...     followers=['5d5fa52bc207040006390244'],
...     attachments=['5d63cf972fb3880011e57e32']
... )
<delairstack.core.resources.resource.Resource ... (annotations)>
```

create_annotations(*annotations*, ***kwargs*)

Create several annotations.

Parameters

- **annotations** (List[dict]) – List of annotation descriptions, each description is a dictionary with keys among arguments of create().
- ****kwargs** – Optional keyword arguments. Those arguments are passed as is to the API provider.

Return type List[*Resource*]

Returns Descriptions of the created annotations.

delete(*annotation*, ***kwargs*)

Delete an annotation or multiple annotations.

Parameters

- **annotation** (NewType()(SomeResourceIds, Union[NewType()(ResourceId, str), List[NewType()(ResourceId, str)]])) – Identifier of the annotation to delete, or list of such identifiers.
- ****kwargs** – Optional keyword arguments. Those arguments are passed as is to the API provider.

describe(*annotation*, ***kwargs*)

Describe a dataset or a list of datasets.

Parameters

- **annotation** (NewType()(SomeResourceIds, Union[NewType()(ResourceId, str), List[NewType()(ResourceId, str)]])) – Identifier of the annotation to describe, or list of such identifiers.
- ****kwargs** – Optional keyword arguments. Those arguments are passed as is to the API provider.

Return type NewType()(SomeResources, Union[*Resource*, List[*Resource*]])

Returns The annotation description or a list of annotation description.

remove_attachments(*annotation*, *, *attachments*, ***kwargs*)

Remove attachment to the annotation.

Parameters

- **annotation** (NewType()(ResourceId, str)) – Identifier of the annotation to remove attachments from.
- **attachments** (NewType()(SomeResourceIds, Union[NewType()(ResourceId, str), List[NewType()(ResourceId, str)]])) – Identifier of attachments to remove.
- ****kwargs** – Optional keyword arguments. Those arguments are passed as is to the API provider.

rename(*annotation*, *, *name*, ***kwargs*)

Rename the annotation.

Parameters

- **annotation** (NewType()(ResourceId, str)) – Identifier of the annotation to rename.
- **name** (str) – New name of the annotation.
- ****kwargs** – Optional keyword arguments. Those arguments are passed as is to the API provider.

restore(*annotation*, ***kwargs*)

Restore an annotation or multiple annotations.

Parameters

- **annotation** (NewType()(SomeResourceIds, Union[NewType()(ResourceId, str), List[NewType()(ResourceId, str)]])) – Identifier of the annotation to restore, or list of such identifiers.
- ****kwargs** – Optional keyword arguments. Those arguments are passed as is to the API provider.

search(*, *project=None*, *filter=None*, *limit=None*, *page=None*, *sort=None*, *return_total=False*, ***kwargs*)
Search annotations.

Parameters

- **project** (Optional[NewType()(ResourceId, str)]) – Optional identifier of a project to search annotatons for.
- **filter** (Optional[dict]) – Search filter dictionary (refer to /search-annotations in the Annotation API specification for a detailed description).
- **limit** (Optional[int]) – Maximum number of results.
- **page** (Optional[int]) – Page number (starting at page 0).
- **sort** (Optional[dict]) – Sort the results on the specified attributes (1 is sorting in ascending order, -1 is sorting in descending order).
- **return_total** (bool) – Return the number of results found.
- ****kwargs** – Optional keyword arguments. Those arguments are passed as is to the API provider.

Return type Union[ResourcesWithTotal, List[*Resource*]]

Returns A list of annotation descriptions or a namedtuple with total number of results and list of annotation descriptions.

set_description(*annotation*, *, *description*, ***kwargs*)
Set the annotation description.

Parameters

- **annotation** (NewType()(ResourceId, str)) – Identifier of the annotation whose description to set.
- **description** (str) – Description of the annotation.
- ****kwargs** – Optional keyword arguments. Those arguments are passed as is to the API provider.

set_fill_color(*annotation*, *, *color*, *opacity=None*, ***kwargs*)
Set the color used to fill the annotation.

Parameters

- **annotation** (NewType()(ResourceId, str)) – Identifier of the annotation whose fill color to set.
- **color** (List[float]) – Fill color to set interpreted as an RGB-triple.
- **opacity** (Optional[float]) – Optional opacity of fill color, a float number between 0 and 1.
- ****kwargs** – Optional keyword arguments. Those arguments are passed as is to the API provider.

set_fill_opacity(*annotation*, *, *opacity*, ***kwargs*)
Set the opacity of the annotation fill.

Parameters

- **annotation** (NewType()(ResourceId, str)) – Identifier of the annotation whose fill opacity to set.
- **opacity** (float) – Fill opacity to set.

- ****kwargs** – Optional keyword arguments. Those arguments are passed as is to the API provider.

set_geometry(*annotation*, *, *geometry*, ***kwargs*)
Set the geometry of the annotation.

Parameters

- **annotation** (NewType()(ResourceId, str)) – Identifier of the annotation whose geometry to set.
- **geometry** (dict) – A dictionary following GeoJSON specification.
- ****kwargs** – Optional keyword arguments. Those arguments are passed as is to the API provider.

set_icon(*annotation*, *, *icon=None*, ***kwargs*)
Set the annotation icon.

Parameters

- **annotation** (NewType()(ResourceId, str)) – Identifier of the annotation whose icon to set.
- **icon** (Union[*Icons*, str, None]) – Icon string or enum. Used for point annotations. Enum can be retrieved through sdk.annotations.Icons (default: sdk.annotations.Icons.ANNOTATE).
- ****kwargs** – Optional keyword arguments. Those arguments are passed as is to the API provider.

set_normals(*annotation*, *, *normals*, ***kwargs*)
Set the annotation normal vectors.

