

---

# **dtlpy Documentation**

***Release 1.59.21***

**Dataloop Team**

**Jun 29, 2022**



# TABLE OF CONTENTS

<b>1</b>	<b>Command Line Interface</b>	<b>3</b>
1.1	Positional Arguments . . . . .	3
1.2	Named Arguments . . . . .	3
1.3	Sub-commands: . . . . .	3
<b>2</b>	<b>Repositories</b>	<b>21</b>
2.1	Organizations . . . . .	21
2.2	Projects . . . . .	28
2.3	Datasets . . . . .	32
2.4	Items . . . . .	40
2.5	Annotations . . . . .	47
2.6	Recipes . . . . .	51
2.7	Tasks . . . . .	56
2.8	Packages . . . . .	67
2.9	Services . . . . .	78
2.10	Triggers . . . . .	87
2.11	Executions . . . . .	90
2.12	Pipelines . . . . .	94
2.13	General Commands . . . . .	99
<b>3</b>	<b>Entities</b>	<b>101</b>
3.1	Organization . . . . .	101
3.2	Project . . . . .	104
3.3	Dataset . . . . .	107
3.4	Item . . . . .	117
3.5	Annotation . . . . .	121
3.6	Filter . . . . .	136
3.7	Recipe . . . . .	139
3.8	Task . . . . .	144
3.9	Package . . . . .	150
3.10	Service . . . . .	154
3.11	Trigger . . . . .	160
3.12	Execution . . . . .	162
3.13	Pipeline . . . . .	163
3.14	Other . . . . .	166
<b>4</b>	<b>Utilities</b>	<b>169</b>
4.1	converter . . . . .	169
<b>5</b>	<b>Tutorials</b>	<b>175</b>

5.1	Data Management Tutorial . . . . .	175
5.2	FaaS Tutorial . . . . .	209
5.3	Task Workflows . . . . .	218
5.4	Image Annotations . . . . .	234
5.5	Video Annotations . . . . .	243
5.6	Recipe and Ontology . . . . .	246
5.7	Model Management . . . . .	251
<b>6</b>	<b>Indices and tables</b>	<b>259</b>
	<b>Python Module Index</b>	<b>261</b>
	<b>Index</b>	<b>263</b>

Drive your AI to production with end-to-end data management, automation pipelines and a quality-first data labeling platform



## COMMAND LINE INTERFACE

Options:

CLI for Dataloop

```
usage: dlp [-h] [-v]
           {shell,upgrade,logout,login,login-token,login-secret,login-m2m,init,checkout-
↪state,help,version,api,projects,datasets,items,videos,services,triggers,deploy,
↪generate,packages,ls,pwd,cd,mkdir,clear,exit}
           * * *
```

### 1.1 Positional Arguments

<b>operation</b>	Possible choices: shell, upgrade, logout, login, login-token, login-secret, login-m2m, init, checkout-state, help, version, api, projects, datasets, items, videos, services, triggers, deploy, generate, packages, ls, pwd, cd, mkdir, clear, exit
	supported operations

### 1.2 Named Arguments

<b>-v, --version</b>	dtlpy version
	Default: False

### 1.3 Sub-commands:

#### 1.3.1 shell

Open interactive Dataloop shell

```
dlp shell [-h]
```

### 1.3.2 upgrade

Update dtlpy package

```
dlp upgrade [-h] [-u ]
```

#### optional named arguments

**-u, --url**                Package url. default 'dtlpy'

### 1.3.3 logout

Logout

```
dlp logout [-h]
```

### 1.3.4 login

Login using web Auth0 interface

```
dlp login [-h]
```

### 1.3.5 login-token

Login by passing a valid token

```
dlp login-token [-h] -t
```

#### required named arguments

**-t, --token**                valid token

### 1.3.6 login-secret

Login client id and secret

```
dlp login-secret [-h] [-e ] [-p ] [-i ] [-s ]
```



#### required named arguments

<b>-e, --email</b>	user email
<b>-p, --password</b>	user password
<b>-i, --client-id</b>	client id
<b>-s, --client-secret</b>	client secret

### 1.3.7 login-m2m

Login client id and secret

```
dlp login-m2m [-h] [-e ] [-p ] [-i ] [-s ]
```

#### required named arguments

<b>-e, --email</b>	user email
<b>-p, --password</b>	user password
<b>-i, --client-id</b>	client id
<b>-s, --client-secret</b>	client secret

### 1.3.8 init

Initialize a .dataloop context

```
dlp init [-h]
```

### 1.3.9 checkout-state

Print checkout state

```
dlp checkout-state [-h]
```

### 1.3.10 help

Get help

```
dlp help [-h]
```

### 1.3.11 version

DTLPY SDK version

```
dlp version [-h]
```

### 1.3.12 api

Connection and environment

```
dlp api [-h] {info,setenv} ...
```

#### Positional Arguments

<b>api</b>	Possible choices: info, setenv gate operations
------------	---

#### Sub-commands:

##### info

Print api information

```
dlp api info [-h]
```

##### setenv

Set platform environment

```
dlp api setenv [-h] -e
```

#### required named arguments

<b>-e, --env</b>	working environment
------------------	---------------------

### 1.3.13 projects

Operations with projects

```
dlp projects [-h] {ls,create,checkout,web} ...
```

## Positional Arguments

**projects**                      Possible choices: ls, create, checkout, web  
projects operations

### Sub-commands:

#### ls

List all projects

```
dlp projects ls [-h]
```

#### create

Create a new project

```
dlp projects create [-h] [-p ]
```

### required named arguments

**-p, --project-name**    project name

#### checkout

checkout a project

```
dlp projects checkout [-h] [-p ]
```

### required named arguments

**-p, --project-name**    project name

#### web

Open in web browser

```
dlp projects web [-h] [-p ]
```

### optional named arguments

**-p, --project-name** project name

## 1.3.14 datasets

Operations with datasets

```
dlp datasets [-h] {web,ls,create,checkout} ...
```

### Positional Arguments

**datasets** Possible choices: web, ls, create, checkout  
datasets operations

### Sub-commands:

#### web

Open in web browser

```
dlp datasets web [-h] [-p ] [-d ]
```

### optional named arguments

**-p, --project-name** project name  
**-d, --dataset-name** dataset name

#### ls

List of datasets in project

```
dlp datasets ls [-h] [-p ]
```

### optional named arguments

**-p, --project-name** project name. Default taken from checked out (if checked out)

## create

Create a new dataset

```
dlp datasets create [-h] -d [-p ] [-c]
```

### required named arguments

**-d, --dataset-name** dataset name

### optional named arguments

**-p, --project-name** project name. Default taken from checked out (if checked out)

**-c, --checkout** checkout the new dataset

Default: False

## checkout

checkout a dataset

```
dlp datasets checkout [-h] [-d ] [-p ]
```

### required named arguments

**-d, --dataset-name** dataset name

### optional named arguments

**-p, --project-name** project name. Default taken from checked out (if checked out)

## 1.3.15 items

Operations with items

```
dlp items [-h] {web,ls,upload,download} ...
```

## Positional Arguments

**items**                      Possible choices: web, ls, upload, download  
items operations

## Sub-commands:

### web

Open in web browser

```
dlp items web [-h] [-r ] [-p ] [-d ]
```

## required named arguments

**-r, --remote-path**      remote path

## optional named arguments

**-p, --project-name**    project name

**-d, --dataset-name**    dataset name

### ls

List of items in dataset

```
dlp items ls [-h] [-p ] [-d ] [-o ] [-r ] [-t ]
```

## optional named arguments

**-p, --project-name**    project name. Default taken from checked out (if checked out)

**-d, --dataset-name**    dataset name. Default taken from checked out (if checked out)

**-o, --page**            page number (integer)

Default: 0

**-r, --remote-path**    remote path

**-t, --type**            Item type

## upload

Upload directory to dataset

```
dlp items upload [-h] -l [-p ] [-d ] [-r ] [-f ] [-lap ] [-ow]
```

### required named arguments

**-l, --local-path** local path

### optional named arguments

**-p, --project-name** project name. Default taken from checked out (if checked out)

**-d, --dataset-name** dataset name. Default taken from checked out (if checked out)

**-r, --remote-path** remote path to upload to. default: /

**-f, --file-types** Comma separated list of file types to upload, e.g “.jpg,.png”. default: all

**-lap, --local-annotations-path** Path for local annotations to upload with items

**-ow, --overwrite** Overwrite existing item  
Default: False

## download

Download dataset to a local directory

```
dlp items download [-h] [-p ] [-d ] [-ao ] [-aft ] [-afl ] [-r ] [-ow]
                    [-t ] [-wt ] [-th ] [-l ] [-wb]
```

### optional named arguments

**-p, --project-name** project name. Default taken from checked out (if checked out)

**-d, --dataset-name** dataset name. Default taken from checked out (if checked out)

**-ao, --annotation-options** which annotation to download. options: json,instance,mask

**-aft, --annotation-filter-type** annotation type filter when downloading annotations. options: box,segment,binary etc

**-afl, --annotation-filter-label** labels filter when downloading annotations.

**-r, --remote-path** remote path to upload to. default: /

**-ow, --overwrite** Overwrite existing item  
Default: False

**-t, --not-items-folder** Download WITHOUT ‘items’ folder  
Default: False

**-wt, --with-text** Annotations will have text in mask  
Default: False

**-th, --thickness** Annotation line thickness  
Default: “1”

**-l, --local-path** local path

**-wb, --without-binaries** Don’t download item binaries  
Default: False

### 1.3.16 videos

Operations with videos

```
dlp videos [-h] {play,upload} ...
```

#### Positional Arguments

**videos** Possible choices: play, upload  
videos operations

#### Sub-commands:

##### play

Play video

```
dlp videos play [-h] [-l ] [-p ] [-d ]
```

#### optional named arguments

**-l, --item-path** Video remote path in platform. e.g /dogs/dog.mp4

**-p, --project-name** project name. Default taken from checked out (if checked out)

**-d, --dataset-name** dataset name. Default taken from checked out (if checked out)

##### upload

Upload a single video

```
dlp videos upload [-h] -f -p -d [-r ] [-sc ] [-ss ] [-st ] [-e]
```



### required named arguments

<b>-f, --filename</b>	local filename to upload
<b>-p, --project-name</b>	project name
<b>-d, --dataset-name</b>	dataset name

### optional named arguments

<b>-r, --remote-path</b>	remote path Default: "/"
<b>-sc, --split-chunks</b>	Video splitting parameter: Number of chunks to split
<b>-ss, --split-seconds</b>	Video splitting parameter: Seconds of each chunk
<b>-st, --split-times</b>	Video splitting parameter: List of seconds to split at. e.g 600,1800,2000
<b>-e, --encode</b>	encode video to mp4, remove bframes and upload Default: False

## 1.3.17 services

Operations with services

```
dlp services [-h] {execute,tear-down,ls,log,delete} ...
```

### Positional Arguments

<b>services</b>	Possible choices: execute, tear-down, ls, log, delete services operations
-----------------	--

### Sub-commands:

#### execute

Create an execution

```
dlp services execute [-h] [-f FUNCTION_NAME] [-s SERVICE_NAME]
                    [-pr PROJECT_NAME] [-as] [-i ITEM_ID] [-d DATASET_ID]
                    [-a ANNOTATION_ID] [-in INPUTS]
```

### optional named arguments

<b>-f, --function-name</b>	which function to run
<b>-s, --service-name</b>	which service to run
<b>-pr, --project-name</b>	Project name
<b>-as, --async</b>	Async execution
	Default: True
<b>-i, --item-id</b>	Item input
<b>-d, --dataset-id</b>	Dataset input
<b>-a, --annotation-id</b>	Annotation input
<b>-in, --inputs</b>	Dictionary string input
	Default: "{}"

### tear-down

tear-down service of service.json file

```
dlp services tear-down [-h] [-l LOCAL_PATH] [-pr PROJECT_NAME]
```

### optional named arguments

<b>-l, --local-path</b>	path to service.json file
<b>-pr, --project-name</b>	Project name

### ls

List project's services

```
dlp services ls [-h] [-pr PROJECT_NAME] [-pkg PACKAGE_NAME]
```

### optional named arguments

<b>-pr, --project-name</b>	Project name
<b>-pkg, --package-name</b>	Package name

## log

Get services log

```
dlp services log [-h] [-pr PROJECT_NAME] [-f SERVICE_NAME] [-t START]
```

### required named arguments

<b>-pr, --project-name</b>	Project name
<b>-f, --service-name</b>	Project name
<b>-t, --start</b>	Log start time

## delete

Delete Service

```
dlp services delete [-h] [-f SERVICE_NAME] [-p PROJECT_NAME]
                    [-pkg PACKAGE_NAME]
```

### optional named arguments

<b>-f, --service-name</b>	Service name
<b>-p, --project-name</b>	Project name
<b>-pkg, --package-name</b>	Package name

## 1.3.18 triggers

Operations with triggers

```
dlp triggers [-h] {create,delete,ls} ...
```

### Positional Arguments

<b>triggers</b>	Possible choices: create, delete, ls triggers operations
-----------------	---

## Sub-commands:

### create

Create a Service Trigger

```
dlp triggers create [-h] -r RESOURCE -a ACTIONS [-p PROJECT_NAME]
                  [-pkg PACKAGE_NAME] [-f SERVICE_NAME] [-n NAME]
                  [-fl FILTERS] [-fn FUNCTION_NAME]
```

### required named arguments

<b>-r, --resource</b>	Resource name
<b>-a, --actions</b>	Actions

### optional named arguments

<b>-p, --project-name</b>	Project name
<b>-pkg, --package-name</b>	Package name
<b>-f, --service-name</b>	Service name
<b>-n, --name</b>	Trigger name
<b>-fl, --filters</b>	Json filter
	Default: “{}”
<b>-fn, --function-name</b>	Function name
	Default: “run”

### delete

Delete Trigger

```
dlp triggers delete [-h] -t TRIGGER_NAME [-f SERVICE_NAME] [-p PROJECT_NAME]
                  [-pkg PACKAGE_NAME]
```

### required named arguments

<b>-t, --trigger-name</b>	Trigger name
---------------------------	--------------

### optional named arguments

**-f, --service-name** Service name  
**-p, --project-name** Project name  
**-pkg, --package-name** Package name

### ls

List triggers

```
dlp triggers ls [-h] [-pr PROJECT_NAME] [-pkg PACKAGE_NAME] [-s SERVICE_NAME]
```

### optional named arguments

**-pr, --project-name** Project name  
**-pkg, --package-name** Package name  
**-s, --service-name** Service name

## 1.3.19 deploy

deploy with json file

```
dlp deploy [-h] [-f JSON_FILE] [-p PROJECT_NAME]
```

### required named arguments

**-f** Path to json file  
**-p** Project name

## 1.3.20 generate

generate a json file

```
dlp generate [-h] [--option PACKAGE_TYPE] [-p PACKAGE_NAME]
```

### optional named arguments

**--option** cataluge of examples  
**-p, --package-name** Package name

### 1.3.21 packages

Operations with packages

```
dlp packages [-h] {ls,push,test,checkout,delete} ...
```

#### Positional Arguments

<b>packages</b>	Possible choices: ls, push, test, checkout, delete package operations
-----------------	--

#### Sub-commands:

##### ls

List packages

```
dlp packages ls [-h] [-p PROJECT_NAME]
```

#### optional named arguments

<b>-p, --project-name</b>	Project name
---------------------------	--------------

##### push

Create package in platform

```
dlp packages push [-h] [-src ] [-cid ] [-pr ] [-p ]
```

#### optional named arguments

<b>-src, --src-path</b>	Revision to deploy if selected True
<b>-cid, --codebase-id</b>	Revision to deploy if selected True
<b>-pr, --project-name</b>	Project name
<b>-p, --package-name</b>	Package name

##### test

Tests that Package locally using mock.json

```
dlp packages test [-h] [-c ] [-f ]
```

### optional named arguments

- c, --concurrency** Revision to deploy if selected True  
Default: 10
- f, --function-name** Function to test  
Default: "run"

### checkout

checkout a package

```
dlp packages checkout [-h] [-p ]
```

### required named arguments

- p, --package-name** package name

### delete

Delete Package

```
dlp packages delete [-h] [-pkg PACKAGE_NAME] [-p PROJECT_NAME]
```

### optional named arguments

- pkg, --package-name** Package name
- p, --project-name** Project name

## 1.3.22 ls

List directories

```
dlp ls [-h]
```

## 1.3.23 pwd

Get current working directory

```
dlp pwd [-h]
```

### 1.3.24 cd

Change current working directory

```
dlp cd [-h] dir
```

#### Positional Arguments

**dir**

### 1.3.25 mkdir

Make directory

```
dlp mkdir [-h] name
```

#### Positional Arguments

**name**

### 1.3.26 clear

Clear shell

```
dlp clear [-h]
```

### 1.3.27 exit

Exit interactive shell

```
dlp exit [-h]
```



## REPOSITORIES

### 2.1 Organizations

**class** `Organizations`(*client\_api: ApiClient*)

Bases: `object`

Organizations Repository

Read our [documentation](#) and [SDK documentation](#) to learn more about Organizations in the Dataloop platform.

**add\_member**(*email: str*, *role: MemberOrgRole = MemberOrgRole.MEMBER*, *organization\_id: Optional[str] = None*, *organization\_name: Optional[str] = None*, *organization: Optional[Organization] = None*)

Add members to your organization. Read about members and groups [here](#).

**Prerequisites:** To add members to an organization, you must be an *owner* in that organization.

You must provide at least ONE of the following params: *organization*, *organization\_name*, or *organization\_id*.

#### Parameters

- **email** (*str*) – the member’s email
- **role** (*str*) – `MemberOrgRole.ADMIN`, `MemberOrgRole.OWNER`, `MemberOrgRole.MEMBER`
- **organization\_id** (*str*) – Organization id
- **organization\_name** (*str*) – Organization name
- **organization** (*entities.Organization*) – Organization object

#### Returns

True if successful or error if unsuccessful

#### Return type

`bool`

#### Example:

```
dl.organizations.add_member(email='user@domain.com',
                             organization_id='organization_id',
                             role=dl.MemberOrgRole.MEMBER)
```

```
cache_action(organization_id: Optional[str] = None, organization_name: Optional[str] = None,
              organization: Optional[Organization] = None, mode=CacheAction.APPLY,
              pod_type=PodType.SMALL)
```

Add or remove Cache for the org

**Prerequisites:** You must be an organization *owner*

You must provide at least ONE of the following params: organization, organization\_name, or organization\_id.

#### Parameters

- **organization\_id** (*str*) – Organization id
- **organization\_name** (*str*) – Organization name
- **organization** (*entities.Organization*) – Organization object
- **mode** (*str*) – dl.CacheAction.APPLY or dl.CacheAction.DESTROY
- **pod\_type** (*entities.PodType*) – dl.PodType.SMALL, dl.PodType.MEDIUM, dl.PodType.HIGH

#### Returns

True if success

#### Return type

*bool*

**Example:**

```
dl.organizations.enable_cache(organization_id='organization_id',
                              mode=dl.CacheAction.APPLY)
```

```
delete_member(user_id: str, organization_id: Optional[str] = None, organization_name: Optional[str] =
              None, organization: Optional[Organization] = None, sure: bool = False, really: bool =
              False) → bool
```

Delete member from the Organization.

**Prerequisites:** Must be an organization *owner* to delete members.

You must provide at least ONE of the following params: organization\_id, organization\_name, organization.

#### Parameters

- **user\_id** (*str*) – user id
- **organization\_id** (*str*) – Organization id
- **organization\_name** (*str*) – Organization name
- **organization** (*entities.Organization*) – Organization object
- **sure** (*bool*) – Are you sure you want to delete?
- **really** (*bool*) – Really really sure?

#### Returns

True if success and error if not

#### Return type

*bool*

**Example:**

```
dl.organizations.delete_member(user_id='user_id',
                               organization_id='organization_id',
                               sure=True,
                               really=True)
```

**get**(*organization\_id: Optional[str] = None, organization\_name: Optional[str] = None, fetch: Optional[bool] = None*) → *Organization*

Get Organization object to be able to use it in your code.

**Prerequisites:** You must be a **superuser** to use this method.

You must provide at least ONE of the following params: `organization_name` or `organization_id`.

#### Parameters

- **organization\_id** (*str*) – optional - search by id
- **organization\_name** (*str*) – optional - search by name
- **fetch** – optional - fetch entity from platform, default taken from cookie

#### Returns

Organization object

#### Return type

*dtlpy.entities.organization.Organization*

#### Example:

```
dl.organizations.get(organization_id='organization_id')
```

**list**() → List[*Organization*]

Lists all the organizations in Dataloop.

**Prerequisites:** You must be a **superuser** to use this method.

#### Returns

List of Organization objects

#### Return type

*list*

#### Example:

```
dl.organizations.list()
```

**list\_groups**(*organization: Optional[Organization] = None, organization\_id: Optional[str] = None, organization\_name: Optional[str] = None*)

List all organization groups (groups that were created within the organization).

**Prerequisites:** You must be an organization *owner* to use this method.

You must provide at least ONE of the following params: `organization`, `organization_name`, or `organization_id`.

#### Parameters

- **organization** (*entities.Organization*) – Organization object
- **organization\_id** (*str*) – Organization id
- **organization\_name** (*str*) – Organization name

**Returns**

groups list

**Return type**

list

**Example:**

```
dl.organizations.list_groups(organization_id='organization_id')
```

**list\_integrations**(*organization*: *Optional*[*Organization*] = *None*, *organization\_id*: *Optional*[*str*] = *None*, *organization\_name*: *Optional*[*str*] = *None*, *only\_available*=*False*)

List all organization integrations with external cloud storage.

**Prerequisites:** You must be an organization *owner* to use this method.

You must provide at least ONE of the following params: *organization\_id*, *organization\_name*, or *organization*.

**Parameters**

- **organization** (*entities.Organization*) – Organization object
- **organization\_id** (*str*) – Organization id
- **organization\_name** (*str*) – Organization name
- **only\_available** (*bool*) – if True list only the available integrations

**Returns**

integrations list

**Return type**

list

**Example:**

```
dl.organizations.list_integrations(organization='organization-entity',  
                                   only_available=True)
```

**list\_members**(*organization*: *Optional*[*Organization*] = *None*, *organization\_id*: *Optional*[*str*] = *None*, *organization\_name*: *Optional*[*str*] = *None*, *role*: *Optional*[*MemberOrgRole*] = *None*)

List all organization members.

**Prerequisites:** You must be an organization *owner* to use this method.

You must provide at least ONE of the following params: *organization\_id*, *organization\_name*, or *organization*.

**Parameters**

- **organization** (*entities.Organization*) – Organization object
- **organization\_id** (*str*) – Organization id
- **organization\_name** (*str*) – Organization name
- **role** (*entities.MemberOrgRole*) – MemberOrgRole.ADMIN, MemberOrgRole.OWNER, MemberOrgRole.MEMBER

**Returns**

projects list

**Return type**

list

**Example:**

```
dl.organizations.list_members(organization='organization-entity',
                              role=dl.MemberOrgRole.MEMBER)
```

**update**(*plan*: str, *organization*: Optional[Organization] = None, *organization\_id*: Optional[str] = None, *organization\_name*: Optional[str] = None) → Organization

Update an organization.

**Prerequisites:** You must be a **superuser** to update an organization.

You must provide at least ONE of the following params: organization, organization\_name, or organization\_id.

**Parameters**

- **plan** (str) – OrganizationsPlans.FREEMIUM, OrganizationsPlans.PREMIUM
- **organization** (entities.Organization) – Organization object
- **organization\_id** (str) – Organization id
- **organization\_name** (str) – Organization name

**Returns**

organization object

**Return type**

dtlpy.entities.organization.Organization

**Example:**

```
dl.organizations.update(organization='organization-entity',
                        plan=dl.OrganizationsPlans.FREEMIUM)
```

**update\_member**(*email*: str, *role*: MemberOrgRole = MemberOrgRole.MEMBER, *organization\_id*: Optional[str] = None, *organization\_name*: Optional[str] = None, *organization*: Optional[Organization] = None)

Update member role.

**Prerequisites:** You must be an organization *owner* to update a member's role.

You must provide at least ONE of the following params: organization, organization\_name, or organization\_id.

**Parameters**

- **email** (str) – the member's email
- **role** (str) – MemberOrgRole.ADMIN, MemberOrgRole.OWNER, MemberOrgRole.MEMBER
- **organization\_id** (str) – Organization id
- **organization\_name** (str) – Organization name
- **organization** (entities.Organization) – Organization object

**Returns**

json of the member fields

**Return type**`dict`**Example:**

```
dl.organizations.update_member(email='user@domain.com',
                               organization_id='organization_id',
                               role=dl.MemberOrgRole.MEMBER)
```

## 2.1.1 Integrations

### Integrations Repository

```
class Integrations(client_api: ApiClient, org: Optional[Organization] = None, project: Optional[Project] = None)
```

Bases: `object`

#### Integrations Repository

The Integrations class allows you to manage data integration from your external storage (e.g., S3, GCS, Azure) into your Dataloop's Dataset storage, as well as sync data in your Dataloop's Datasets with data in your external storage.

For more information on Organization Storage Integration see the [Dataloop documentation](#) and [SDK External Storage](#).

```
create(integrations_type: ExternalStorage, name: str, options: dict)
```

Create an integration between an external storage and the organization.

**Examples for options include:** s3 - {key: "", secret: ""}; gcs - {key: "", secret: "", content: ""}; azureblob - {key: "", secret: "", clientId: "", tenantId: ""}; key\_value - {key: "", value: ""} aws-sts - {key: "", secret: "", roleArns: ""}

**Prerequisites:** You must be an *owner* in the organization.

#### Parameters

- **integrations\_type** (*str*) – integrations type `dl.ExternalStorage`
- **name** (*str*) – integrations name
- **options** (*dict*) – dict of storage secrets

#### Returns

`success`

#### Return type

`bool`**Example:**

```
project.integrations.create(integrations_type=dl.ExternalStorage.S3,
                             name='S3integration',
                             options={key: "Access key ID", secret: "Secret access key"})
```

```
delete(integrations_id: str, sure: bool = False, really: bool = False) → bool
```

Delete integrations from the organization.

**Prerequisites:** You must be an organization *owner* to delete an integration.

#### Parameters

- **integrations\_id** (*str*) – integrations id
- **sure** (*bool*) – Are you sure you want to delete?
- **really** (*bool*) – Really really sure?

**Returns**

success

**Return type***bool***Example:**

```
project.integrations.delete(integrations_id='integrations_id', sure=True,
↪really=True)
```

**get**(*integrations\_id: str*)

Get organization integrations. Use this method to access your integration and be able to use it in your code.

**Prerequisites:** You must be an *owner* in the organization.

**Parameters**

**integrations\_id** (*str*) – integrations id

**Returns**

Integration object

**Return type***dtlpy.entities.integration.Integration***Example:**

```
project.integrations.get(integrations_id='integrations_id')
```

**list**(*only\_available=False*)

List all the organization's integrations with external storage.

**Prerequisites:** You must be an *owner* in the organization.

**Parameters**

**only\_available** (*bool*) – if True list only the available integrations.

**Returns**

groups list

**Return type***list***Example:**

```
project.integrations.list(only_available=True)
```

**update**(*new\_name: str, integrations\_id: str*)

Update the integration's name.

**Prerequisites:** You must be an *owner* in the organization.

**Parameters**

- **new\_name** (*str*) – new name
- **integrations\_id** (*str*) – integrations id

**Returns**

Integration object

**Return type***dtlpy.entities.integration.Integration***Example:**

```
project.integrations.update(integrations_id='integrations_id', new_name="new_
↪integration_name")
```

## 2.2 Projects

**class Projects**(*client\_api: ApiClient, org=None*)Bases: `object`

Projects Repository

The Projects class allows the user to manage projects and their properties.

For more information on Projects see the [Dataloop documentation](#) and [SDK documentation](#).**add\_member**(*email: str, project\_id: str, role: MemberRole = MemberRole.DEVELOPER*)

Add a member to the project.

**Prerequisites:** You must be in the role of an *owner* to add a member to a project.**Parameters**

- **email** (*str*) – member email
- **project\_id** (*str*) – project id
- **role** – `dl.MemberRole.OWNER`, `dl.MemberRole.DEVELOPER`,  
`dl.MemberRole.ANNOTATOR`, `dl.MemberRole.ANNOTATION_MANAGER`

**Returns**

dict that represent the user

**Return type**`dict`**Example:**

```
dl.projects.add_member(project_id='project_id', email='user@dataloop.ai',
↪role=dl.MemberRole.DEVELOPER)
```

**checkout**(*identifier: Optional[str] = None, project\_name: Optional[str] = None, project\_id: Optional[str] = None, project: Optional[Project] = None*)

Checkout (switch) to a project to work on it.

**Prerequisites:** All users can open a project in the web.You must provide at least ONE of the following params: `project_id`, `project_name`.**Parameters**

- **identifier** (*str*) – project name or partial id
- **project\_name** (*str*) – project name



- **project\_id** (*str*) – project id
- **project** (*dtlpy.entities.project.Project*) – project entity

**Example:**

```
dl.projects.checkout(project_id='project_id')
```

**create**(*project\_name: str, checkout: bool = False*) → *Project*

Create a new project.

**Prerequisites:** Any user can create a project.

#### Parameters

- **project\_name** (*str*) – project name
- **checkout** – checkout

#### Returns

Project object

#### Return type

*dtlpy.entities.project.Project*

**Example:**

```
dl.projects.create(project_name='project_name')
```

**delete**(*project\_name: Optional[str] = None, project\_id: Optional[str] = None, sure: bool = False, really: bool = False*) → *bool*

Delete a project forever!

**Prerequisites:** You must be in the role of an *owner* to delete a project.

#### Parameters

- **project\_name** (*str*) – optional - search by name
- **project\_id** (*str*) – optional - search by id
- **sure** (*bool*) – Are you sure you want to delete?
- **really** (*bool*) – Really really sure?

#### Returns

True if success error if not

#### Return type

*bool*

**Example:**

```
dl.projects.delete(project_id='project_id', sure=True, really=True)
```

**get**(*project\_name: Optional[str] = None, project\_id: Optional[str] = None, checkout: bool = False, fetch: Optional[bool] = None, log\_error=True*) → *Project*

Get a Project object.

**Prerequisites:** You must be in the role of an *owner* to get a project object.

You must check out to a project or provide at least one of the following params: *project\_id*, *project\_name*

#### Parameters

- **project\_name** (*str*) – optional - search by name
- **project\_id** (*str*) – optional - search by id
- **checkout** (*bool*) – checkout
- **fetch** (*bool*) – optional - fetch entity from platform, default taken from cookie
- **log\_error** (*bool*) – optional - show the logs errors

**Returns**

Project object

**Return type**

*dtlpy.entities.project.Project*

**Example:**

```
dl.projects.get(project_id='project_id')
```

**list()** → List[*Project*]

Get users' project list.

**Prerequisites:** You must be a **superuser** to list all users' projects.

**Returns**

List of Project objects

**Example:**

```
dl.projects.list()
```

**list\_members**(*project*: *Project*, *role*: *Optional*[*MemberRole*] = *None*)

List the project members.

**Prerequisites:** You must be in the role of an *owner* to list project members.

**Parameters**

- **project** (*dtlpy.entities.project.Project*) – project entity
- **role** – *dl.MemberRole.OWNER*, *dl.MemberRole.DEVELOPER*,  
*dl.MemberRole.ANNOTATOR*, *dl.MemberRole.ANNOTATION\_MANAGER*

**Returns**

list of the project members

**Return type**

*list*

**Example:**

```
dl.projects.list_members(project_id='project_id', role=dl.MemberRole.DEVELOPER)
```

**open\_in\_web**(*project\_name*: *Optional*[*str*] = *None*, *project\_id*: *Optional*[*str*] = *None*, *project*:  
*Optional*[*Project*] = *None*)

Open the project in our web platform.

**Prerequisites:** All users can open a project in the web.

**Parameters**

- **project\_name** (*str*) – project name

- **project\_id** (*str*) – project id
- **project** (`dtlpy.entities.project.Project`) – project entity

Example:

```
dl.projects.open_in_web(project_id='project_id')
```

**remove\_member**(*email: str, project\_id: str*)

Remove a member from the project.

**Prerequisites:** You must be in the role of an *owner* to delete a member from a project.

#### Parameters

- **email** (*str*) – member email
- **project\_id** (*str*) – project id

#### Returns

dict that represents the user

#### Return type

dict

Example:

```
dl.projects.remove_member(project_id='project_id', email='user@dataloop.ai')
```

**update**(*project: Project, system\_metadata: bool = False*) → *Project*

Update a project information (e.g., name, member roles, etc.).

**Prerequisites:** You must be in the role of an *owner* to add a member to a project.

#### Parameters

- **project** (`dtlpy.entities.project.Project`) – project entity
- **system\_metadata** (*bool*) – True, if you want to change metadata system

#### Returns

Project object

#### Return type

`dtlpy.entities.project.Project`

Example:

```
dl.projects.delete(project='project_entity')
```

**update\_member**(*email: str, project\_id: str, role: MemberRole = MemberRole.DEVELOPER*)

Update member's information/details in the project.

**Prerequisites:** You must be in the role of an *owner* to update a member.

#### Parameters

- **email** (*str*) – member email
- **project\_id** (*str*) – project id
- **role** – `dl.MemberRole.OWNER`, `dl.MemberRole.DEVELOPER`, `dl.MemberRole.ANNOTATOR`, `dl.MemberRole.ANNOTATION_MANAGER`

**Returns**

dict that represent the user

**Return type**

dict

**Example:**

```
dl.projects.update_member(project_id='project_id', email='user@dataloop.ai',  
↪role=dl.MemberRole.DEVELOPER)
```

## 2.3 Datasets

Datasets Repository

**class Datasets**(*client\_api: ApiClient, project: Optional[Project] = None*)

Bases: `object`

Datasets Repository

The Datasets class allows the user to manage datasets. Read more about datasets in our [documentation](#) and [SDK documentation](#).