Setting the normals of an annotation makes sense for annotations of type 3d only. The argument normals is expected to be a list of 3-dimensional vectors, one for each vertice of the annotation geometry. Those vectors are interpreted as normals to the target of the annotation and are used to shift the annotation geometry when it is drawn so that it doesn't overlap the target of the annotation and is drawn on the right side of the target facets.

Parameters

- **annotation** (NewType()(ResourceId, str)) – Identifier of the annotation whose normal vector to set.
- **normals** (List) – List of coordinates of normal vectors.
- ****kwargs** – Optional keyword arguments. Those arguments are passed as is to the API provider.

set_stroke_color(*annotation*, *, *color*, *opacity=None*, ***kwargs*)
Set the stroke color of the annotation.

Parameters

- **annotation** (NewType()(ResourceId, str)) – Identifier of the annotation whose stroke color to set.
- **color** (Tuple[int, int, int]) – Stroke color to set interpreted as an RGB-triple.
- **opacity** (Optional[float]) – Optional opacity of stroke color, a float number between 0 and 1.
- ****kwargs** – Optional keyword arguments. Those arguments are passed as is to the API provider.

set_stroke_dasharray(*annotation*, *, *dasharray*, ***kwargs*)
Set the dasharray of the annotation stroke.

The dasharray defines the pattern of dashes and gaps used to paint the outline of the annotation.

Parameters

- **annotation** (NewType()(ResourceId, str)) – Identifier of the annotation whose stroke opacity to set.
- **dasharray** (List[float]) – Dasharray to set.
- ****kwargs** – Optional keyword arguments. Those arguments are passed as is to the API provider.

set_stroke_opacity(*annotation*, *, *opacity*, ***kwargs*)
Set the opacity of the annotation stroke.

Parameters

- **annotation** (NewType()(ResourceId, str)) – Identifier of the annotation whose stroke opacity to set.
- **opacity** (float) – Stroke opacity to set.
- ****kwargs** – Optional keyword arguments. Those arguments are passed as is to the API provider.

set_stroke_width(*annotation*, *, *width*, ***kwargs*)
Set the stroke width of the annotation.

Parameters

- **annotation** (NewType()(ResourceId, str)) – Identifier of the annotation whose stroke width to set.
- **width** (float) – Stroke width to set.
- ****kwargs** – Optional keyword arguments. Those arguments are passed as is to the API provider.

update(*, *resource*)
Deprecated.

Return type *Resource*

Comments

class CommentsImpl(*ui_services_api*, *sdk*, ***kwargs*)

create(*text*, *, *project*, *type*, *target=None*, *flight=None*, ***kwargs*)
Create a comment.

Parameters

- **text** (str) – Comment content.
- **project** (NewType()(ResourceId, str)) – Identifier of project to comment.
- **type** (str) – Comment type (must be one of project, annotation, flight, photo, dataset, feature, gcp, task).
- **target** (Optional[NewType()(ResourceId, str)]) – Optional identifier of the target.

- **flight** (Optional[NewType()(ResourceId, str)]) – Optional identifier of the flight (mandatory when the comment type is photo).
- ****kwargs** – Optional keyword arguments. Those arguments are passed as is to the API provider.

Returns The created comment.

Return type *Resource*

Examples

```
>>> sdk.comments.create(
...     text='my comment',
...     project='5d63cf972fb3880011e57f22',
...     type='dataset',
...     target='5d63cf972fb3880011e57e34')
<delairstack.core.resources.comments.Comment ... (comments)>
```

mark_as_read(*, *project*, *type=None*, *target=None*, *flight=None*, ***kwargs*)

Mark all the comments of a target or project as read.

Parameters

- **project** (NewType()(ResourceId, str)) – Identifier of project.
- **type** (Optional[str]) – Optional comment type (must be one of project, annotation, flight, photo, dataset, feature, gcp, task).
- **target** (Optional[NewType()(ResourceId, str)]) – Optional identifier of the target.
- **flight** (Optional[NewType()(ResourceId, str)]) – Optional identifier of the flight (mandatory when the comment type is photo).
- ****kwargs** – Optional keyword arguments. Those arguments are passed as is to the API provider.

Examples

```
>>> sdk.comments.mark_as_read(
...     project='5d63cf972fb3880011e57f22',
...     type='dataset',
...     target='5d63cf972fb3880011e57e34')
```

Return type None

search(*, *project*, *type=None*, *target=None*, *flight=None*, ***kwargs*)

Search for comments.

When searching for comments on a photo. Both the flight id and target id must be supplied.

Parameters

- **project** (NewType()(ResourceId, str)) – Identifier of the project.
- **type** (Optional[str]) – Optional comment type (must be one of project, annotation, flight, photo, dataset, feature, gcp, task).

- **target** (Optional[NewType()(ResourceId, str)]) – Optional identifier of the target.
- **flight** (Optional[NewType()(ResourceId, str)]) – Optional identifier of the flight (mandatory when the comment type is photo).
- ****kwargs** – Optional keyword arguments. Those arguments are passed as is to the API provider.

Returns The found comments.

Return type Resources

Examples

```
>>> sdk.comments.search(project='5d63cf972fb3880011e57f22')
[<delairstack.core.resources.comments.Comment ... (comments)>]
```

Datasets

class DatasetsImpl(*data_management_api*, *auth_api*, ***kwargs*)

add_categories(*dataset*, *, *categories*, ***kwargs*)

Add categories to the dataset.

Parameters

- **dataset** (NewType()(ResourceId, str)) – Identifier of the dataset to add categories to.
- **categories** (List[str]) – Categories to add.
- ****kwargs** – Optional keyword arguments. Those arguments are passed as is to the API provider.

create_datasets(*datasets*)

Create several datasets *(bulk dataset creation)*.

Parameters **datasets** (List[dict]) – List of dataset dictionnaries (refer to */create-datasets* definition in the Data Manager API for a detailed description of datasets)

Return type List[*Resource*]

Returns A list of the created dataset descriptions.

Example

```
>>> sdk.datasets.create_datasets(
... datasets=[{'name': 'My file dataset',
...            'type': 'file',
...            'components': [{'name': 'kind_of_file'}]},
...           {'name': 'My image',
...            'type': 'image',
...            'components': [{'name': 'image'}]}])
[<delairstack.core.resources.resource.Resource... (dataset)>, ...]
```

create_file_dataset(*, *name*, *categories=None*, *project=None*, *mission=None*, *hidden=None*, *published=None*, *horizontal_srs_wkt=None*, *vertical_srs_wkt=None*, *dataset_format=None*, *geometry=None*, *properties=None*, *file_count=1*, *components=None*, ***kwargs*)

Create a dataset of type `file`.

Parameters

- **name** (`str`) – Name of the dataset.
- **categories** (`Optional[Sequence[str]]`) – Sequence of categories or None if there's no category to set on the dataset.
- **project** (`Optional[NewType()(ResourceId, str)]`) – Optional project identifier.
- **mission** (`Optional[NewType()(ResourceId, str)]`) – Optional mission identifier.
- **hidden** (`Optional[bool]`) – Whether not to display the dataset to end-users or not.
- **published** (`Optional[bool]`) – Whether the dataset is ready for delivery or not.
- **horizontal_srs_wkt** (`Optional[str]`) – Optional geographic coordinate system for horizontal coordinattes in WKT format.
- **vertical_srs_wkt** (`Optional[str]`) – Optional geographic coordinate system for vertical coordinattes in WKT format.
- **dataset_format** (`Optional[str]`) – Optional file format.
- **geometry** (`Optional[dict]`) – Optional geometry of the dataset.
- **properties** (`Optional[dict]`) – Optional custom properties of the dataset.
- **file_count** (`int`) – Number of files. Default to 1. It is used to generate the component names automatically, except if `components` is defined.
- **components** (`Optional[List[str]]`) – Optional sequence of component names. When defined, `file_count` is ignored.
- ****kwargs** – Optional keyword arguments. Those arguments are passed as is to the API provider.

Returns Resource for the created dataset.

Return type *Resource*

create_image_dataset(*, *name*, *categories=None*, *project=None*, *mission=None*, *flight=None*, *hidden=None*, *published=None*, *horizontal_srs_wkt=None*, *vertical_srs_wkt=None*, *dataset_format=None*, *geometry=None*, *properties=None*, *acquisition_date=None*, *width=None*, *height=None*, *sensor=None*, *lens=None*, *camera_parameters=None*, *reflectance_calibration_panel=None*, ***kwargs*)

Create a dataset of type `image`.

Parameters

- **name** (`str`) – Name of the dataset.
- **categories** (`Optional[Sequence[str]]`) – Sequence of categories or None if there's no category to set on the dataset.
- **project** (`Optional[NewType()(ResourceId, str)]`) – Optional project identifier.

- **mission** (Optional[NewType()(ResourceId, str)]) – Optional mission identifier.
- **flight** (Optional[NewType()(ResourceId, str)]) – Optional flight identifier.
- **hidden** (Optional[bool]) – Whether not to display the dataset to end-users or not.
- **published** (Optional[bool]) – Whether the dataset is ready for delivery or not.
- **horizontal_srs_wkt** (Optional[str]) – Optional geographic coordinate system for horizontal coordinattes in WKT format.
- **vertical_srs_wkt** (Optional[str]) – Optional geographic coordinate system for vertical coordinattes in WKT format.
- **dataset_format** (Optional[str]) – Optional file format.
- **geometry** (Optional[dict]) – Optional geometry of the dataset.
- **properties** (Optional[dict]) – Optional custom properties of the dataset.
- **acquisition_date** (Optional[str]) – Optional acquisition date (format: YYYY-MM-DDTHH:MM:SS.sssZ).
- **width** (Optional[int]) – Optional image width.
- **height** (Optional[int]) – Optional image height.
- **sensor** (Optional[dict]) – Optional sensor properties.
- **lens** (Optional[dict]) – Optional lens properties.
- **camera_parameters** (Optional[dict]) – Optional camera parameters description.
- **reflectance_calibration_panel** (Optional[dict]) – Optional reflectance calibration panel description.
- ****kwargs** – Optional keyword arguments. Those arguments are passed as is to the API provider.

Returns Resource for the created dataset.

Return type *Resource*

create_mesh_dataset(*, *name*, *categories=None*, *project=None*, *mission=None*, *hidden=None*, *published=None*, *horizontal_srs_wkt=None*, *vertical_srs_wkt=None*, *dataset_format=None*, *geometry=None*, *properties=None*, *texture_count=1*, *material_count=1*, *offset=None*, ***kwargs*)

Create a dataset of type mesh.

Parameters

- **name** (str) – Name of the dataset.
- **categories** (Optional[Sequence[str]]) – Sequence of categories or None if there's no category to set on the dataset.
- **project** (Optional[NewType()(ResourceId, str)]) – Optional project identifier.
- **mission** (Optional[NewType()(ResourceId, str)]) – Optional mission identifier.
- **hidden** (Optional[bool]) – Whether not to display the dataset to end-users or not.
- **published** (Optional[bool]) – Whether the dataset is ready for delivery or not.

- **horizontal_srs_wkt** (Optional[str]) – Optional geographic coordinate system for horizontal coordinattes in WKT format.
- **vertical_srs_wkt** (Optional[str]) – Optional geographic coordinate system for vertical coordinattes in WKT format.
- **dataset_format** (Optional[str]) – Optional file format.