**checkout**(*identifier: Optional[str] = None, dataset\_name: Optional[str] = None, dataset\_id: Optional[str] = None, dataset: Optional[Dataset] = None*)

Checkout (switch) to a dataset to work on it.

**Prerequisites:** You must be an *owner* or *developer* to use this method.

You must provide at least ONE of the following params: `dataset_id`, `dataset_name`.

**Parameters**

- **identifier** (*str*) – project name or partial id
- **dataset\_name** (*str*) – dataset name
- **dataset\_id** (*str*) – dataset id
- **dataset** (`dtlpy.entities.dataset.Dataset`) – dataset object

**Example:**

```
project.datasets.checkout(dataset_id='dataset_id')
```

**clone**(*dataset\_id: str, clone\_name: str, filters: Optional[Filters] = None, with\_items\_annotations: bool = True, with\_metadata: bool = True, with\_task\_annotations\_status: bool = True*)

Clone a dataset. Read more about cloning datasets and items in our [documentation](#) and [SDK documentation](#).

**Prerequisites:** You must be in the role of an *owner* or *developer*.

**Parameters**

- **dataset\_id** (*str*) – id of the dataset you wish to clone
- **clone\_name** (*str*) – new dataset name
- **filters** (`dtlpy.entities.filters.Filters`) – Filters entity or a query dict
- **with\_items\_annotations** (*bool*) – true to clone with items annotations

- **with\_metadata** (*bool*) – true to clone with metadata
- **with\_task\_annotations\_status** (*bool*) – true to clone with task annotations' status

**Returns**

dataset object

**Return type***dtlpy.entities.dataset.Dataset***Example:**

```
project.datasets.clone(dataset_id='dataset_id',
                       clone_name='dataset_clone_name',
                       with_metadata=True,
                       with_items_annotations=False,
                       with_task_annotations_status=False)
```

**create**(*dataset\_name: str*, *labels=None*, *attributes=None*, *ontology\_ids=None*, *driver: Optional[Driver] = None*, *driver\_id: Optional[str] = None*, *checkout: bool = False*, *expiration\_options: Optional[ExpirationOptions] = None*, *index\_driver: IndexDriver = IndexDriver.V1*, *recipe\_id: Optional[str] = None*) → *Dataset*

Create a new dataset

**Prerequisites:** You must be in the role of an *owner* or *developer*.**Parameters**

- **dataset\_name** (*str*) – dataset name
- **labels** (*list*) – dictionary of {tag: color} or list of label entities
- **attributes** (*list*) – dataset's ontology's attributes
- **ontology\_ids** (*list*) – optional - dataset ontology
- **driver** (*dtlpy.entities.driver.Driver*) – optional - storage driver Driver object or driver name
- **driver\_id** (*str*) – optional - driver id
- **checkout** (*bool*) – bool. cache the dataset to work locally
- **expiration\_options** (*ExpirationOptions*) – dl.ExpirationOptions object that contain definitions for dataset like MaxItemDays
- **index\_driver** (*str*) – dl.IndexDriver, dataset driver version
- **recipe\_id** (*str*) – optional - recipe id

**Returns**

Dataset object

**Return type***dtlpy.entities.dataset.Dataset***Example:**

```
project.datasets.create(dataset_name='dataset_name', ontology_ids='ontology_ids
↪')
```

**delete**(*dataset\_name*: *Optional[str]* = None, *dataset\_id*: *Optional[str]* = None, *sure*: *bool* = False, *really*: *bool* = False)

Delete a dataset forever!

**Prerequisites:** You must be an *owner* or *developer* to use this method.

**Example:**

```
project.datasets.delete(dataset_id='dataset_id', sure=True, really=True)
```

#### Parameters

- **dataset\_name** (*str*) – optional - search by name
- **dataset\_id** (*str*) – optional - search by id
- **sure** (*bool*) – Are you sure you want to delete?
- **really** (*bool*) – Really really sure?

#### Returns

True is success

#### Return type

*bool*

**directory\_tree**(*dataset*: *Optional[Dataset]* = None, *dataset\_name*: *Optional[str]* = None, *dataset\_id*: *Optional[str]* = None)

Get dataset's directory tree.

**Prerequisites:** You must be an *owner* or *developer* to use this method.

You must provide at least ONE of the following params: *dataset*, *dataset\_name*, *dataset\_id*.

#### Parameters

- **dataset** (*dtlpy.entities.dataset.Dataset*) – dataset object
- **dataset\_name** (*str*) – dataset name
- **dataset\_id** (*str*) – dataset id

#### Returns

DirectoryTree

**Example:**

```
project.datasets.directory_tree(dataset='dataset_entity')
```

**static download\_annotations**(*dataset*: *Dataset*, *local\_path*: *Optional[str]* = None, *filters*: *Optional[Filters]* = None, *annotation\_options*: *Optional[ViewAnnotationOptions]* = None, *annotation\_filters*: *Optional[Filters]* = None, *overwrite*: *bool* = False, *thickness*: *int* = 1, *with\_text*: *bool* = False, *remote\_path*: *Optional[str]* = None, *include\_annotations\_in\_output*: *bool* = True, *export\_png\_files*: *bool* = False, *filter\_output\_annotations*: *bool* = False, *alpha*: *Optional[float]* = None, *export\_version*=*ExportVersion.V1*) → *str*

Download dataset's annotations by filters.

You may filter the dataset both for items and for annotations and download annotations.

Optional – download annotations as: mask, instance, image mask of the item.

**Prerequisites:** You must be in the role of an *owner* or *developer*.

#### Parameters

- **dataset** (`dtlpy.entities.dataset.Dataset`) – dataset object
- **local\_path** (`str`) – local folder or filename to save to.
- **filters** (`dtlpy.entities.filters.Filters`) – Filters entity or a dictionary containing filters parameters
- **annotation\_options** (`list`) – download annotations options: list(`dl.ViewAnnotationOptions`)
- **annotation\_filters** (`dtlpy.entities.filters.Filters`) – Filters entity to filter annotations for download
- **overwrite** (`bool`) – optional - default = False
- **thickness** (`int`) – optional - line thickness, if -1 annotation will be filled, default = 1
- **with\_text** (`bool`) – optional - add text to annotations, default = False
- **remote\_path** (`str`) – DEPRECATED and ignored
- **include\_annotations\_in\_output** (`bool`) – default - False , if export should contain annotations
- **export\_png\_files** (`bool`) – default - if True, semantic annotations should be exported as png files
- **filter\_output\_annotations** (`bool`) – default - False, given an export by filter - determine if to filter out annotations
- **alpha** (`float`) – opacity value [0 1], default 1
- **export\_version** (`str`) – exported items will have original extension in filename, *V1* - no original extension in filenames

#### Returns

local\_path of the directory where all the downloaded item

#### Return type

`str`

#### Example:

```
project.datasets.download_annotations(dataset='dataset_entity',
                                     local_path='local_path',
                                     annotation_options=dl.
↳ ViewAnnotationOptions,
                                     overwrite=False,
                                     thickness=1,
                                     with_text=False,
                                     alpha=1
                                     )
```

**get**(*dataset\_name: Optional[str] = None, dataset\_id: Optional[str] = None, checkout: bool = False, fetch: Optional[bool] = None*) → *Dataset*

Get dataset by name or id.

**Prerequisites:** You must be an *owner* or *developer* to use this method.

You must provide at least ONE of the following params: `dataset_id`, `dataset_name`.

**Parameters**

- **dataset\_name** (*str*) – optional - search by name
- **dataset\_id** (*str*) – optional - search by id
- **checkout** (*bool*) – True to checkout
- **fetch** (*bool*) – optional - fetch entity from platform, default taken from cookie

**Returns**

Dataset object

**Return type**

*dtlpy.entities.dataset.Dataset*

**Example:**

```
project.datasets.get(dataset_id='dataset_id')
```

**list**(*name=None, creator=None*) → List[*Dataset*]

List all datasets.

**Prerequisites:** You must be an *owner* or *developer* to use this method.

**Parameters**

- **name** (*str*) – list by name
- **creator** (*str*) – list by creator

**Returns**

List of datasets

**Return type**

*list*

**Example:**

```
project.datasets.list(name='name')
```

**merge**(*merge\_name: str, dataset\_ids: str, project\_ids: str, with\_items\_annotations: bool = True, with\_metadata: bool = True, with\_task\_annotations\_status: bool = True, wait: bool = True*)

Merge a dataset. See our [SDK docs](#) for more information.

**Prerequisites:** You must be an *owner* or *developer* to use this method.

**Parameters**

- **merge\_name** (*str*) – new dataset name
- **dataset\_ids** (*str*) – id's of the datasets you wish to merge
- **project\_ids** (*str*) – project id
- **with\_items\_annotations** (*bool*) – with items annotations
- **with\_metadata** (*bool*) – with metadata
- **with\_task\_annotations\_status** (*bool*) – with task annotations status
- **wait** (*bool*) – wait for the command to finish

**Returns**

True if success



**Return type**`bool`**Example:**

```
project.datasets.clone(dataset_ids=['dataset_id1', 'dataset_id2'],
                        merge_name='dataset_merge_name',
                        with_metadata=True,
                        with_items_annotations=False,
                        with_task_annotations_status=False)
```

**open\_in\_web**(*dataset\_name: Optional[str] = None, dataset\_id: Optional[str] = None, dataset: Optional[Dataset] = None*)

Open the dataset in web platform.

**Prerequisites:** You must be an *owner* or *developer* to use this method.

**Parameters**

- **dataset\_name** (*str*) – dataset name
- **dataset\_id** (*str*) – dataset id
- **dataset** (`dtlpy.entities.dataset.Dataset`) – dataset object

**Example:**

```
project.datasets.open_in_web(dataset_id='dataset_id')
```

**set\_readonly**(*state: bool, dataset: Dataset*)

Set dataset readonly mode.

**Prerequisites:** You must be in the role of an *owner* or *developer*.

**Parameters**

- **state** (*bool*) – state to update readonly mode
- **dataset** (`dtlpy.entities.dataset.Dataset`) – dataset object

**Example:**

```
project.datasets.set_readonly(dataset='dataset_entity', state=True)
```

**sync**(*dataset\_id: str, wait: bool = True*)

Sync dataset with external storage.

**Prerequisites:** You must be in the role of an *owner* or *developer*.

**Parameters**

- **dataset\_id** (*str*) – to sync dataset
- **wait** (*bool*) – wait for the command to finish

**Returns**

True if success

**Return type**`bool`**Example:**

```
project.datasets.sync(dataset_id='dataset_id')
```

**update**(dataset: Dataset, system\_metadata: bool = False, patch: Optional[dict] = None) → Dataset

Update dataset field.

**Prerequisites:** You must be an *owner* or *developer* to use this method.

#### Parameters

- **dataset** (dtlpy.entities.dataset.Dataset) – dataset object
- **system\_metadata** (bool) – True, if you want to change metadata system
- **patch** (dict) – Specific patch request

#### Returns

Dataset object

#### Return type

dtlpy.entities.dataset.Dataset

#### Example:

```
project.datasets.update(dataset='dataset_entity')
```

**upload\_annotations**(dataset, local\_path, filters: Optional[Filters] = None, clean=False, remote\_root\_path='/', export\_version=ExportVersion.V1)

Upload annotations to dataset.

Example for remote\_root\_path: If the item filepath is a/b/item and remote\_root\_path is /a the start folder will be b instead of a

**Prerequisites:** You must have a dataset with items that are related to the annotations. The relationship between the dataset and annotations is shown in the name. You must be in the role of an *owner* or *developer*.

#### Parameters

- **dataset** (dtlpy.entities.dataset.Dataset) – dataset to upload to
- **local\_path** (str) – str - local folder where the annotations files is
- **filters** (dtlpy.entities.filters.Filters) – Filters entity or a dictionary containing filters parameters
- **clean** (bool) – True to remove the old annotations
- **remote\_root\_path** (str) – the remote root path to match remote and local items
- **export\_version** (str) – exported items will have original extension in filename, V1 - no original extension in filenames

#### Example:

```
project.datasets.upload_annotations(dataset='dataset_entity',
                                   local_path='local_path',
                                   clean=False,
                                   export_version=dl.ExportVersion.V1
                                   )
```

### 2.3.1 Drivers

**class Drivers**(*client\_api*: *ApiClient*, *project*: *Optional*[*Project*] = *None*)

Bases: `object`

Drivers Repository

The Drivers class allows users to manage drivers that are used to connect with external storage. Read more about external storage in our [documentation](#) and [SDK documentation](#).

**create**(*name*: *str*, *driver\_type*: `ExternalStorage`, *integration\_id*: *str*, *bucket\_name*: *str*, *project\_id*: *Optional*[*str*] = *None*, *allow\_external\_delete*: *bool* = *True*, *region*: *Optional*[*str*] = *None*, *storage\_class*: *str* = "", *path*: *str* = "")

Create a storage driver.

**Prerequisites:** You must be in the role of an *owner* or *developer*.

#### Parameters

- **name** (*str*) – the driver name
- **driver\_type** (*str*) – `ExternalStorage.S3`, `ExternalStorage.GCS`, `ExternalStorage.AZUREBLOB`
- **integration\_id** (*str*) – the integration id
- **bucket\_name** (*str*) – the external bucket name
- **project\_id** (*str*) – project id
- **allow\_external\_delete** (*bool*) – true to allow deleting files from external storage when files are deleted in your Dataloop storage
- **region** (*str*) – relevant only for s3 - the bucket region
- **storage\_class** (*str*) – relevant only for s3
- **path** (*str*) – Optional. By default path is the root folder. Path is case sensitive integration

#### Returns

driver object

#### Return type

`dtlpy.entities.driver.Driver`

#### Example:

```
project.drivers.create(name='driver_name',
                       driver_type=dl.ExternalStorage.S3,
                       integration_id='integration_id',
                       bucket_name='bucket_name',
                       project_id='project_id',
                       region='eu-west-1')
```

**get**(*driver\_name*: *Optional*[*str*] = *None*, *driver\_id*: *Optional*[*str*] = *None*) → *Driver*

Get a Driver object to use in your code.

**Prerequisites:** You must be in the role of an *owner* or *developer*.

You must provide at least ONE of the following params: *driver\_name*, *driver\_id*.

#### Parameters

- **driver\_name** (*str*) – optional - search by name

- **driver\_id** (*str*) – optional - search by id

**Returns**

Driver object

**Return type**

*dtlpy.entities.driver.Driver*

**Example:**

```
project.drivers.get(driver_id='driver_id')
```

**list()** → List[*Driver*]

Get the project's drivers list.

**Prerequisites:** You must be in the role of an *owner* or *developer*.

**Returns**

List of Drivers objects

**Return type**

*list*

**Example:**

```
project.drivers.list()
```

## 2.4 Items

**class Items**(*client\_api: ApiClient, datasets: Optional[Datasets] = None, dataset: Optional[Dataset] = None, dataset\_id=None, items\_entity=None, project=None*)

Bases: *object*

Items Repository

The Items class allows you to manage items in your datasets. For information on actions related to items see [Organizing Your Dataset](#), [Item Metadata](#), and [Item Metadata-Based Filtering](#).

**clone**(*item\_id: str, dst\_dataset\_id: str, remote\_filepath: Optional[str] = None, metadata: Optional[dict] = None, with\_annotations: bool = True, with\_metadata: bool = True, with\_task\_annotations\_status: bool = False, allow\_many: bool = False, wait: bool = True*)

Clone item. Read more about cloning datasets and items in our [documentation](#) and [SDK documentation](#).

**Prerequisites:** You must be in the role of an *owner* or *developer*.

**Parameters**

- **item\_id** (*str*) – item to clone
- **dst\_dataset\_id** (*str*) – destination dataset id
- **remote\_filepath** (*str*) – complete filepath
- **metadata** (*dict*) – new metadata to add
- **with\_annotations** (*bool*) – clone annotations
- **with\_metadata** (*bool*) – clone metadata
- **with\_task\_annotations\_status** (*bool*) – clone task annotations status

- **allow\_many** (*bool*) – *bool* if True, using multiple clones in single dataset is allowed, (default=False)
- **wait** (*bool*) – wait for the command to finish

**Returns**

Item object

**Return type***dtlpy.entities.item.Item***Example:**

```
dataset.items.clone(item_id='item_id',
                    dst_dataset_id='dist_dataset_id',
                    with_metadata=True,
                    with_task_annotations_status=False,
                    with_annotations=False)
```

**delete**(*filename: Optional[str] = None, item\_id: Optional[str] = None, filters: Optional[Filters] = None*)

Delete item from platform.

**Prerequisites:** You must be in the role of an *owner* or *developer*.

You must provide at least ONE of the following params: item id, filename, filters.