- **geometry** (Optional[dict]) – Optional geometry of the dataset.
- **properties** (Optional[dict]) – Optional custom properties of the dataset.
- **texture_count** – Number of texture files. Default to 1.
- **material_count** – Number of materials files. Defaut to 1.
- **offset** (Optional[NewType()(Offset, Tuple[float, float, float])]) – Optional translation from mesh coordinates to spatial coordinates.
- ****kwargs** – Optional keyword arguments. Those arguments are passed as is to the API provider.

Returns Resource for the created dataset.

Return type *Resource*

create_pcl_dataset(*, *name*, *categories=None*, *project=None*, *mission=None*, *hidden=None*, *published=None*, *horizontal_srs_wkt=None*, *vertical_srs_wkt=None*, *dataset_format=None*, *geometry=None*, *properties=None*, ***kwargs*)

Create a dataset of type pcl.

Parameters

- **name** (str) – Name of the dataset.
- **categories** (Optional[Sequence[str]]) – Sequence of categories or None if there's no category to set on the dataset.
- **project** (Optional[NewType()(ResourceId, str)]) – Optional project identifier.
- **mission** (Optional[NewType()(ResourceId, str)]) – Optional mission identifier.
- **hidden** (Optional[bool]) – Whether not to display the dataset to end-users or not.
- **published** (Optional[bool]) – Whether the dataset is ready for delivery or not.
- **horizontal_srs_wkt** (Optional[str]) – Optional geographic coordinate system for horizontal coordinattes in WKT format.
- **vertical_srs_wkt** (Optional[str]) – Optional geographic coordinate system for vertical coordinattes in WKT format.
- **dataset_format** (Optional[str]) – Optional file format.
- **geometry** (Optional[dict]) – Optional geometry of the dataset.
- **properties** (Optional[dict]) – Optional custom properties of the dataset.
- ****kwargs** – Optional keyword arguments. Those arguments are passed as is to the API provider.

Returns Resource for the created dataset.

Return type *Resource*

create_raster_dataset(*, *name*, *categories=None*, *project=None*, *mission=None*, *hidden=None*, *published=None*, *horizontal_srs_wkt=None*, *vertical_srs_wkt=None*, *dataset_format=None*, *geometry=None*, *properties=None*, *bands=None*, *has_projection_file=False*, *has_worldfile=False*, *has_headerfile=False*, ***kwargs*)

Create a dataset of type `raster`.

Parameters

- **name** (`str`) – Name of the dataset.
- **categories** (`Optional[Sequence[str]]`) – Sequence of categories or None if there's no category to set on the dataset.
- **project** (`Optional[NewType()(ResourceId, str)]`) – Optional project identifier.
- **mission** (`Optional[NewType()(ResourceId, str)]`) – Optional mission identifier.
- **hidden** (`Optional[bool]`) – Whether not to display the dataset to end-users or not.
- **published** (`Optional[bool]`) – Whether the dataset is ready for delivery or not.
- **horizontal_srs_wkt** (`Optional[str]`) – Optional geographic coordinate system for horizontal coordinattes in WKT format.
- **vertical_srs_wkt** (`Optional[str]`) – Optional geographic coordinate system for vertical coordinattes in WKT format.
- **dataset_format** (`Optional[str]`) – Optional file format.
- **geometry** (`Optional[dict]`) – Optional geometry of the dataset.
- **properties** (`Optional[dict]`) – Optional custom properties of the dataset.
- **bands** (`Optional[List[dict]]`) – Option list of band properties.
- **has_projection_file** (`bool`) – Whether there is a sidecar file to define the raster projection.
- **has_worldfile** (`bool`) – Whether there is a sidecar file to georeference the raster.
- **has_headerfile** (`bool`) – Whether there is a sidecar file for envi format raster.
- ****kwargs** – Optional keyword arguments. Those arguments are passed as is to the API provider.

Returns Resource for the created dataset.

Return type *Resource*

create_vector_dataset(*, *name*, *categories=None*, *project=None*, *mission=None*, *hidden=None*, *published=None*, *collection=None*, *origin=None*, *horizontal_srs_wkt=None*, *vertical_srs_wkt=None*, *dataset_format=None*, *geometry=None*, *properties=None*, *is_shape_file=False*, *is_archive=False*, *has_projection_file=False*, ***kwargs*)

Create a dataset of type `vector`.

When `is_archive` is True, `is_shape_file` and `has_projection_file` must be False.

Parameters

- **name** (`str`) – Name of the dataset.

- **categories** (Optional[Sequence[str]]) – Sequence of categories or None if there's no category to set on the dataset.
- **project** (Optional[NewType()(ResourceId, str)]) – Optional project identifier.
- **mission** (Optional[NewType()(ResourceId, str)]) – Optional mission identifier.
- **hidden** (Optional[bool]) – Whether not to display the dataset to end-users or not.
- **published** (Optional[bool]) – Whether the dataset is ready for delivery or not.
- **collection** (Optional[NewType()(ResourceId, str)]) – Optional map-service collection to use as data source. Providing a collection isn't compatible with setting is_shape_file, has_projection_file, is_archive to True, nor setting dataset_format.
- **origin** (Optional[NewType()(ResourceId, str)]) – Optional origin vector dataset (source: data-manager) for a vector collection dataset (source: map-service).
- **horizontal_srs_wkt** (Optional[str]) – Optional geographic coordinate system for horizontal coordinattes in WKT format.
- **vertical_srs_wkt** (Optional[str]) – Optional geographic coordinate system for vertical coordinattes in WKT format.
- **dataset_format** (Optional[str]) – Optional file format.
- **geometry** (Optional[dict]) – Optional geometry of the dataset.
- **properties** (Optional[dict]) – Optional custom properties of the dataset.
- **is_shape_file** (bool) – Whether it is an ESRI Shapefile.