**Parameters**

- **filename** (*str*) – optional - search item by remote path
- **item\_id** (*str*) – optional - search item by id
- **filters** (*dtlpy.entities.filters.Filters*) – optional - delete items by filter

**Returns**

True if success

**Return type***bool***Example:**

```
dataset.items.delete(item_id='item_id')
```

**download**(*filters: Optional[Filters] = None, items=None, local\_path: Optional[str] = None, file\_types: Optional[list] = None, save\_locally: bool = True, to\_array: bool = False, annotation\_options: Optional[ViewAnnotationOptions] = None, annotation\_filters: Optional[Filters] = None, overwrite: bool = False, to\_items\_folder: bool = True, thickness: int = 1, with\_text: bool = False, without\_relative\_path=None, avoid\_unnecessary\_annotation\_download: bool = False, include\_annotations\_in\_output: bool = True, export\_png\_files: bool = False, filter\_output\_annotations: bool = False, alpha: float = 1, export\_version=ExportVersion.V1*)

Download dataset items by filters.

Filters the dataset for items and saves them locally.

Optional – download annotation, mask, instance, and image mask of the item.

**Prerequisites:** You must be in the role of an *owner* or *developer*.**Parameters**

- **filters** (*dtlpy.entities.filters.Filters*) – Filters entity or a dictionary containing filters parameters

- **items** (*List*[`dtlpy.entities.item.Item`] or `dtlpy.entities.item.Item`) – download Item entity or item\_id (or a list of item)
- **local\_path** (*str*) – local folder or filename to save to.
- **file\_types** (*list*) – a list of file type to download. e.g ['video/webm', 'video/mp4', 'image/jpeg', 'image/png']
- **save\_locally** (*bool*) – bool. save to disk or return a buffer
- **to\_array** (*bool*) – returns Narray when True and local\_path = False
- **annotation\_options** (*list*) – download annotations options: list(`dl.ViewAnnotationOptions`)
- **annotation\_filters** (`dtlpy.entities.filters.Filters`) – Filters entity to filter annotations for download
- **overwrite** (*bool*) – optional - default = False
- **to\_items\_folder** (*bool*) – Create 'items' folder and download items to it
- **thickness** (*int*) – optional - line thickness, if -1 annotation will be filled, default = 1
- **with\_text** (*bool*) – optional - add text to annotations, default = False
- **without\_relative\_path** (*bool*) – bool - download items without the relative path from platform
- **avoid\_unnecessary\_annotation\_download** (*bool*) – default - False
- **include\_annotations\_in\_output** (*bool*) – default - False , if export should contain annotations
- **export\_png\_files** (*bool*) – default - if True, semantic annotations should be exported as png files
- **filter\_output\_annotations** (*bool*) – default - False, given an export by filter - determine if to filter out annotations
- **alpha** (*float*) – opacity value [0 1], default 1
- **export\_version** (*str*) – exported items will have original extension in filename, *V1* - no original extension in filenames

**Returns**

generator of local\_path per each downloaded item

**Return type**

generator or single item

**Example:**

```
dataset.items.download(local_path='local_path',
                       annotation_options=dl.ViewAnnotationOptions,
                       overwrite=False,
                       thickness=1,
                       with_text=False,
                       alpha=1,
                       save_locally=True
                       )
```

**get**(filepath: *Optional[str]* = None, item\_id: *Optional[str]* = None, fetch: *Optional[bool]* = None, is\_dir: *bool* = False) → *Item*

Get Item object

**Prerequisites:** You must be in the role of an *owner* or *developer*.

#### Parameters

- **filepath** (*str*) – optional - search by remote path
- **item\_id** (*str*) – optional - search by id
- **fetch** (*bool*) – optional - fetch entity from platform, default taken from cookie
- **is\_dir** (*bool*) – True if you want to get an item from dir type

#### Returns

Item object

#### Return type

*dtlpy.entities.item.Item*

#### Example:

```
dataset.items.get(item_id='item_id')
```

**get\_all\_items**(filters: *Optional[Filters]* = None) → [*<class 'dtlpy.entities.item.Item'>*]

Get all items in dataset.

**Prerequisites:** You must be in the role of an *owner* or *developer*.

#### Parameters

**filters** (*dtlpy.entities.filters.Filters*) – dl.Filters entity to filters items

#### Returns

list of all items

#### Return type

*list*

#### Example:

```
dataset.items.get_all_items()
```

**list**(filters: *Optional[Filters]* = None, page\_offset: *Optional[int]* = None, page\_size: *Optional[int]* = None) → *PagedEntities*

List items in a dataset.

**Prerequisites:** You must be in the role of an *owner* or *developer*.

#### Parameters

- **filters** (*dtlpy.entities.filters.Filters*) – Filters entity or a dictionary containing filters parameters
- **page\_offset** (*int*) – start page
- **page\_size** (*int*) – page size

#### Returns

Pages object

#### Return type

*dtlpy.entities.paged\_entities.PagedEntities*

Example:

```
dataset.items.list(page_offset=0, page_size=100)
```

**make\_dir**(*directory*, *dataset*: *Optional*[*Dataset*] = *None*) → *Item*

Create a directory in a dataset.

**Prerequisites:** All users.

**Parameters**

- **directory** (*str*) – name of directory
- **dataset** (*dtlpy.entities.dataset.Dataset*) – dataset object

**Returns**

Item object

**Return type**

*dtlpy.entities.item.Item*

Example:

```
dataset.items.make_dir(directory='directory_name')
```

**move\_items**(*destination*: *str*, *filters*: *Optional*[*Filters*] = *None*, *items*=*None*, *dataset*: *Optional*[*Dataset*] = *None*) → *bool*

Move items to another directory. If directory does not exist we will create it

**Prerequisites:** You must be in the role of an *owner* or *developer*.

**Parameters**

- **destination** (*str*) – destination directory
- **filters** (*dtlpy.entities.filters.Filters*) – optional - either this or items. Query of items to move
- **items** – optional - either this or filters. A list of items to move
- **dataset** (*dtlpy.entities.dataset.Dataset*) – dataset object

**Returns**

True if success

**Return type**

*bool*

Example:

```
dataset.items.move_items(destination='directory_name')
```

**open\_in\_web**(*filepath*=*None*, *item\_id*=*None*, *item*=*None*)

Open the item in web platform

**Prerequisites:** You must be in the role of an *owner* or *developer* or be an *annotation manager/annotator* with access to that item through task.

**Parameters**

- **filepath** (*str*) – item file path
- **item\_id** (*str*) – item id



- **item** (`dtlpy.entities.item.Item`) – item entity

**Example:**

```
dataset.items.open_in_web(item_id='item_id')
```

**set\_items\_entity**(*entity*)

Set the item entity type to `Artifact`, `Item`, or `Codebase`.

**Parameters**

**entity** (`entities.Item`, `entities.Artifact`, `entities.Codebase`) – entity type  
[`entities.Item`, `entities.Artifact`, `entities.Codebase`]

**update**(*item*: *Optional*[`Item`] = *None*, *filters*: *Optional*[`Filters`] = *None*, *update\_values*=*None*,  
*system\_update\_values*=*None*, *system\_metadata*: *bool* = *False*)

Update item metadata.

**Prerequisites:** You must be in the role of an *owner* or *developer*.

You must provide at least ONE of the following params: `update_values`, `system_update_values`.

**Parameters**

- **item** (`dtlpy.entities.item.Item`) – Item object
- **filters** (`dtlpy.entities.filters.Filters`) – optional update filtered items by given filter
- **update\_values** – optional field to be updated and new values
- **system\_update\_values** – values in system metadata to be updated
- **system\_metadata** (*bool*) – True, if you want to update the metadata system

**Returns**

Item object

**Return type**

`dtlpy.entities.item.Item`

**Example:**

```
dataset.items.update(item='item_entity')
```

**update\_status**(*status*: `ItemStatus`, *items*=*None*, *item\_ids*=*None*, *filters*=*None*, *dataset*=*None*, *clear*=*False*)

Update item status in task

**Prerequisites:** You must be in the role of an *owner* or *developer* or *annotation manager* who has been assigned a task with the item.

You must provide at least ONE of the following params: `items`, `item_ids`, `filters`.

**Parameters**

- **status** (*str*) – `ItemStatus.COMPLETED`, `ItemStatus.APPROVED`, `ItemStatus.DISCARDED`
- **items** (*list*) – list of items
- **item\_ids** (*list*) – list of items id
- **filters** (`dtlpy.entities.filters.Filters`) – Filters entity or a dictionary containing filters parameters
- **dataset** (`dtlpy.entities.dataset.Dataset`) – dataset object

- **clear** (*bool*) – to delete status

**Example:**

```
dataset.items.update_status(item_ids='item_id', status=dl.ItemStatus.COMPLETED)
```

**upload**(*local\_path*: str, *local\_annotations\_path*: ~typing.Optional[str] = None, *remote\_path*: str = '/',  
*remote\_name*: ~typing.Optional[str] = None, *file\_types*:  
~typing.Optional[~dtlpy.repositories.items.Items.list] = None, *overwrite*: bool = False,  
*item\_metadata*: ~typing.Optional[dict] = None, *output\_entity*=<class 'dtlpy.entities.item.Item'>,  
*no\_output*: bool = False, *export\_version*: str = ExportVersion.VI)

Upload local file to dataset. Local filesystem will remain unchanged. If “\*” at the end of *local\_path* (e.g. “/images/\*”) items will be uploaded without the head directory.

**Prerequisites:** Any user can upload items.

#### Parameters

- **local\_path** (*str*) – list of local file, local folder, BufferIO, numpy.ndarray or url to upload
- **local\_annotations\_path** (*str*) – path to dataloop format annotations json files.
- **remote\_path** (*str*) – remote path to save.
- **remote\_name** (*str*) – remote base name to save. when upload numpy.ndarray as local path, remote\_name with .jpg or .png ext is mandatory
- **file\_types** (*list*) – list of file type to upload. e.g [‘.jpg’, ‘.png’]. default is all
- **item\_metadata** (*dict*) – metadata dict to upload to item or ExportMetadata option to export metadata from annotation file
- **overwrite** (*bool*) – optional - default = False
- **output\_entity** – output type
- **no\_output** (*bool*) – do not return the items after upload
- **export\_version** (*str*) – exported items will have original extension in filename, VI - no original extension in filenames

#### Returns

Output (generator/single item)

#### Return type

generator or single item

**Example:**

```
dataset.items.upload(local_path='local_path',  
                    local_annotations_path='local_annotations_path',  
                    overwrite=True,  
                    item_metadata={'Hellow': 'Word'})
```

## 2.5 Annotations

**class Annotations**(*client\_api: ApiClient, item=None, dataset=None, dataset\_id=None*)

Bases: `object`

Annotations Repository

The Annotation class allows you to manage the annotations of data items. For information on annotations explore our documentation at [Classification SDK](#), [Annotation Labels and Attributes](#), [Show Video with Annotations](#).

**builder()**

Create Annotation collection.

**Prerequisites:** You must have an item to be annotated. You must have the role of an *owner* or *developer* or be assigned a task that includes that item as an *annotation manager* or *annotator*.

**Returns**

Annotation collection object

**Return type**

`dtlpy.entities.annotation_collection.AnnotationCollection`

**Example:**

```
item.annotations.builder()
```

**delete**(*annotation: Optional[Annotation] = None, annotation\_id: Optional[str] = None, filters: Optional[Filters] = None*) → `bool`

Remove an annotation from item.

**Prerequisites:** You must have an item that has been annotated. You must have the role of an *owner* or *developer* or be assigned a task that includes that item as an *annotation manager* or *annotator*.

**Parameters**

- **annotation** (`dtlpy.entities.annotation.Annotation`) – Annotation object
- **annotation\_id** (`str`) – annotation id
- **filters** (`dtlpy.entities.filters.Filters`) – Filters entity or a dictionary containing filters parameters

**Returns**

True/False

**Return type**

`bool`

**Example:**

```
item.annotations.delete(annotation_id='annotation_id')
```

**download**(*filepath: str, annotation\_format: ViewAnnotationOptions = ViewAnnotationOptions.MASK, img\_filepath: Optional[str] = None, height: Optional[float] = None, width: Optional[float] = None, thickness: int = 1, with\_text: bool = False, alpha: float = 1*)

Save annotation to file.

**Prerequisites:** You must have an item that has been annotated. You must have the role of an *owner* or *developer* or be assigned a task that includes that item as an *annotation manager* or *annotator*.

**Parameters**

- **filepath** (*str*) – Target download directory
- **annotation\_format** (*list*) – optional - list(`dl.ViewAnnotationOptions`)
- **img\_filepath** (*str*) – img file path - needed for `img_mask`
- **height** (*float*) – optional - image height
- **width** (*float*) – optional - image width
- **thickness** (*int*) – optional - annotation format, default = 1
- **with\_text** (*bool*) – optional - draw annotation with text, default = False
- **alpha** (*float*) – opacity value [0 1], default 1

**Returns**

file path to where save the annotations

**Return type**

*str*

**Example:**

```
item.annotations.download(  
    filepath='file_path',  
    annotation_format=dl.ViewAnnotationOptions.MASK,  
    img_filepath='img_filepath',  
    height=100,  
    width=100,  
    thickness=1,  
    with_text=False,  
    alpha=1)
```

**get**(*annotation\_id: str*) → *Annotation*

Get a single annotation.

**Prerequisites:** You must have an item that has been annotated. You must have the role of an *owner* or *developer* or be assigned a task that includes that item as an *annotation manager* or *annotator*.

**Parameters**

**annotation\_id** (*str*) – annotation id

**Returns**

Annotation object or None

**Return type**

*dtlpy.entities.annotation.Annotation*

**Example:**

```
item.annotations.get(annotation_id='annotation_id')
```

**list**(*filters: Optional[Filters] = None, page\_offset: Optional[int] = None, page\_size: Optional[int] = None*)

List Annotations of a specific item. You must get the item first and then list the annotations with the desired filters.

**Prerequisites:** You must have an item that has been annotated. You must have the role of an *owner* or *developer* or be assigned a task that includes that item as an *annotation manager* or *annotator*.

**Parameters**

- **filters** (`dtlpy.entities.filters.Filters`) – Filters entity or a dictionary containing filters parameters
- **page\_offset** (`int`) – starting page
- **page\_size** (`int`) – size of page

**Returns**

Pages object

**Return type**

`dtlpy.entities.paged_entities.PagedEntities`

**Example:**

```
item.annotations.list(filters=dl.Filters(
                        resource=dl.FiltersResource.ANNOTATION,
                        field='type',
                        values='box'),
                    page_size=100,
                    page_offset=0)
```

**show**(*image=None, thickness: int = 1, with\_text: bool = False, height: Optional[float] = None, width: Optional[float] = None, annotation\_format: ViewAnnotationOptions = ViewAnnotationOptions.MASK, alpha: float = 1*)

Show annotations. To use this method, you must get the item first and then show the annotations with the desired filters. The method returns an array showing all the annotations.

**Prerequisites:** You must have an item that has been annotated. You must have the role of an *owner* or *developer* or be assigned a task that includes that item as an *annotation manager* or *annotator*.

**Parameters**

- **image** (`ndarray`) – empty or image to draw on
- **thickness** (`int`) – line thickness
- **with\_text** (`bool`) – add label to annotation
- **height** (`float`) – height
- **width** (`float`) – width
- **annotation\_format** (`str`) – options: list(`dl.ViewAnnotationOptions`)
- **alpha** (`float`) – opacity value [0 1], default 1

**Returns**

`ndarray` of the annotations

**Return type**

`ndarray`

**Example:**

```
item.annotations.show(image='nd array',
                    thickness=1,
                    with_text=False,
                    height=100,
                    width=100,
                    annotation_format=dl.ViewAnnotationOptions.MASK,
                    alpha=1)
```

**update**(*annotations*, *system\_metadata=False*)

Update an existing annotation. For example, you may change the annotation's label and then use the update method.

**Prerequisites:** You must have an item that has been annotated. You must have the role of an *owner* or *developer* or be assigned a task that includes that item as an *annotation manager* or *annotator*.

**Parameters**

- **annotation** (`dtlpy.entities.annotation.Annotation`) – Annotation object
- **system\_metadata** (*bool*) – bool - True, if you want to change metadata system

**Returns**

True if successful or error if unsuccessful

**Return type**

bool

**Example:**

```
item.annotations.update(annotation='annotation')
```

**update\_status**(*annotation: Optional[Annotation] = None*, *annotation\_id: Optional[str] = None*, *status: AnnotationStatus = AnnotationStatus.ISSUE*) → *Annotation*

Set status on annotation.

**Prerequisites:** You must have an item that has been annotated. You must have the role of an *owner* or *developer* or be assigned a task that includes that item as an *annotation manager*.

**Parameters**

- **annotation** (`dtlpy.entities.annotation.Annotation`) – Annotation object
- **annotation\_id** (*str*) – optional - annotation id to set status
- **status** (*str*) – can be AnnotationStatus.ISSUE, AnnotationStatus.APPROVED, AnnotationStatus.REVIEW, AnnotationStatus.CLEAR

**Returns**

Annotation object

**Return type**

`dtlpy.entities.annotation.Annotation`

**Example:**

```
item.annotations.update_status(annotation_id='annotation_id', status=dl.  
↪ AnnotationStatus.ISSUE)
```

**upload**(*annotations*)

Upload a new annotation/annotations. You must first create the annotation using the annotation *builder* method.

**Prerequisites:** Any user can upload annotations.

**Parameters**

**annotations** (*List[dtlpy.entities.annotation.Annotation]* or *dtlpy.entities.annotation.Annotation*) – list or single annotation of type Annotation

**Returns**

list of annotation objects

**Return type**

list

**Example:**

```
item.annotations.upload(annotations='builder')
```

## 2.6 Recipes

```
class Recipes(client_api: ApiClient, dataset: Optional[Dataset] = None, project: Optional[Project] = None,
              project_id: Optional[str] = None)
```

Bases: `object`

Recipes Repository

The Recipes class allows you to manage recipes and their properties. For more information on Recipes, see our [documentation](#) and [SDK documentation](#).

```
clone(recipe: Optional[Recipe] = None, recipe_id: Optional[str] = None, shallow: bool = False)
```

Clone recipe.

**Prerequisites:** You must be in the role of an *owner* or *developer*.

**Parameters**

- **recipe** (`dtlpy.entities.recipe.Recipe`) – Recipe object
- **recipe\_id** (`str`) – Recipe id
- **shallow** (`bool`) – If True, link to existing ontology, clones all ontologies that are linked to the recipe as well

**Returns**

Cloned ontology object

**Return type**`dtlpy.entities.recipe.Recipe`**Example:**

```
dataset.recipes.clone(recipe_id='recipe_id')
```

```
create(project_ids=None, ontology_ids=None, labels=None, recipe_name=None, attributes=None) →
Recipe
```

Create a new Recipe. Note: If the param `ontology_ids` is None, an ontology will be created first.

**Prerequisites:** You must be in the role of an *owner* or *developer*.

**Parameters**

- **project\_ids** – project ids
- **ontology\_ids** – ontology ids
- **labels** – labels
- **recipe\_name** – recipe name
- **attributes** – attributes

### Returns

Recipe entity

### Return type

*dtlpy.entities.recipe.Recipe*

### Example:

```
dataset.recipes.create(recipe_name='My Recipe', labels=labels))
```

**delete**(*recipe\_id*: *str*, *force*: *bool* = *False*)

Delete recipe from platform.

**Prerequisites:** You must be in the role of an *owner* or *developer*.

### Parameters

- **recipe\_id** (*str*) – recipe id
- **force** (*bool*) – force delete recipe

### Returns

True if success

### Return type

*bool*

### Example:

```
dataset.recipes.delete(recipe_id='recipe_id')
```

**get**(*recipe\_id*: *str*) → *Recipe*

Get a Recipe object to use in your code.

**Prerequisites:** You must be in the role of an *owner* or *developer*.

### Parameters

**recipe\_id** (*str*) – recipe id

### Returns

Recipe object

### Return type

*dtlpy.entities.recipe.Recipe*

### Example:

```
dataset.recipes.get(recipe_id='recipe_id')
```

**list**(*filters*: *Optional*[*Filters*] = *None*) → *List*[*Recipe*]

List recipes for a dataset.

**Prerequisites:** You must be in the role of an *owner* or *developer*.

### Parameters

**filters** (*dtlpy.entities.filters.Filters*) – Filters entity or a dictionary containing filters parameters

### Returns

list of all recipes

### Retype

list



**Example:**

```
dataset.recipes.list()
```

**open\_in\_web**(*recipe*: *Optional*[*Recipe*] = *None*, *recipe\_id*: *Optional*[*str*] = *None*)

Open the recipe in web platform.

**Prerequisites:** All users.

**Parameters**

- **recipe** (*dtlpy.entities.recipe.Recipe*) – recipe entity
- **recipe\_id** (*str*) – recipe id

**Example:**

```
dataset.recipes.open_in_web(recipe_id='recipe_id')
```

**update**(*recipe*: *Recipe*, *system\_metadata*=*False*) → *Recipe*

Update recipe.

**Prerequisites:** You must be in the role of an *owner* or *developer*.

**Parameters**

- **recipe** (*dtlpy.entities.recipe.Recipe*) – Recipe object
- **system\_metadata** (*bool*) – True, if you want to change metadata system

**Returns**

Recipe object

**Return type**

*dtlpy.entities.recipe.Recipe*

**Example:**

```
dataset.recipes.update(recipe='recipe_entity')
```

## 2.6.1 Ontologies

**class Ontologies**(*client\_api*: *ApiClient*, *recipe*: *Optional*[*Recipe*] = *None*, *project*: *Optional*[*Project*] = *None*, *dataset*: *Optional*[*Dataset*] = *None*)

Bases: *object*

Ontologies Repository

The Ontologies class allows users to manage ontologies and their properties. Read more about ontology in our [SDK docs](#).

**create**(*labels*, *title*=*None*, *project\_ids*=*None*, *attributes*=*None*) → *Ontology*

Create a new ontology.

**Prerequisites:** You must be in the role of an *owner* or *developer*.

**Parameters**

- **labels** – recipe tags
- **title** (*str*) – ontology title, name

- **project\_ids** (*list*) – recipe project/s
- **attributes** (*list*) – recipe attributes

#### Returns

Ontology object

#### Return type

*dtlpy.entities.ontology.Ontology*

#### Example:

```
recipe.ontologies.create(labels='labels_entity',
                        title='new_ontology',
                        project_ids='project_ids')
```

### **delete**(*ontology\_id*)

Delete Ontology from the platform.

**Prerequisites:** You must be in the role of an *owner* or *developer*.

#### Parameters

**ontology\_id** – ontology id

#### Returns

True if success

#### Return type

*bool*

#### Example:

```
recipe.ontologies.delete(ontology_id='ontology_id')
```

### **delete\_attributes**(*ontology\_id*, *keys*: *list*)

Delete a bulk of attributes

#### Parameters

- **ontology\_id** (*str*) – ontology id
- **keys** (*list*) – Keys of attributes to delete

#### Returns

True if success

#### Return type

*bool*

#### Example:

```
ontology.delete_attributes(['1'])
```

### **get**(*ontology\_id*: *str*) → *Ontology*

Get Ontology object to use in your code.

**Prerequisites:** You must be in the role of an *owner* or *developer*.

#### Parameters

**ontology\_id** (*str*) – ontology id

#### Returns

Ontology object

**Return type***dtlpy.entities.ontology.Ontology***Example:**

```
recipe.ontologies.get(ontology_id='ontology_id')
```

**static labels\_to\_roots(labels)**

Converts labels dictionary to a list of platform representation of labels.

**Parameters**

**labels** (*dict*) – labels dict

**Returns**

platform representation of labels

**list**(*project\_ids=None*) → List[*Ontology*]

List ontologies for recipe

**Prerequisites:** You must be in the role of an *owner* or *developer*.

**Parameters**

**project\_ids** –

**Returns**

list of all the ontologies

**Example:**

```
recipe.ontologies.list(project_ids='project_ids')
```

**update**(*ontology: Ontology, system\_metadata=False*) → *Ontology*

Update the Ontology metadata.

**Prerequisites:** You must be in the role of an *owner* or *developer*.

**Parameters**

- **ontology** (*dtlpy.entities.ontology.Ontology*) – Ontology object
- **system\_metadata** (*bool*) – bool - True, if you want to change metadata system

**Returns**

Ontology object

**Return type***dtlpy.entities.ontology.Ontology***Example:**

```
recipe.ontologies.delete(ontology='ontology_entity')
```

**update\_attributes**(*ontology\_id: str, title: str, key: str, attribute\_type: AttributesTypes, scope: Optional[list] = None, optional: Optional[bool] = None, multi: Optional[bool] = None, values: Optional[list] = None, attribute\_range: Optional[AttributesRange] = None*)

ADD a new attribute or update if exist

**Parameters**

- **ontology\_id** (*str*) – ontology\_id

- **title** (*str*) – attribute title
- **key** (*str*) – the key of the attribute must be unique
- **attribute\_type** (*AttributesTypes*) – `dl.AttributesTypes` your attribute type
- **scope** (*list*) – list of the labels or `*` for all labels
- **optional** (*bool*) – optional attribute
- **multi** (*bool*) – if can get multiple selection
- **values** (*list*) – list of the attribute values ( for checkbox and radio button)
- **attribute\_range** (*dict* or *AttributesRange*) – `dl.AttributesRange` object

**Returns**

true in success

**Return type**

*bool*

**Example:**

```
ontology.update_attributes(key='1',
                           title='checkbox',
                           attribute_type=dl.AttributesTypes.CHECKBOX,
                           values=[1,2,3])
```

## 2.7 Tasks

**class Tasks**(*client\_api: ApiClient, project: Optional[Project] = None, dataset: Optional[Dataset] = None, project\_id: Optional[str] = None*)

Bases: *object*

Tasks Repository

The Tasks class allows the user to manage tasks and their properties. For more information, read in our SDK documentation about [Creating Tasks](#), [Redistributing and Reassigning Tasks](#), and [Task Assignment](#).

**add\_items**(*task: Optional[Task] = None, task\_id=None, filters: Optional[Filters] = None, items=None, assignee\_ids=None, query=None, workload=None, limit=None, wait=True*) → *Task*

Add items to a Task.

**Prerequisites:** You must be in the role of an *owner*, *developer*, or *annotation manager* who has been assigned to be *owner* of the annotation task.

**Parameters**

- **task** (`dtlpy.entities.task.Task`) – task entity
- **task\_id** (*str*) – task id
- **filters** (`dtlpy.entities.filters.Filters`) – Filters entity or a dictionary containing filters parameters
- **items** (*list*) – list of items to add to the task
- **assignee\_ids** (*list*) – list to assignee who works in the task
- **query** (*dict*) – query to filter the items use it

- **workload** (*list*) – list of the work load ber assignee and work load
- **limit** (*int*) – task limit
- **wait** (*bool*) – wait for the command to finish

**Returns**

task entity

**Return type***dtlpy.entities.task.Task***Example:**

```
dataset.tasks.add_items(task= 'task_entity',
                        items = [items])
```

**create**(*task\_name*, *due\_date*=None, *assignee\_ids*=None, *workload*=None, *dataset*=None, *task\_owner*=None, *task\_type*='annotation', *task\_parent\_id*=None, *project\_id*=None, *recipe\_id*=None, *assignments\_ids*=None, *metadata*=None, *filters*=None, *items*=None, *query*=None, *available\_actions*=None, *wait*=True, *check\_if\_exist*: *Filters* = False, *limit*=None, *batch\_size*=None, *max\_batch\_workload*=None, *allowed\_assignees*=None) → *Task*

Create a new Annotation Task.

**Prerequisites:** You must be in the role of an *owner*, *developer*, or *annotation manager* who has been assigned to be *owner* of the annotation task.

**Parameters**

- **task\_name** (*str*) – task name
- **due\_date** (*float*) – date by which the task should be finished; for example, *due\_date* = *datetime.datetime(day= 1, month= 1, year= 2029).timestamp()*
- **assignee\_ids** (*list*) – list of assignee
- **workload** (*List[WorkloadUnit]*) – list WorkloadUnit for the task assignee
- **dataset** (*entities.Dataset*) – dataset entity
- **task\_owner** (*str*) – task owner
- **task\_type** (*str*) – “annotation” or “qa”
- **task\_parent\_id** (*str*) – optional if type is qa - parent task id
- **project\_id** (*str*) – project id
- **recipe\_id** (*str*) – recipe id
- **assignments\_ids** (*list*) – assignments ids
- **metadata** (*dict*) – metadata for the task
- **filters** (*entities.Filters*) – filter to the task
- **items** (*List[entities.Item]*) – item to insert to the task
- **query** (*entities.Filters*) – filter to the task
- **available\_actions** (*list*) – list of available actions to the task
- **wait** (*bool*) – wait for the command to finish
- **check\_if\_exist** (*entities.Filters*) – dl.Filters check if task exist according to filter
- **limit** (*int*) – task limit

- **batch\_size** (*int*) – Pulling batch size (items) . Restrictions - Min 3, max 100
- **max\_batch\_workload** (*int*) – Max items in assignment . Restrictions - Min batchSize + 2 , max batchSize \* 2
- **allowed\_assignees** (*list*) – It's like the workload, but without percentage.

**Returns**

Annotation Task object

**Return type**

*dtlpy.entities.task.Task*

**Example:**

```
dataset.tasks.create(task= 'task_entity',
                      due_date = datetime.datetime(day= 1, month= 1, year= 2029).
↳ timestamp(),
                      assignee_ids =[ 'annotator1@dataloop.ai',
↳ 'annotator2@dataloop.ai'])
```

**create\_qa\_task**(*task: Task, assignee\_ids, due\_date=None, filters=None, items=None, query=None, workload=None, metadata=None, available\_actions=None, wait=True, batch\_size=None, max\_batch\_workload=None, allowed\_assignees=None*) → *Task*

Create a new QA Task.

**Prerequisites:** You must be in the role of an *owner*, *developer*, or *annotation manager* who has been assigned to be *owner* of the annotation task.

**Parameters**

- **task** (*dtlpy.entities.task.Task*) – parent task
- **assignee\_ids** (*list*) – list of assignee
- **due\_date** (*float*) – date by which the task should be finished; for example, `due_date = datetime.datetime(day= 1, month= 1, year= 2029).timestamp()`
- **filters** (*entities.Filters*) – filter to the task
- **items** (*List[entities.Item]*) – item to insert to the task
- **query** (*entities.Filters*) – filter to the task
- **workload** (*List[WorkloadUnit]*) – list WorkloadUnit for the task assignee
- **metadata** (*dict*) – metadata for the task
- **available\_actions** (*list*) – list of available actions to the task
- **wait** (*bool*) – wait for the command to finish
- **batch\_size** (*int*) – Pulling batch size (items) . Restrictions - Min 3, max 100
- **max\_batch\_workload** (*int*) – Max items in assignment . Restrictions - Min batchSize + 2 , max batchSize \* 2
- **allowed\_assignees** (*list*) – It's like the workload, but without percentage.

**Returns**

task object

**Return type**

*dtlpy.entities.task.Task*

**Example:**

```
dataset.tasks.create_qa_task(task= 'task_entity',
                             due_date = datetime.datetime(day= 1, month= 1,
↪year= 2029).timestamp(),
                             assignee_ids =[ 'annotator1@dataloop.ai',
↪'annotator2@dataloop.ai'])
```

**delete**(task: *Optional[Task]* = None, task\_name: *Optional[str]* = None, task\_id: *Optional[str]* = None, wait: *bool* = True)

Delete an Annotation Task.

**Prerequisites:** You must be in the role of an *owner* or *developer* or *annotation manager* who created that task.

**Parameters**

- **task** (*dtlpy.entities.task.Task*) – task entity
- **task\_name** (*str*) – task name
- **task\_id** (*str*) – task id
- **wait** (*bool*) – wait for the command to finish

**Returns**

True is success

**Return type**

*bool*

**Example:**

```
dataset.tasks.delete(task_id='task_id')
```

**get**(task\_name=None, task\_id=None) → *Task*

Get an Annotation Task object to use in your code.

**Prerequisites:** You must be in the role of an *owner* or *developer* or *annotation manager* who has been assigned the task.

**Parameters**

- **task\_name** (*str*) – optional - search by name
- **task\_id** (*str*) – optional - search by id

**Returns**

task object

**Return type**

*dtlpy.entities.task.Task*

**Example:**

```
dataset.tasks.get(task_id='task_id')
```

**get\_items**(task\_id: *Optional[str]* = None, task\_name: *Optional[str]* = None, dataset: *Optional[Dataset]* = None, filters: *Optional[Filters]* = None) → *PagedEntities*

Get the task items to use in your code.

**Prerequisites:** You must be in the role of an *owner*, *developer*, or *annotation manager* who has been assigned to be *owner* of the annotation task.

If a filters param is provided, you will receive a PagedEntity output of the task items. If no filter is provided, you will receive a list of the items.

#### Parameters

- **task\_id** (*str*) – task id
- **task\_name** (*str*) – task name
- **dataset** (`dtlpy.entities.dataset.Dataset`) – dataset entity
- **filters** (`dtlpy.entities.filters.Filters`) – Filters entity or a dictionary containing filters parameters

#### Returns

list of the items or PagedEntity output of items

#### Return type

list or `dtlpy.entities.paged_entities.PagedEntities`

#### Example:

```
dataset.tasks.get_items(task_id= 'task_id')
```

**list**(*project\_ids=None, status=None, task\_name=None, pages\_size=None, page\_offset=None, recipe=None, creator=None, assignments=None, min\_date=None, max\_date=None, filters: Optional[Filters] = None*) → Union[List[Task], PagedEntities]

List all Annotation Tasks.