- **is_archive** (bool) – Whether it is an archive.
- **has_projection_file** (bool) – Whether there is a sidecar file to define the shapes projection.
- ****kwargs** – Optional keyword arguments. Those arguments are passed as is to the API provider.

Returns Resource for the created dataset.

Return type *Resource*

delete(*dataset*, ***kwargs*)

Delete a dataset or multiple datasets.

Parameters

- **dataset** (NewType()(SomeResourceIds, Union[NewType()(ResourceId, str), List[NewType()(ResourceId, str)]])) – Identifier of the dataset to delete, or list of such identifiers.
- ****kwargs** – Optional keyword arguments. Those arguments are passed as is to the API provider.

delete_properties(*dataset*, ***, *properties*, ***kwargs*)

Delete properties of the dataset.

Parameters

- **dataset** (NewType()(ResourceId, str)) – Identifier of the dataset whose properties to delete.

- **properties** (List[str]) – Names of properties to delete.
- ****kwargs** – Optional keyword arguments. Those arguments are passed as is to the API provider.

describe(*dataset*, ***kwargs*)

Describe a dataset or a list of datasets.

Parameters

- **dataset** (NewType()(SomeResourceIds, Union[NewType()(ResourceId, str), List[NewType()(ResourceId, str)]])) – Identifier of the dataset to describe, or list of such identifiers.
- ****kwargs** – Optional keyword arguments. Those arguments are passed as is to the API provider.

Return type NewType()(SomeResources, Union[*Resource*, List[*Resource*]])

Returns The dataset description or a list of dataset description.

download_component(*dataset*, *, *component*, *target_path=None*, *target_name=None*, *overwrite=False*, *md5hash=None*)

Download the file from a component.

If the path target_path doesn't exists, it is created.

Parameters

- **dataset** (NewType()(ResourceId, str)) – Identifier of the dataset to download from.
- **component** (str) – Name of component to download from.
- **target_path** (Optional[str]) – Path of directory where to save the downloaded file. Default to current directory.
- **target_name** (Optional[str]) – Name of downloaded file. Default to the file name suggested by the server.
- **overwrite** – Whether to overwrite an existing file. Default to False.
- **md5hash** (Optional[str]) – Optional MD5 hash of the file to download read in binary mode and containing only hexadecimal digits. When not equal to None (the default), will be compared to the equivalent hash for the downloaded file.

Raises ***DownloadError*** – When the MD5 hash of the downloaded file doesn't match md5hash.

Return type str

Returns Path of the downloaded file.

download_image_as_jpeg(*dataset*, *target_path=None*, *target_name=None*, *overwrite=False*, *md5hash=None*)

Download an image as JPEG.

If the path target_path doesn't exists, it is created.

Parameters

- **dataset** (NewType()(ResourceId, str)) – Identifier of the dataset to download from.
- **target_path** (Optional[str]) – Path of directory where to save the downloaded file. Default to current directory.

- **target_name** (Optional[str]) – Name of downloaded file. Default to the file name suggested by the server.
- **overwrite** – Whether to overwrite an existing file. Default to False.
- **md5hash** (Optional[str]) – Optional MD5 hash of the file to download read in binary mode and containing only hexadecimal digits. When not equal to None (the default), will be compared to the equivalent hash for the downloaded file.

Raises ***DownloadError*** – When the MD5 hash of the downloaded file doesn't match md5hash.

Return type str

Returns Path of the downloaded file.

download_preview(*dataset*, *target_path=None*, *target_name=None*, *overwrite=False*)

Download a dataset preview.

If the path target_path doesn't exists, it is created.

Parameters

- **dataset** (NewType()(ResourceId, str)) – Identifier of the dataset to download from.
- **target_path** (Optional[str]) – Path of directory where to save the downloaded file. Default to current directory.
- **target_name** (Optional[str]) – Name of downloaded file. Default to the file name suggested by the server.
- **overwrite** – Whether to overwrite an existing file. Default to False.

Return type str

Returns Path of the downloaded file.

remove_categories(*dataset*, ***, *categories*, ***kwargs*)

Remove categories from the dataset.

Parameters

- **dataset** (NewType()(ResourceId, str)) – Identifier of the dataset to remove categories from.
- **categories** (List[str]) – Categories to remove.
- ****kwargs** – Optional keyword arguments. Those arguments are passed as is to the API provider.

rename(*dataset*, ***, *name*, ***kwargs*)

Rename the dataset.

Parameters

- **dataset** (NewType()(ResourceId, str)) – Identifier of the dataset to rename.
- **name** (str) – New name of the dataset.
- ****kwargs** – Optional keyword arguments. Those arguments are passed as is to the API provider.

restore(*dataset*, ***kwargs*)

Restore a dataset or multiple datasets.

Parameters

- **dataset** (NewType()(SomeResourceIds, Union[NewType()(ResourceId, str), List[NewType()(ResourceId, str)]])) – Identifier of the dataset to restore, or list of such identifiers.
- ****kwargs** – Optional keyword arguments. Those arguments are passed as is to the API provider.

search(*, *filter=None*, *limit=None*, *page=None*, *sort=None*, *return_total=False*, ***kwargs*)

Search datasets.

Parameters

- **filter** (Optional[dict]) – Search filter dictionary (refer to /search-datasets definition in the Data Manager API for a detailed description of filter).
- **limit** (Optional[int]) – Maximum number of results to extract.
- **page** (Optional[int]) – Page number (starting at page 0).
- **sort** (Optional[dict]) – Sort the results on the specified attributes (1 is sorting in ascending order, -1 is sorting in descending order).
- **return_total** (bool) – Return the number of results found.
- ****kwargs** – Optional keyword arguments. Those arguments are passed as is to the API provider.

Return type Union[ResourcesWithTotal, List[*Resource*]]

Returns A list of dataset descriptions OR a namedtuple with total number of results and list of dataset descriptions.

Examples

```
>>> sdk.datasets.search(filter={'name': {'$eq': 'My image'}})
[<delairstack.core.resources.resource.Resource ... (dataset)>, ...]
```

```
>>> sdk.datasets.search(filter={'name': {'$eq': 'My image'}},
...                     return_total=True)
ResourcesWithTotal(
    total=...,
    results=[<delairstack.core.resources.resource.Resource..., ...]
)
```

search_generator(*, *filter=None*, *limit=50*, ***kwargs*)

Search datasets and return results through a generator.

The generator allows the user not to care about the pagination of results, while being memory-effective. Found datasets are sorted chronologically in order to allow new datasets to be found during the search.

Parameters

- **filter** (Optional[dict]) – Search filter dictionary (refer to /search-datasets definition in the Data Manager API for a detailed description of filter)
- **limit** (int) – Optional maximum number of results by search request.
- ****kwargs** – Optional keyword arguments. Those arguments are passed as is to the API provider.

Return type Generator[*Resource*, None, None]

Returns A generator yielding each found dataset.

Examples

```
>>> datasets = sdk.datasets.search_generator(
...     filter={'name': {'$eq': 'My image'}})
>>> print(datasets)
<generator object DatasetsImpl.search_generator at ...>
>>> for dataset in datasets:
>>>     print(dataset)
<delairstack.core.resources.resource.Resource with id... (dataset)>
```

share_tiles(*dataset*, *, *company=None*, *duration=None*)

Return a URL template to share access to the tiles of a dataset.

Parameters

- **dataset** (NewType()(ResourceId, str)) – Identifier of the dataset to create a URL for.
- **company** (Optional[NewType()(ResourceId, str)]) – Optional identifier of a company to attach the created token too. When equal to None (the default), the user company is used.
- **duration** (Optional[int]) – Optional duration in seconds of the created token. When equal to None (the default) the created token won't expire.

Raises ***UnsupportedResourceError*** – In case the dataset ingestion status isn't equal to complete or its type isn't raster or vector with mapservice source.

Return type str

update_properties(*dataset*, *, *properties*, ***kwargs*)

Update the dataset properties.

Parameters

- **dataset** (NewType()(ResourceId, str)) – Identifier of the dataset whose properties to update.
- **properties** (dict) – Dictionary of properties to update.
- ****kwargs** – Optional keyword arguments. Those arguments are passed as is to the API provider.

upload_file(*dataset*, *, *component*, *file_path*, *md5hash=None*, *multipart=True*, *chunk_size=None*)

Upload a file to a dataset component.

Parameters

- **dataset** (NewType()(ResourceId, str)) – Identifier of the dataset to upload to.
- **component** (str) – Name of component to upload to.
- **file_path** (NewType()(AnyPath, Union[str, Path])) – Path to the file to upload.
- **md5hash** (Optional[str]) – Optional MD5 hash of the file to upload read in binary mode and containing only hexadecimal digits. Will be computed when equal to None (the default).
- **multipart** (bool) – Whether to upload the file using multipart upload. Default to True unless the file size is less than 5MB the file is upload in one request.

- **chunk_size** (Optional[int]) – The size in byte of each part for a multipart upload. If file size is less than this number, multipart will not used. The value should be between 5MB and 50MB. 5MB is default.

Missions

class MissionsImpl(*project_manager_api*, *ui_services_api*, ***kwargs*)

complete_survey_upload(*, *flight*, *status='complete'*)
Complete the survey upload.

It notifies the front-end that all the images have been uploaded. It is necessary after the create_survey. Otherwise, the upload progression bar will still be displayed.

Parameters

- **flight** (NewType()(ResourceId, str)) – Flight identifier.
- **status** (str) – Upload completion status (complete or 'failed).

Raises ***QueryError*** – When the response is not consistent.

create(*, *project*, *survey_date*, *number_of_images*, *name=None*, ***kwargs*)
Creates a mission.

Based on the number of images to attach to the mission, this function calls create_survey() or create_mission().

Parameters

- **project** (NewType()(ResourceId, str)) – Identifier of the project on which the mission is added.
- **survey_date** (str) – Survey date of the mission (format: YYYY-MM-DDTHH:MM:SS.sssZ).
- **number_of_images** (int) – Number of images that will be uploaded.
- **name** (Optional[str]) – Optional mission name.
- ****kwargs** – Optional arguments that will be merged into the mission description.

Returns A tuple with the created flight and mission. Flight = None when the number of images is 0.

Return type (*Flight*, *Mission*)

create_mission(*, *project*, *survey_date*, *name=None*, ***kwargs*)
Creates a mission without images.

This function is used when no image is attached to the mission. As a consequence, no flight will be created.

Parameters

- **project** (NewType()(ResourceId, str)) – Identifier of the project on which the mission is added.
- **survey_date** (str) – Survey date of the mission (format: YYYY-MM-DDTHH:MM:SS.sssZ).
- **name** (Optional[str]) – Optional mission name.
- ****kwargs** – Optional arguments that will be merged into the mission description.

Returns The created mission.

Return type *Mission*

create_survey(*, *survey_date*, *project*, *number_of_images*, *name=None*, *coordinates=None*, *area=0*, ***kwargs*)

Create a survey (mission + flight).

This function is used when images will be attached to the mission. As a consequence, a flight will be created as well.

Parameters

- **survey_date** (str) – Survey date (format: YYYY-MM-DDTHH:MM:SS.sssZ).
- **project** (NewType()(ResourceId, str)) – Project identifier on which the survey is added.
- **number_of_images** (int) – Number of photos that will be uploaded.