**Prerequisites:** You must be in the role of an *owner* or *developer* or *annotation manager* who has been assigned the task.

#### Parameters

- **project\_ids** – list of project ids
- **status** (*str*) – status
- **task\_name** (*str*) – task name
- **pages\_size** (*int*) – pages size
- **page\_offset** (*int*) – page offset
- **recipe** (`dtlpy.entities.recipe.Recipe`) – recipe entity
- **creator** (*str*) – creator
- **assignments** (`dtlpy.entities.assignment.Assignment` recipe) – assignments entity
- **min\_date** (*double*) – double min date
- **max\_date** (*double*) – double max date
- **filters** (`dtlpy.entities.filters.Filters`) – dl.Filters entity to filters items

#### Returns

List of Annotation Task objects

#### Example:



```
dataset.tasks.list(project_ids='project_ids', pages_size=100, page_offset=0)
```

**open\_in\_web**(*task\_name*: *Optional*[*str*] = *None*, *task\_id*: *Optional*[*str*] = *None*, *task*: *Optional*[*Task*] = *None*)

Open the task in the web platform.

**Prerequisites:** You must be in the role of an *owner* or *developer* or *annotation manager* who has been assigned the task.

#### Parameters

- **task\_name** (*str*) – task name
- **task\_id** (*str*) – task id
- **task** (*dtlpy.entities.task.Task*) – task entity

**Example:**

```
dataset.tasks.open_in_web(task_id='task_id')
```

**query**(*filters*=*None*, *project\_ids*=*None*)

List all tasks by filter.

**Prerequisites:** You must be in the role of an *owner* or *developer* or *annotation manager* who has been assigned the task.

#### Parameters

- **filters** (*dtlpy.entities.filters.Filters*) – Filters entity or a dictionary containing filters parameters
- **project\_ids** (*list*) – list of project ids

#### Returns

Paged entity

#### Return type

*dtlpy.entities.paged\_entities.PagedEntities*

**Example:**

```
dataset.tasks.query(project_ids='project_ids')
```

**remove\_items**(*task*: *Optional*[*Task*] = *None*, *task\_id*=*None*, *filters*: *Optional*[*Filters*] = *None*, *query*=*None*, *items*=*None*, *wait*=*True*)

remove items from Task.

**Prerequisites:** You must be in the role of an *owner*, *developer*, or *annotation manager* who has been assigned to be *owner* of the annotation task.

#### Parameters

- **task** (*dtlpy.entities.task.Task*) – task entity
- **task\_id** (*str*) – task id
- **filters** (*dtlpy.entities.filters.Filters*) – Filters entity or a dictionary containing filters parameters
- **query** (*dict*) – query to filter the items use it
- **items** (*list*) – list of items to add to the task

- **wait** (*bool*) – wait for the command to finish

**Returns**

True if success and an error if failed

**Return type**

*bool*

**Examples:**

```
dataset.tasks.remove_items(task= 'task_entity',
                           items = [items])
```

**set\_status**(*status: str, operation: str, task\_id: str, item\_ids: List[str]*)

Update an item status within a task.

**Prerequisites:** You must be in the role of an *owner*, *developer*, or *annotation manager* who has been assigned to be *owner* of the annotation task.

**Parameters**

- **status** (*str*) – string the describes the status
- **operation** (*str*) – ‘create’ or ‘delete’
- **task\_id** (*str*) – task id
- **item\_ids** (*list*) – List[str] id items ids

**Returns**

True if success

**Return type**

*bool*

**Example:**

```
dataset.tasks.set_status(task_id= 'task_id', status='complete', operation=
↪ 'create')
```

**update**(*task: Optional[Task] = None, system\_metadata=False*) → *Task*

Update an Annotation Task.

**Prerequisites:** You must be in the role of an *owner* or *developer* or *annotation manager* who created that task.

**Parameters**

- **task** (*dtlpy.entities.task.Task*) – task entity
- **system\_metadata** (*bool*) – True, if you want to change metadata system

**Returns**

Annotation Task object

**Return type**

*dtlpy.entities.task.Task*

**Example:**

```
dataset.tasks.update(task='task_entity')
```

## 2.7.1 Assignments

**class** `Assignments`(*client\_api*: `ApiClient`, *project*: `Optional[Project]` = `None`, *task*: `Optional[Task]` = `None`, *dataset*: `Optional[Dataset]` = `None`, *project\_id*=`None`)

Bases: `object`

Assignments Repository

The Assignments class allows users to manage assignments and their properties. Read more about [Task Assignment](#) in our SDK documentation.

**create**(*assignee\_id*: `str`, *task*: `Optional[Task]` = `None`, *filters*: `Optional[Filters]` = `None`, *items*: `Optional[list]` = `None`) → `Assignment`

Create a new assignment.

**Prerequisites:** You must be in the role of an *owner*, *developer*, or *annotation manager* who has been assigned as *owner* of the annotation task.

### Parameters

- **assignee\_id** (`str`) – the assignee for the assignment
- **task** (`dtlpy.entities.task.Task`) – task entity
- **filters** (`dtlpy.entities.filters.Filters`) – Filters entity or a dictionary containing filters parameters
- **items** (`list`) – list of items

### Returns

Assignment object

### Return type

`dtlpy.entities.assignment.Assignment` assignment

### Example:

```
task.assignments.create(assignee_id='annotator1@dataloop.ai')
```

**get**(*assignment\_name*: `Optional[str]` = `None`, *assignment\_id*: `Optional[str]` = `None`)

Get Assignment object to use it in your code.

### Parameters

- **assignment\_name** (`str`) – optional - search by name
- **assignment\_id** (`str`) – optional - search by id

### Returns

Assignment object

### Return type

`dtlpy.entities.assignment.Assignment`

### Example:

```
task.assignments.get(assignment_id='assignment_id')
```

**get\_items**(*assignment*: `Optional[Assignment]` = `None`, *assignment\_id*=`None`, *assignment\_name*=`None`, *dataset*=`None`, *filters*=`None`) → `PagedEntities`

Get all the items in the assignment.

**Prerequisites:** You must be in the role of an *owner*, *developer*, or *annotation manager* who has been assigned as *owner* of the annotation task.

#### Parameters

- **assignment** (`dtlpy.entities.assignment.Assignment`) – assignment entity
- **assignment\_id** (`str`) – assignment id
- **assignment\_name** (`str`) – assignment name
- **dataset** (`dtlpy.entities.dataset.Dataset`) – dataset entity
- **filters** (`dtlpy.entities.filters.Filters`) – Filters entity or a dictionary containing filters parameters

#### Returns

pages of the items

#### Return type

`dtlpy.entities.paged_entities.PagedEntities`

#### Example:

```
task.assignments.get_items(assignment_id='assignment_id')
```

```
list(project_ids: Optional[list] = None, status: Optional[str] = None, assignment_name: Optional[str] = None, assignee_id: Optional[str] = None, pages_size: Optional[int] = None, page_offset: Optional[int] = None, task_id: Optional[int] = None) → List[Assignment]
```

Get Assignment list to be able to use it in your code.

**Prerequisites:** You must be in the role of an *owner*, *developer*, or *annotation manager* who has been assigned as *owner* of the annotation task.

#### Parameters

- **project\_ids** (`list`) – list of project ids
- **status** (`str`) – assignment status
- **assignment\_name** (`str`) – assignment name
- **assignee\_id** (`str`) – the user that assignee the assignment to it
- **pages\_size** (`int`) – pages size
- **page\_offset** (`int`) – page offset
- **task\_id** (`str`) – task id

#### Returns

List of Assignment objects

#### Return type

`miscellaneous.List[dtlpy.entities.assignment.Assignment]`

#### Example:

```
task.assignments.list(status='complete', assignee_id='user@dataloop.ai', pages_size=100, page_offset=0)
```

```
open_in_web(assignment_name: Optional[str] = None, assignment_id: Optional[str] = None, assignment: Optional[str] = None)
```

Open the assignment in the platform.

**Prerequisites:** All users.

#### Parameters

- **assignment\_name** (*str*) – assignment name
- **assignment\_id** (*str*) – assignment id
- **assignment** (`dtlpy.entities.assignment.Assignment`) – assignment object

**Example:**

```
task.assignments.open_in_web(assignment_id='assignment_id')
```

**reassign**(*assignee\_id: str*, *assignment: Optional[Assignment] = None*, *assignment\_id: Optional[str] = None*, *task: Optional[Task] = None*, *task\_id: Optional[str] = None*, *wait: bool = True*)

Reassign an assignment.

**Prerequisites:** You must be in the role of an *owner*, *developer*, or *annotation manager* who has been assigned as *owner* of the annotation task.

#### Parameters

- **assignee\_id** (*str*) – the id of the user whom you want to assign the assignment to
- **assignment** (`dtlpy.entities.assignment.Assignment`) – assignment object
- **assignment\_id** – assignment id
- **task** (`dtlpy.entities.task.Task`) – task object
- **task\_id** (*str*) – task id
- **wait** (*bool*) – wait for the command to finish

#### Returns

Assignment object

#### Return type

`dtlpy.entities.assignment.Assignment`

**Example:**

```
task.assignments.reassign(assignee_ids='annotator1@dataloop.ai')
```

**redistribute**(*workload: Workload*, *assignment: Optional[Assignment] = None*, *assignment\_id: Optional[str] = None*, *task: Optional[Task] = None*, *task\_id: Optional[str] = None*, *wait: bool = True*)

Redistribute an assignment.

**Prerequisites:** You must be in the role of an *owner*, *developer*, or *annotation manager* who has been assigned as *owner* of the annotation task.

**Example:**

#### Parameters

- **workload** (`dtlpy.entities.assignment.Workload`) – workload object that contain the assignees and the work load
- **assignment** (`dtlpy.entities.assignment.Assignment`) – assignment object

- **assignment\_id** (*str*) – assignment id
- **task** (`dtlpy.entities.task.Task`) – task object
- **task\_id** (*str*) – task id
- **wait** (*bool*) – wait for the command to finish

**Returns**

Assignment object

**Return type**

`dtlpy.entities.assignment.Assignment` assignment

```
task.assignments.redistribute(workload=dl.Workload([dl.WorkloadUnit(assignee_id=
↪ "annotator1@dataloop.ai", load=50),
                                                                    dl.WorkloadUnit(assignee_id=
↪ "annotator2@dataloop.ai", load=50)]))
```

**set\_status**(*status: str, operation: str, item\_id: str, assignment\_id: str*) → *bool*

Set item status within assignment.

**Prerequisites:** You must be in the role of an *owner*, *developer*, or *annotation manager* who has been assigned as *owner* of the annotation task.

**Parameters**

- **status** (*str*) – status
- **operation** (*str*) – created/deleted
- **item\_id** (*str*) – item id
- **assignment\_id** (*str*) – assignment id

**Returns**

True id success

**Return type**

*bool*

**Example:**

```
task.assignments.set_status(assignment_id='assignment_id',
                           status='complete',
                           operation='created',
                           item_id='item_id')
```

**update**(*assignment: Optional[Assignment] = None, system\_metadata: bool = False*) → *Assignment*

Update an assignment.

**Prerequisites:** You must be in the role of an *owner*, *developer*, or *annotation manager* who has been assigned as *owner* of the annotation task.

**Parameters**

- **assignment** (`dtlpy.entities.assignment.Assignment` *assignment*) – assignment entity
- **system\_metadata** (*bool*) – True, if you want to change metadata system

**Returns**

Assignment object

**Return type**

dtlpy.entities.assignment.Assignment assignment

**Example:**

```
task.assignments.update(assignment='assignment_entity', system_metadata=False)
```

## 2.8 Packages

```
class LocalServiceRunner(client_api: ApiClient, packages, cwd=None, multithreading=False,
                          concurrency=10, package: Optional[Package] = None,
                          module_name='default_module', function_name='run',
                          class_name='ServiceRunner', entry_point='main.py', mock_file_path=None)
```

Bases: `object`

Service Runner Class

```
get_field(field_name, field_type, mock_json, project=None, mock_inputs=None)
```

Get field in mock json.

**Parameters**

- **field\_name** – field name
- **field\_type** – field type
- **mock\_json** – mock json
- **project** – project
- **mock\_inputs** – mock inputs

**Returns**

```
get_mainpy_run_service()
```

Get mainpy run service

**Returns**

```
run_local_project(project=None)
```

Run local project

**Parameters**

- **project** – project entity

```
class Packages(client_api: ApiClient, project: Optional[Project] = None)
```

Bases: `object`

Packages Repository

The Packages class allows users to manage packages (code used for running in Dataloop's FaaS) and their properties. Read more about [Packages](#).

```
build_requirements(filepath) → list
```

Build a requirement list (list of packages your code requires to run) from a file path. **The file listing the requirements MUST BE a txt file.**

**Prerequisites:** You must be in the role of an *owner* or *developer*.

**Parameters**

**filepath** – path of the requirements file

**Returns**

a list of `dl.PackageRequirement`

**Return type**

`list`

```
static build_trigger_dict(actions, name='default_module', filters=None, function='run',
                          execution_mode: TriggerExecutionMode = 'Once', type_t: TriggerType =
                          'Event')
```

Build a trigger dictionary to trigger FaaS. Read more about [FaaS Triggers](#).

**Prerequisites:** You must be in the role of an *owner* or *developer*.

**Parameters**

- **actions** – list of `dl.TriggerAction`
- **name** (`str`) – trigger name
- **filters** (`dtlpy.entities.filters.Filters`) – Filters entity or a dictionary containing filters parameters
- **function** (`str`) – function name
- **execution\_mode** (`str`) – execution mode `dl.TriggerExecutionMode`
- **type\_t** (`str`) – trigger type `dl.TriggerType`

**Returns**

trigger dict

**Return type**

`dict`

**Example:**

```
project.packages.build_trigger_dict(actions=dl.TriggerAction.CREATED,
                                    function='run',
                                    execution_mode=dl.TriggerExecutionMode.ONCE)
```

```
static check_cls_arguments(cls, missing, function_name, function_inputs)
```

Check class arguments. This method checks that the package function is correct.

**Prerequisites:** You must be in the role of an *owner* or *developer*.

**Parameters**

- **cls** – packages class
- **missing** (`list`) – list of the missing params
- **function\_name** (`str`) – name of function
- **function\_inputs** (`list`) – list of function inputs

```
checkout(package: Optional[Package] = None, package_id: Optional[str] = None, package_name:
Optional[str] = None)
```

Checkout (switch) to a package.

**Prerequisites:** You must be in the role of an *owner* or *developer*.

**Parameters**



- **package** (`dtlpy.entities.package.Package`) – package entity
- **package\_id** (`str`) – package id
- **package\_name** (`str`) – package name

**Example:**

```
project.packages.checkout(package='package_entity')
```

**delete**(*package*: `Optional[Package]` = `None`, *package\_name*=`None`, *package\_id*=`None`)

Delete a Package object.

**Prerequisites:** You must be in the role of an *owner* or *developer*.

#### Parameters

- **package** (`dtlpy.entities.package.Package`) – package entity
- **package\_id** (`str`) – package id
- **package\_name** (`str`) – package name

#### Returns

True if success

#### Return type

`bool`

**Example:**

```
project.packages.delete(package_name='package_name')
```

**deploy**(*package\_id*: `Optional[str]` = `None`, *package\_name*: `Optional[str]` = `None`, *package*: `Optional[Package]` = `None`, *service\_name*: `Optional[str]` = `None`, *project\_id*: `Optional[str]` = `None`, *revision*: `Optional[str]` = `None`, *init\_input*: `Optional[Union[List[FunctionIO], FunctionIO, dict]]` = `None`, *runtime*: `Optional[Union[KubernetesRuntime, dict]]` = `None`, *sdk\_version*: `Optional[str]` = `None`, *agent\_versions*: `Optional[dict]` = `None`, *bot*: `Optional[Union[Bot, str]]` = `None`, *pod\_type*: `Optional[InstanceCatalog]` = `None`, *verify*: `bool` = `True`, *checkout*: `bool` = `False`, *module\_name*: `Optional[str]` = `None`, *run\_execution\_as\_process*: `Optional[bool]` = `None`, *execution\_timeout*: `Optional[int]` = `None`, *drain\_time*: `Optional[int]` = `None`, *on\_reset*: `Optional[str]` = `None`, *max\_attempts*: `Optional[int]` = `None`, *force*: `bool` = `False`, *secrets*: `Optional[list]` = `None`, *\*\*kwargs*)  
→ *Service*

Deploy a package. A service is required to run the code in your package.