- **name** (Optional[str]) – Optional mission name.
- **coordinates** (Optional[List]) – Coordinates bounding the mission to create.
- **area** (float) – Optional survey area.
- ****kwargs** – Optional keyword arguments. Those arguments are passed as is to the API provider.

Raises ***QueryError*** – The survey creation response is incorrect.

Returns A tuple with the created flight and mission.

Return type (*Flight*, *Mission*)

delete(*mission*)

Delete a mission.

Parameters **mission** (NewType()(ResourceId, str)) – Identifier of the mission to delete.

search(*, *missions=None*, *flights=None*, *project=None*, *deleted=False*, *name=None*, ***kwargs*)

Search missions.

Parameters

- **missions** (Optional[List[NewType()(ResourceId, str)]]) – Optional list of mission identifiers.
- **flights** (Optional[List[NewType()(ResourceId, str)]]) – Optional list of flight identifiers.
- **project** (Optional[NewType()(ResourceId, str)]) – Optional project identifier.
- **deleted** (bool) – Optional parameter to search for deleted missions or not (False by default).
- **name** (Optional[str]) – *Deprecated* Optional project, mission or flight name.
- ****kwargs** – Optional keyword arguments. Those arguments are passed as is to the API provider.

Raises ***QueryError*** – When the response is not consistent.

Returns List of missions matching the search criteria.

Return type [*Mission*]

Examples

Get the missions related to a specific project:

```
>>> sdk.missions.search(project=my_project_resource.id)
[<delairstack.core.resources.projectmngt.missions.Mission...>, ...]
```

Mission resource

class Mission(***kwargs*)

__init__(***kwargs*)
Mission resource.

Parameters

- **id** – Mission identifier.
- **name** – Mission name.
- **project** – Project identifier.
- **survey_date** – Survey date (format: `YYYY-MM-DDTHH:MM:SS.sssZ`).
- **flights** – List of flight identifiers.
- **geometry** – Mission geometry.
- **created** – Mission creation date (format: `YYYY-MM-DDTHH:MM:SS.sssZ`).
- **user** – Mission creation user.
- **modification_date** – Mission last modification date (format: `YYYY-MM-DDTHH:MM:SS.sssZ`).
- **modification_user** – Mission last modification user.
- **real_bbox** – Mission bounding box.

Returns A mission resource.

Return type *Mission*

Flight resource

class Flight(***kwargs*)

__init__(***kwargs*)
Flight resource.

Parameters

- **id** – Flight identifier.
- **name** – Flight name.
- **survey_date** – Survey date (format: `YYYY-MM-DDTHH:MM:SS.sssZ`).
- **project** – Project identifier.
- **mission** – Mission identifier.

- **number_of_photos** – Number of images in the flight.
- **created** – Flight creation date.
- **user** – Flight creation user.

Returns A flight resource.

Return type *Flight*

Projects

class ProjectsImpl(*project_manager_api*, *ui_services_api*, ***kwargs*)

create(***kwargs*)
Create a project.

Parameters

- **name** – Project name.
- **geometry** – Optional project geometry.
- ****kwargs** – Optional keyword arguments. Those arguments are passed as is to the API provider.

Raises ***QueryError*** – The project creation response is incorrect.

Returns A resource encapsulating the created project.

Return type *Project*

delete(*project*)
Delete the specified Project.

Parameters **project** (`NewType()(ResourceId, str)`) – Identifier of the project to delete.

Return type `None`

describe(*project*, *deleted=False*)
Describe the project for the specified id.

Parameters

- **project** (`NewType()(ResourceId, str)`) – Project identifier.
- **deleted** (`bool`) – Optional parameter to describe a deleted project or not (`False` by default).

Returns Project resource matching the id (`None` if not found).

Return type *Project*

Examples

```
>>> sdk.projects.describe('5ce7f379327e9d5f15e37bb4')
<delairstack.core.resources.projectmngt.projects.Project ...>
```

search (*, *name*, *deleted=False*, ***kwargs*)

Search for projects.

Parameters

- **name** (str) – Name of the project *(the comparison is case insensitive)*. * wildcard is supported
- **deleted** (bool) – Optional parameter to search for deleted project or not (False by default).
- ****kwargs** – Optional keyword arguments. Those arguments are passed as is to the API provider.

Raises ***QueryError*** – When the response is not consistent.

Returns List of project resources matching the search criteria.

Return type [*Project*]

Examples

Get the projects with a specific name (name is not unique):

```
>>> sdk.projects.search(name='My_project')
[<delairstack.core.resources.projectmngt.projects.Project...>, ...]
```

Get all the projects available:

```
>>> sdk.projects.search(name='*')
[<delairstack.core.resources.projectmngt.projects.Project...>, ...]
```

update_status (*project*, *status*)

Update the project status.

Parameters

- **project** (NewType() (ResourceId, str)) – Project identifier.
- **status** (str) – Project status (pending, available, failed).

Raises

- ***ResponseError*** – When the project has not been found.
- **RuntimeError** – The passed status is not allowed.

Returns Updated project resource.

Return type *Project*

Project resource

class Project(***kwargs*)

__init__(***kwargs*)
Project resource.

Parameters

- **id** – Project identifier.
- **name** – Project name.
- **geometry** – Project geometry.
- **industry** – Project industry.
- **created** – Project creation date (format: `YYYY-MM-DDTHH:MM:SS.sssZ`).
- **company** – Project's company identifier.
- **missions** – Project's mission identifiers.
- **user** – Project creation user.
- **modification_date** – Project last modification date (format: `YYYY-MM-DDTHH:MM:SS.sssZ`).
- **modification_user** – Project last modification user.
- **real_bbox** – Project bounding box.
- **place_name** – Project place name.

Returns A project resource.

Return type *Project*

Tags

class TagsImpl(*ui_services_api*, *sdk*, ***kwargs*)

create(*name*, ***, *project*, *type*, *target=None*, *flight=None*, ***kwargs*)
Create a tag.

Parameters

- **name** (`str`) – Tag name.
- **project** (`NewType()(ResourceId, str)`) – Identifier of project to tag.
- **type** (`str`) – Tag type (must be one of `project`, `annotation`, `flight`, `photo`, `dataset`, `feature`, `gcp`, `task`).
- **target** (`Optional[NewType()(ResourceId, str)]`) – Optional identifier of the target.
- **flight** (`Optional[NewType()(ResourceId, str)]`) – Optional identifier of the flight (mandatory when the tag type is `photo`).
- ****kwargs** – Optional keyword arguments. Those arguments are passed as is to the API provider.

Returns The created tag.

Return type Tag

Examples

```
>>> sdk.tags.create(
...     name='my tag',
...     project='5d63cf972fb3880011e57f22',
...     type='dataset',
...     target='5d63cf972fb3880011e57e34')
<delairstack.core.resources.tags.Tag with id ... (tags)>
```

delete(*tag*)

Delete a tag.

Parameters

- **tag** (NewType()(ResourceId, str)) – Identifier of the tag to delete.
- ****kwargs** – Optional keyword arguments. Those arguments are passed as is to the API provider.

Examples

```
>>> sdk.tags.delete('5d63cf972fb3880011e57f22')
```

Return type None

search(*, *project*, *type=None*, *target=None*, *flight=None*, ***kwargs*)

Search tags.

When searching for tags on a photo. Both the flight id and target id must be supplied.

Parameters

- **project** (NewType()(ResourceId, str)) – Identifier of the project.
- **type** (Optional[str]) – Optional tag type (must be one of project, annotation, flight, photo, dataset, feature, gcp, task).
- **target** (Optional[NewType()(ResourceId, str)]) – Optional identifier of the target.
- **flight** (Optional[NewType()(ResourceId, str)]) – Optional identifier of the flight (mandatory when the tag type is photo).
- ****kwargs** – Optional keyword arguments. Those arguments are passed as is to the API provider.

Returns The found tags.

Return type Resources

Examples

```
>>> sdk.tags.search(project='5d63cf972fb3880011e57f22')
[<delairstack.core.resources.tags.Tag with id ... (tags)>]
```

CHAPTER

THREE

CHANGELOG

3.1 Changelog

Notable changes to Delair-Stack Python SDK are documented here.

The format is based on Keep a Changelog, and this project adheres to Semantic Versioning.

3.1.1 [1.7.8] - 2020-07-20

Changed

- Specify `utf-8` encoding when opening `README.md` in the `setup.py` (DAI-6185)

Added

- Support creation/deletion/search of credentials in `sdk.credentials` (DAI-6300)

3.1.2 [1.7.7] - 2020-05-14

Changed

- Always delete analytics permanently in `sdk.analytics.delete`, `permanent` parameter is now removed (DAI-5752)

Added

- Add `sdk.products.cancel` to cancel a running analytic product (DAI-5411)

3.1.3 [1.7.6] - 2020-04-24

Changed

- Use tox to execute unit tests against several Python versions (3.4 to 3.8) (DAI-5488)

Added

- Add functions to retrieve the logs of your custom analytic executions `sdk.products.retrieve_logs` and `sdk.products.follow_logs` (DAI-5403)
- Add functions to interact with Analytics and Products: `sdk.analytics` and `sdk.products` (DAI-5460)
- Add `has_headerfile` parameter in `sdk.datasets.create_raster_dataset` to support ENVI header file (DAI-5282)

3.1.4 [1.7.5] - 2020-03-03

Added

- Support the connection through a proxy server (DAI-5047)

3.1.5 [1.7.4] - 2020-02-04

Changed

- Move `CHANGELOG.md` in `docs` and use recommonmark to render it in the documentation (DAI-4868)

Added

- Add a **Getting started Jupyter Notebook** (DAI-4868)

3.1.6 [1.7.3] - 2020-02-03

Changed

- Prepare for public release on Github (DAI-4868)

Added

- Parameter to specify the `domain` at SDK instantiation (DAI-4877)

3.1.7 [1.7.2] - 2020-01-20

Changed

- Show `deleted` and hide `dxobjects` in `Project` resources (DAI-4592)
- Improve the logic to download datasets with various filenames and encodings (update `extract_filename_from_headers` to support attachment headers respecting RFC6266) (DAI-4592)

3.1.8 [1.7.1] - 2020-01-06

Added

- Retry a chunk upload (multipart) when the token is expired (DAI-4363)

3.1.9 [1.7.0] - 2019-12-03

Changed

- `missions.search`: Deprecate the use of the `name` parameter and use UI-Services instead of Project Manager for the other cases (DAI-4204)
- `projects.search`, `projects.delete`: Use UI-Services API instead of Project Manager (DAI-4204)
- `missions.delete`: Use the UI-services API to delete a survey instead of Project Manager (DAI-3855)
- Replace the default configuration file `config-connection.json` by a hardcoded default (DAI-3912)
- Instantiate providers and resources without `config-resources.json` (to allow code completion in the editor) (DAI-3912)
- Make it possible to use various clients (DAI-4009)

Added

- `projects.search`: Add `deleted` parameter to search for deleted projects (DAI-4204)
- Add the `deleted` option in `projects.describe` to allow the description of a deleted project (`False` by default) (DAI-4024)

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

d

INDEX

Symbols

A

B

C

D

F

G

I

M

P

Q

R

S

T

U