**Prerequisites:** You must be in the role of an *owner* or *developer*.

#### Parameters

- **package\_id** (`str`) – package id
- **package\_name** (`str`) – package name
- **package** (`dtlpy.entities.package.Package`) – package entity
- **service\_name** (`str`) – service name
- **project\_id** (`str`) – project id
- **revision** (`str`) – package revision - default=latest
- **init\_input** – config to run at startup
- **runtime** (`dict`) – runtime resources

- **sdk\_version** (*str*) –
  - optional - string - sdk version
- **agent\_versions** (*dict*) –
  - dictionary - - optional - versions of sdk, agent runner and agent proxy
- **bot** (*str*) – bot email
- **pod\_type** (*str*) – pod type `dl.InstanceCatalog`
- **verify** (*bool*) – verify the inputs
- **checkout** (*bool*) – checkout
- **module\_name** (*str*) – module name
- **run\_execution\_as\_process** (*bool*) – run execution as process
- **execution\_timeout** (*int*) – execution timeout
- **drain\_time** (*int*) – drain time
- **on\_reset** (*str*) – on reset
- **max\_attempts** (*int*) – Maximum execution retries in-case of a service reset
- **force** (*bool*) – optional - terminate old replicas immediately
- **secrets** (*list*) – list of the integrations ids

**Returns**

Service object

**Return type**

*dtlpy.entities.service.Service*

**Example:**

```
project.packages.deploy(service_name=package_name,
                        execution_timeout=3 * 60 * 60,
                        module_name=module.name,
                        runtime=dl.KubernetesRuntime(
                            concurrency=10,
                            pod_type=dl.InstanceCatalog.REGULAR_S,
                            autoscaler=dl.KubernetesRabbitmqAutoscaler(
                                min_replicas=1,
                                max_replicas=20,
                                queue_length=20
                            )
                        )
                    )
```

**deploy\_from\_file**(*project*, *json\_filepath*)

Deploy package and service from a JSON file.

**Prerequisites:** You must be in the role of an *owner* or *developer*.

**Parameters**

- **project** (*dtlpy.entities.project.Project*) – project entity
- **json\_filepath** (*str*) – path of the file to deploy

**Returns**

the package and the services

**Example:**

```
project.packages.deploy_from_file(project='project_entity', json_filepath='json_
↪filepath')
```

**static generate**(name=None, src\_path: *Optional[str]* = None, service\_name: *Optional[str]* = None, package\_type='default\_package\_type')

Generate a new package. Provide a file path to a JSON file with all the details of the package and service to generate the package.

**Prerequisites:** You must be in the role of an *owner* or *developer*.

**Parameters**

- **name** (*str*) – name
- **src\_path** (*str*) – source file path
- **service\_name** (*str*) – service name
- **package\_type** (*str*) – package type from PackageCatalog

**Example:**

```
project.packages.generate(name='package_name',
                           src_path='src_path')
```

**get**(package\_name: *Optional[str]* = None, package\_id: *Optional[str]* = None, checkout: *bool* = False, fetch=None) → *Package*

Get Package object to use in your code.

**Prerequisites:** You must be in the role of an *owner* or *developer*.

**Parameters**

- **package\_id** (*str*) – package id
- **package\_name** (*str*) – package name
- **checkout** (*bool*) – checkout
- **fetch** – optional - fetch entity from platform, default taken from cookie

**Returns**

Package object

**Return type**

*dtlpy.entities.package.Package*

**Example:**

```
project.packages.get(package_id='package_id')
```

**list**(filters: *Optional[Filters]* = None, project\_id: *Optional[str]* = None) → *PagedEntities*

List project packages.

**Prerequisites:** You must be in the role of an *owner* or *developer*.

**Parameters**

- **filters** (`dtlpy.entities.filters.Filters`) – Filters entity or a dictionary containing filters parameters
- **project\_id** (`str`) – project id

#### Returns

Paged entity

#### Return type

`dtlpy.entities.paged_entities.PagedEntities`

#### Example:

```
project.packages.list()
```

**open\_in\_web**(*package*: `Optional[Package]` = `None`, *package\_id*: `Optional[str]` = `None`, *package\_name*: `Optional[str]` = `None`)

Open the package in the web platform.

**Prerequisites:** You must be in the role of an *owner* or *developer*.

#### Parameters

- **package** (`dtlpy.entities.package.Package`) – package entity
- **package\_id** (`str`) – package id
- **package\_name** (`str`) – package name

#### Example:

```
project.packages.open_in_web(package_id='package_id')
```

**pull**(*package*: `Package`, *version*=`None`, *local\_path*=`None`, *project\_id*=`None`)

Pull (download) the package to a local path.

**Prerequisites:** You must be in the role of an *owner* or *developer*.

#### Parameters

- **package** (`dtlpy.entities.package.Package`) – package entity
- **version** –
- **local\_path** –
- **project\_id** –

#### Returns

local path where the package pull

#### Return type

`str`

#### Example:

```
project.packages.pull(package='package_entity', local_path='local_path')
```

```
push(project: Optional[Project] = None, project_id: Optional[str] = None, package_name: Optional[str] = None, src_path: Optional[str] = None, codebase: Optional[Union[GitCodebase, ItemCodebase, FilesystemCodebase]] = None, modules: Optional[List[PackageModule]] = None, is_global: Optional[bool] = None, checkout: bool = False, revision_increment: Optional[str] = None, version: Optional[str] = None, ignore_sanity_check: bool = False, service_update: bool = False, service_config: Optional[dict] = None, slots: Optional[List[PackageSlot]] = None, requirements: Optional[List[PackageRequirement]] = None) → Package
```

Push your local package to the UI.

**Prerequisites:** You must be in the role of an *owner* or *developer*.

Project will be taken in the following hierarchy: project(input) -> project\_id(input) -> self.project(context) -> checked out

#### Parameters

- **project** (`dtlpy.entities.project.Project`) – optional - project entity to deploy to. default from context or checked-out
- **project\_id** (`str`) – optional - project id to deploy to. default from context or checked-out
- **package\_name** (`str`) – package name
- **src\_path** (`str`) – path to package codebase
- **codebase** (`dtlpy.entities.codebase.Codebase`) – codebase object
- **modules** (`list`) – list of modules PackageModules of the package
- **is\_global** (`bool`) – is package is global or local
- **checkout** (`bool`) – checkout package to local dir
- **revision\_increment** (`str`) – optional - str - version bumping method - major/minor/patch - default = None
- **version** (`str`) – semver version f the package
- **ignore\_sanity\_check** (`bool`) – NOT RECOMMENDED - skip code sanity check before pushing
- **service\_update** (`bool`) – optional - bool - update the service
- **service\_config** (`dict`) – json of service - a service that have config from the main service if wanted
- **slots** (`list`) – optional - list of slots PackageSlot of the package
- **requirements** (`list`) – requirements - list of package requirements

#### Returns

Package object

#### Return type

`dtlpy.entities.package.Package`

#### Example:

```
project.packages.push(package_name='package_name',
                      modules=[module],
                      version='1.0.0',
                      src_path=os.getcwd()
                      )
```

**revisions**(*package*: *Optional*[*Package*] = *None*, *package\_id*: *Optional*[*str*] = *None*)

Get the package revisions history.

**Prerequisites:** You must be in the role of an *owner* or *developer*.

**Parameters**

- **package** (*dtlpy.entities.package.Package*) – package entity
- **package\_id** (*str*) – package id

**Example:**

```
project.packages.revisions(package='package_entity')
```

**test\_local\_package**(*cwd*: *Optional*[*str*] = *None*, *concurrency*: *Optional*[*int*] = *None*, *package*: *Optional*[*Package*] = *None*, *module\_name*: *str* = 'default\_module', *function\_name*: *str* = 'run', *class\_name*: *str* = 'ServiceRunner', *entry\_point*: *str* = 'main.py', *mock\_file\_path*: *Optional*[*str*] = *None*)

Test local package in local environment.

**Prerequisites:** You must be in the role of an *owner* or *developer*.

**Parameters**

- **cwd** (*str*) – path to the file
- **concurrency** (*int*) – the concurrency of the test
- **package** (*dtlpy.entities.package.Package*) – entities.package
- **module\_name** (*str*) – module name
- **function\_name** (*str*) – function name
- **class\_name** (*str*) – class name
- **entry\_point** (*str*) – the file to run like main.py
- **mock\_file\_path** (*str*) – the mock file that have the inputs

**Returns**

list created by the function that tested the output

**Return type**

list

**Example:**

```
project.packages.test_local_package(cwd='path_to_package',
                                     package='package_entity',
                                     function_name='run')
```

**update**(*package*: *Package*, *revision\_increment*: *Optional*[*str*] = *None*) → *Package*

Update Package changes to the platform.

**Prerequisites:** You must be in the role of an *owner* or *developer*.

**Parameters**

- **package** (*dtlpy.entities.package.Package*) –
- **revision\_increment** – optional - str - version bumping method - major/minor/patch - default = None

**Returns**

Package object

**Return type***dtlpy.entities.package.Package***Example:**

```
project.packages.delete(package='package_entity')
```

## 2.8.1 Codebases

**class** `Codebases`(*client\_api*: *ApiClient*, *project*: *Optional*[*Project*] = *None*, *dataset*: *Optional*[*Dataset*] = *None*, *project\_id*: *Optional*[*str*] = *None*)

Bases: `object`

Codebase Repository

The Codebases class allows the user to manage codebases and their properties. The codebase is the code the user uploads for the user's packages to run. Read more about [codebase](#) in our FaaS (function as a service).

**clone\_git**(*codebase*: *Codebase*, *local\_path*: *str*)

Clone code base

**Prerequisites:** You must be in the role of an *owner* or *developer*. You must have a package.

**Parameters**

- **codebase** (*dtlpy.entities.codebase.Codebase*) – codebase object
- **local\_path** (*str*) – local path

**Returns**

path where the clone will be

**Return type***str***Example:**

```
package.codebases.clone_git(codebase='codebase_entity', local_path='local_path')
```

**get**(*codebase\_name*: *Optional*[*str*] = *None*, *codebase\_id*: *Optional*[*str*] = *None*, *version*: *Optional*[*str*] = *None*)

Get a Codebase object to use in your code.

**Prerequisites:** You must be in the role of an *owner* or *developer*. You must have a package.

**Example:**

```
package.codebases.get(codebase_name='codebase_name')
```

**Parameters**

- **codebase\_name** (*str*) – optional - search by name
- **codebase\_id** (*str*) – optional - search by id
- **version** (*str*) – codebase version. default is latest. options: “all”, “latest” or ver number - “10”

**Returns**

Codebase object

**static** `get_current_version`(*all\_versions\_pages*, *zip\_md*)

This method returns the current version of the codebase and other versions found.

**Prerequisites:** You must be in the role of an *owner* or *developer*. You must have a package.**Parameters**

- **all\_versions\_pages** (*codebase*) – codebase object
- **zip\_md** – zipped file of codebase

**Returns**

current version and all versions found of codebase

**Return type**`int, int`**Example:**

```
package.codebases.get_current_version(all_versions_pages='codebase_entity', zip_md='path')
```

**list()** → *PagedEntities*

List all codebases.

**Prerequisites:** You must be in the role of an *owner* or *developer*. You must have a package.**Example:**

```
package.codebases.list()
```

**Returns**

Paged entity

**Return type***dtlpy.entities.paged\_entities.PagedEntities***list\_versions**(*codebase\_name*: *str*)

List all codebase versions.

**Prerequisites:** You must be in the role of an *owner* or *developer*. You must have a package.**Example:**

```
package.codebases.list_versions(codebase_name='codebase_name')
```

**Parameters****codebase\_name** (*str*) – code base name**Returns**

list of versions

**Return type**`list`



**pack**(*directory*: *str*, *name*: *Optional[str]* = None, *description*: *str* = "")

Zip a local code directory and post to codebases.

**Prerequisites:** You must be in the role of an *owner* or *developer*. You must have a package.

**Parameters**

- **directory** (*str*) – local directory to pack
- **name** (*str*) – codebase name
- **description** (*dtr*) – codebase description

**Returns**

Codebase object

**Return type**

dtlpy.entities.codebase.Codebase

**Example:**

```
package.codebases.pack(directory='path_dir', name='codebase_name')
```

**pull\_git**(*codebase*, *local\_path*)

Pull (download) a codebase.

**Prerequisites:** You must be in the role of an *owner* or *developer*. You must have a package.

**Parameters**

- **codebase** (dtlpy.entities.codebase.Codebase) – codebase object
- **local\_path** (*str*) – local path

**Returns**

path where the Pull will be

**Return type**

*str*

**Example:**

```
package.codebases.pull_git(codebase='codebase_entity', local_path='local_path')
```

**unpack**(*codebase*: *Optional[Codebase]* = None, *codebase\_name*: *Optional[str]* = None, *codebase\_id*: *Optional[str]* = None, *local\_path*: *Optional[str]* = None, *version*: *Optional[str]* = None)

Unpack codebase locally. Download source code and unzip.

**Prerequisites:** You must be in the role of an *owner* or *developer*. You must have a package.

**Parameters**

- **codebase** (dtlpy.entities.codebase.Codebase) – *dtl.Codebase* object
- **codebase\_name** (*str*) – search by name
- **codebase\_id** (*str*) – search by id
- **local\_path** (*str*) – local path to save codebase
- **version** (*str*) – codebase version to unpack. default - latest

**Returns**

String (dirpath)

**Return type**`str`**Example:**

```
package.codebases.unpack(codebase='codebase_entity', local_path='local_path')
```

## 2.9 Services

```
class ServiceLog(_json: dict, service: Service, services: Services, start=None, follow=None,
                 execution_id=None, function_name=None, replica_id=None, system=False)
```

Bases: `object`

Service Log

**view**(until\_completed)

View logs

**Parameters**

**until\_completed** –

```
class Services(client_api: ApiClient, project: Optional[Project] = None, package: Optional[Package] = None,
               project_id=None)
```

Bases: `object`

Services Repository

The Services class allows the user to manage services and their properties. Services are created from the packages users create. See our documentation for more information about [services](#).

```
activate_slots(service: Service, project_id: Optional[str] = None, task_id: Optional[str] = None,
               dataset_id: Optional[str] = None, org_id: Optional[str] = None, user_email:
               Optional[str] = None, slots: Optional[List[PackageSlot]] = None, role=None,
               prevent_override: bool = True, visible: bool = True, icon: str = 'fas fa-magic', **kwargs)
```

Activate service slots (creates buttons in the UI that activate services).

**Prerequisites:** You must be in the role of an *owner* or *developer*. You must have a package.

**Parameters**

- **service** (`dtlpy.entities.service.Service`) – service entity
- **project\_id** (`str`) – project id
- **task\_id** (`str`) – task id
- **dataset\_id** (`str`) – dataset id
- **org\_id** (`str`) – org id
- **user\_email** (`str`) – user email
- **slots** (`list`) – list of entities.PackageSlot
- **role** (`str`) – user role MemberOrgRole.ADMIN, MemberOrgRole.owner, MemberOrgRole.MEMBER
- **prevent\_override** (`bool`) – True to prevent override
- **visible** (`bool`) – visible

- **icon** (*str*) – icon
- **kwargs** – all additional arguments

**Returns**

list of user setting for activated slots

**Return type**

*list*

**Example:**

```
package.services.activate_slots(service='service_entity',
                               project_id='project_id',
                               slots=List[entities.PackageSlot],
                               icon='fas fa-magic')
```

**checkout** (*service: Optional[Service] = None, service\_name: Optional[str] = None, service\_id: Optional[str] = None*)

Checkout (switch) to a service.

**Prerequisites:** You must be in the role of an *owner* or *developer*. You must have a package.

**Parameters**

- **service** (*dtlpy.entities.service.Service*) – Service entity
- **service\_name** (*str*) – service name
- **service\_id** (*str*) – service id

**Example:**

```
package.services.checkout(service_id='service_id')
```

**delete** (*service\_name: Optional[str] = None, service\_id: Optional[str] = None*)

Delete Service object

**Prerequisites:** You must be in the role of an *owner* or *developer*. You must have a package.

You must provide at least ONE of the following params: *service\_id*, *service\_name*.

**Parameters**

- **service\_name** (*str*) – by name
- **service\_id** (*str*) – by id

**Returns**

True

**Return type**

*bool*

**Example:**

```
package.services.delete(service_id='service_id')
```

**deploy**(*service\_name*: *Optional*[*str*] = *None*, *package*: *Optional*[*Package*] = *None*, *bot*: *Optional*[*Union*[*Bot*, *str*]] = *None*, *revision*: *Optional*[*str*] = *None*, *init\_input*: *Optional*[*Union*[*List*[*FunctionIO*], *FunctionIO*, *dict*]] = *None*, *runtime*: *Optional*[*Union*[*KubernetesRuntime*, *dict*]] = *None*, *pod\_type*: *Optional*[*InstanceCatalog*] = *None*, *sdk\_version*: *Optional*[*str*] = *None*, *agent\_versions*: *Optional*[*dict*] = *None*, *verify*: *bool* = *True*, *checkout*: *bool* = *False*, *module\_name*: *Optional*[*str*] = *None*, *project\_id*: *Optional*[*str*] = *None*, *driver\_id*: *Optional*[*str*] = *None*, *func*: *Optional*[*Callable*] = *None*, *run\_execution\_as\_process*: *Optional*[*bool*] = *None*, *execution\_timeout*: *Optional*[*int*] = *None*, *drain\_time*: *Optional*[*int*] = *None*, *max\_attempts*: *Optional*[*int*] = *None*, *on\_reset*: *Optional*[*str*] = *None*, *force*: *bool* = *False*, *secrets*: *Optional*[*list*] = *None*, *\*\*kwargs*) → *Service*

Deploy service.

**Prerequisites:** You must be in the role of an *owner* or *developer*. You must have a package.

#### Parameters

- **service\_name** (*str*) – name
- **package** (*dtlpy.entities.package.Package*) – package entity
- **bot** (*str*) – bot email
- **revision** (*str*) – package revision of version
- **init\_input** – config to run at startup
- **runtime** (*dict*) – runtime resources
- **pod\_type** (*str*) – pod type *dl.InstanceCatalog*
- **sdk\_version** (*str*) –
  - optional - string - sdk version
- **agent\_versions** (*str*) –
  - dictionary - - optional -versions of sdk
- **verify** (*bool*) – if true, verify the inputs
- **checkout** (*bool*) – if true, checkout (switch) to service
- **module\_name** (*str*) – module name
- **project\_id** (*str*) – project id
- **driver\_id** (*str*) – driver id
- **func** (*Callable*) – function to deploy
- **run\_execution\_as\_process** (*bool*) – if true, run execution as process
- **execution\_timeout** (*int*) – execution timeout in seconds
- **drain\_time** (*int*) – drain time in seconds
- **max\_attempts** (*int*) – maximum execution retries in-case of a service reset
- **on\_reset** (*str*) – what happens on reset
- **force** (*bool*) – optional - if true, terminate old replicas immediately
- **secrets** (*list*) – list of the integrations ids
- **kwargs** – list of additional arguments

#### Returns

Service object

**Return type***dtlpy.entities.service.Service***Example:**

```
package.services.deploy(service_name=package_name,
                        execution_timeout=3 * 60 * 60,
                        module_name=module.name,
                        runtime=dl.KubernetesRuntime(
                            concurrency=10,
                            pod_type=dl.InstanceCatalog.REGULAR_S,
                            autoscaler=dl.KubernetesRabbitmqAutoscaler(
                                min_replicas=1,
                                max_replicas=20,
                                queue_length=20
                            )
                        )
                    )
```

**deploy\_from\_local\_folder**(*cwd=None, service\_file=None, bot=None, checkout=False, force=False*) → *Service*

Deploy from local folder in local environment.

**Prerequisites:** You must be in the role of an *owner* or *developer*. You must have a package.

**Parameters**

- **cwd** (*str*) – optional - package working directory. Default=cwd
- **service\_file** (*str*) – optional - service file. Default=None
- **bot** (*str*) – bot
- **checkout** – checkout
- **force** (*bool*) – optional - terminate old replicas immediately

**Returns**

Service object

**Return type***dtlpy.entities.service.Service***Example:**

```
package.services.deploy_from_local_folder(cwd='file_path',
                                          service_file='service_file')
```

**execute**(*service: Optional[Service] = None, service\_id: Optional[str] = None, service\_name: Optional[str] = None, sync: bool = False, function\_name: Optional[str] = None, stream\_logs: bool = False, execution\_input=None, resource=None, item\_id=None, dataset\_id=None, annotation\_id=None, project\_id=None*) → *Execution*

Execute a function on an existing service.

**Prerequisites:** You must be in the role of an *owner* or *developer*. You must have a package.

**Parameters**

- **service** (*dtlpy.entities.service.Service*) – service entity
- **service\_id** (*str*) – service id

- **service\_name** (*str*) – service name
- **sync** (*bool*) – wait for function to end
- **function\_name** (*str*) – function name to run
- **stream\_logs** (*bool*) – prints logs of the new execution. only works with sync=True
- **execution\_input** – input dictionary or list of FunctionIO entities
- **resource** (*str*) – dl.PackageInputType - input type.
- **item\_id** (*str*) – str - optional - input to function
- **dataset\_id** (*str*) – str - optional - input to function
- **annotation\_id** (*str*) – str - optional - input to function
- **project\_id** (*str*) – str - resource's project

**Returns**

entities.Execution

**Return type**

*dtlpy.entities.execution.Execution*

**Example:**

```
package.services.execute(service='service_entity',
                        function_name='run',
                        item_id='item_id',
                        project_id='project_id')
```

**get**(service\_name=None, service\_id=None, checkout=False, fetch=None) → *Service*

Get service to use in your code.

**Prerequisites:** You must be in the role of an *owner* or *developer*. You must have a package.

**Parameters**

- **service\_name** (*str*) – optional - search by name
- **service\_id** (*str*) – optional - search by id
- **checkout** (*bool*) – if true, checkout (switch) to service
- **fetch** – optional - fetch entity from platform, default taken from cookie

**Returns**

Service object

**Return type**

*dtlpy.entities.service.Service*

**Example:**

```
package.services.get(service_id='service_id')
```

**list**(filters: *Optional[Filters]* = None) → *PagedEntities*

List all services (services can be listed for a package or for a project).

**Prerequisites:** You must be in the role of an *owner* or *developer*. You must have a package.

**Parameters**

**filters** (`dtlpy.entities.filters.Filters`) – Filters entity or a dictionary containing filters parameters

**Returns**

Paged entity

**Return type**

`dtlpy.entities.paged_entities.PagedEntities`

**Example:**

```
package.services.list()
```

```
log(service, size=100, checkpoint=None, start=None, end=None, follow=False, text=None,
     execution_id=None, function_name=None, replica_id=None, system=False, view=True,
     until_completed=True)
```

Get service logs.

**Prerequisites:** You must be in the role of an *owner* or *developer*. You must have a package.

**Parameters**

- **service** (`dtlpy.entities.service.Service`) – service object
- **size** (`int`) – size
- **checkpoint** (`dict`) – the information from the 1st point checked in the service
- **start** (`str`) – iso format time
- **end** (`str`) – iso format time
- **follow** (`bool`) – if true, keep stream future logs
- **text** (`str`) – text
- **execution\_id** (`str`) – execution id
- **function\_name** (`str`) – function name
- **replica\_id** (`str`) – replica id
- **system** (`bool`) – system
- **view** (`bool`) – if true, print out all the logs
- **until\_completed** (`bool`) – wait until completed

**Returns**

ServiceLog entity

**Return type**

`ServiceLog`

**Example:**

```
package.services.log(service='service_entity')
```

```
name_validation(name: str)
```

Validation service name.

**Prerequisites:** You must be in the role of an *owner* or *developer*.

**Parameters**

**name** (*str*) – service name

**Example:**

```
package.services.name_validation(name='name')
```

**open\_in\_web**(*service*: *Optional*[*Service*] = *None*, *service\_id*: *Optional*[*str*] = *None*, *service\_name*: *Optional*[*str*] = *None*)

Open the service in web platform

**Prerequisites:** You must be in the role of an *owner* or *developer*. You must have a package.

**Parameters**

- **service\_name** (*str*) – service name
- **service\_id** (*str*) – service id
- **service** (*dtlpy.entities.service.Service*) – service entity

**Example:**

```
package.services.open_in_web(service_id='service_id')
```

**pause**(*service\_name*: *Optional*[*str*] = *None*, *service\_id*: *Optional*[*str*] = *None*, *force*: *bool* = *False*)

Pause service.

**Prerequisites:** You must be in the role of an *owner* or *developer*. You must have a package.

You must provide at least ONE of the following params: *service\_id*, *service\_name*

**Parameters**

- **service\_name** (*str*) – service name
- **service\_id** (*str*) – service id
- **force** (*bool*) – optional - terminate old replicas immediately

**Returns**

True if success

**Return type**

*bool*

**Example:**

```
package.services.pause(service_id='service_id')
```

**resume**(*service\_name*: *Optional*[*str*] = *None*, *service\_id*: *Optional*[*str*] = *None*, *force*: *bool* = *False*)

Resume service.

**Prerequisites:** You must be in the role of an *owner* or *developer*. You must have a package.

You must provide at least ONE of the following params: *service\_id*, *service\_name*.

**Parameters**

- **service\_name** (*str*) – service name
- **service\_id** (*str*) – service id
- **force** (*bool*) – optional - terminate old replicas immediately



**Returns**

json of the service

**Return type**

dict

**Example:**

```
package.services.resume(service_id='service_id')
```

**revisions**(*service*: *Optional*[*Service*] = *None*, *service\_id*: *Optional*[*str*] = *None*)

Get service revisions history.

**Prerequisites:** You must be in the role of an *owner* or *developer*. You must have a package.

You must provide at least ONE of the following params: *service*, *service\_id*

**Parameters**

- **service** (*dtlpy.entities.service.Service*) – Service entity
- **service\_id** (*str*) – service id

**Example:**

```
package.services.revisions(service_id='service_id')
```

**status**(*service\_name*=*None*, *service\_id*=*None*)

Get service status.

**Prerequisites:** You must be in the role of an *owner* or *developer*. You must have a package.

You must provide at least ONE of the following params: *service\_id*, *service\_name*

**Parameters**

- **service\_name** (*str*) – service name
- **service\_id** (*str*) – service id

**Returns**

status json

**Return type**

dict

**Example:**

```
package.services.status(service_id='service_id')
```

**update**(*service*: *Service*, *force*: *bool* = *False*) → *Service*

Update service changes to platform.

**Prerequisites:** You must be in the role of an *owner* or *developer*. You must have a package.

**Parameters**

- **service** (*dtlpy.entities.service.Service*) – Service entity
- **force** (*bool*) – optional - terminate old replicas immediately

**Returns**

Service entity

**Return type***dtlpy.entities.service.Service***Example:**

```
package.services.update(service='service_entity')
```

## 2.9.1 Bots

**class** `Bots`(*client\_api*: *ApiClient*, *project*: *Project*)

Bases: `object`

Bots Repository

The Bots class allows the user to manage bots and their properties. See our documentation for more information on [bots](#).

**create**(*name*: *str*, *return\_credentials*: *bool* = *False*)

Create a new Bot.

**Prerequisites:** You must be in the role of an *owner* or *developer*. You must have a service.

**Parameters**

- **name** (*str*) – bot name
- **return\_credentials** (*str*) – True will return the password when created

**Returns**

Bot object

**Return type***dtlpy.entities.bot.Bot***Example:**

```
service.bots.delete(name='bot', return_credentials=False)
```

**delete**(*bot\_id*: *Optional[str]* = *None*, *bot\_email*: *Optional[str]* = *None*)

Delete a Bot.

**Prerequisites:** You must be in the role of an *owner* or *developer*. You must have a service.

You must provide at least ONE of the following params: *bot\_id*, *bot\_email*

**Parameters**

- **bot\_id** (*str*) – bot id to delete
- **bot\_email** (*str*) – bot email to delete

**Returns**

True if successful

**Return type***bool***Example:**

```
service.bots.delete(bot_id='bot_id')
```

**get**(*bot\_email*: *Optional*[*str*] = None, *bot\_id*: *Optional*[*str*] = None, *bot\_name*: *Optional*[*str*] = None)

Get a Bot object.

**Prerequisites:** You must be in the role of an *owner* or *developer*. You must have a service.

**Parameters**

- **bot\_email** (*str*) – get bot by email
- **bot\_id** (*str*) – get bot by id
- **bot\_name** (*str*) – get bot by name

**Returns**

Bot object

**Return type**

*dtlpy.entities.bot.Bot*

**Example:**

```
service.bots.get(bot_id='bot_id')
```

**list**() → List[*Bot*]

Get a project or package bots list.

**Prerequisites:** You must be in the role of an *owner* or *developer*. You must have a service.

**Returns**

List of Bots objects

**Return type**

list

**Example:**

```
service.bots.list()
```

## 2.10 Triggers

**class Triggers**(*client\_api*: *ApiClient*, *project*: *Optional*[*Project*] = None, *service*: *Optional*[*Service*] = None, *project\_id*: *Optional*[*str*] = None, *pipeline*: *Optional*[*Pipeline*] = None)

Bases: *object*

Triggers Repository

The Triggers class allows users to manage triggers and their properties. Triggers activate services. See our documentation for more information on [triggers](#).

**create**(*service\_id*: *Optional*[*str*] = None, *trigger\_type*: *TriggerType* = *TriggerType.EVENT*, *name*: *Optional*[*str*] = None, *webhook\_id*=None, *function\_name*='run', *project\_id*=None, *active*=True, *filters*=None, *resource*: *TriggerResource* = *TriggerResource.ITEM*, *actions*: *Optional*[*TriggerAction*] = None, *execution\_mode*: *TriggerExecutionMode* = *TriggerExecutionMode.ONCE*, *start\_at*=None, *end\_at*=None, *inputs*=None, *cron*=None, *pipeline\_id*=None, *pipeline*=None, *pipeline\_node\_id*=None, *root\_node\_namespace*=None, *\*\*kwargs*) → *BaseTrigger*

Create a Trigger. Can create two types: a cron trigger or an event trigger. Inputs are different for each type

**Prerequisites:** You must be in the role of an *owner* or *developer*. You must have a service.

Inputs for all types:

### Parameters

- **service\_id** (*str*) – Id of services to be triggered
- **trigger\_type** (*str*) – can be cron or event. use enum `dl.TriggerType` for the full list
- **name** (*str*) – name of the trigger
- **webhook\_id** (*str*) – id for webhook to be called
- **function\_name** (*str*) – the function name to be called when triggered (must be defined in the package)
- **project\_id** (*str*) – project id where trigger will work
- **active** (*bool*) – optional - True/False, default = True, if true trigger is active

Inputs for event trigger: :param `dtlpy.entities.filters.Filters` filters: optional - Item/Annotation metadata filters, default = none :param `str` resource: optional - Dataset/Item/Annotation/ItemStatus, default = Item :param `str` actions: optional - Created/Updated/Deleted, default = create :param `str` execution\_mode: how many times trigger should be activated; default is “Once”. enum `dl.TriggerExecutionMode`

Inputs for cron trigger: :param `start_at`: iso format date string to start activating the cron trigger :param `end_at`: iso format date string to end the cron activation :param `inputs`: dictionary “name”:”val” of inputs to the function :param `str` cron: cron spec specifying when it should run. more information: <https://en.wikipedia.org/wiki/Cron> :param `str` pipeline\_id: Id of pipeline to be triggered :param pipeline: pipeline entity to be triggered :param `str` pipeline\_node\_id: Id of pipeline root node to be triggered :param `str` root\_node\_namespace: namespace of pipeline root node to be triggered

### Returns

Trigger entity

### Return type

*dtlpy.entities.trigger.Trigger*

### Example:

```
service.triggers.create(name='triggername',
                        execution_mode=dl.TriggerExecutionMode.ONCE,
                        resource='Item',
                        actions='Created',
                        function_name='run',
                        filters={'$and': [{'hidden': False},
                                         {'type': 'file'}]}
                        )
```

**delete**(*trigger\_id=None, trigger\_name=None*)

Delete Trigger object

**Prerequisites:** You must be in the role of an *owner* or *developer*. You must have a service.

### Parameters

- **trigger\_id** (*str*) – trigger id
- **trigger\_name** (*str*) – trigger name

### Returns

True is successful error if not

### Return type

*bool*

**Example:**

```
service.triggers.delete(trigger_id='trigger_id')
```

**get**(trigger\_id=None, trigger\_name=None) → *BaseTrigger*

Get Trigger object

**Prerequisites:** You must be in the role of an *owner* or *developer*. You must have a service.

**Parameters**

- **trigger\_id** (*str*) – trigger id
- **trigger\_name** (*str*) – trigger name

**Returns**

Trigger entity

**Return type**

*dtlpy.entities.trigger.Trigger*

**Example:**

```
service.triggers.get(trigger_id='trigger_id')
```

**list**(filters: *Optional[Filters]* = None) → *PagedEntities*

List triggers of a project, package, or service.

**Prerequisites:** You must be in the role of an *owner* or *developer*. You must have a service.

**Parameters**

**filters** (*dtlpy.entities.filters.Filters*) – Filters entity or a dictionary containing filters parameters

**Returns**

Paged entity

**Return type**

*dtlpy.entities.paged\_entities.PagedEntities*

**Example:**

```
service.triggers.list()
```

**name\_validation**(name: *str*)

This method validates the trigger name. If name is not valid, this method will return an error. Otherwise, it will not return anything.

**Parameters**

**name** (*str*) – trigger name

**resource\_information**(resource, resource\_type, action='Created')

Returns which function should run on an item (based on global triggers).

**Prerequisites:** You must be a **superuser** to run this method.

**Parameters**

- **resource** – 'Item' / 'Dataset' / etc
- **resource\_type** – dictionary of the resource object
- **action** – 'Created' / 'Updated' / etc.

Example:

```
service.triggers.resource_information(resource='Item', resource_type=item_
↪object, action='Created')
```

**update**(*trigger*: [BaseTrigger](#)) → [BaseTrigger](#)

Update trigger

**Prerequisites:** You must be in the role of an *owner* or *developer*. You must have a service.

**Parameters**

**trigger** ([dtlpy.entities.trigger.Trigger](#)) – Trigger entity

**Returns**

Trigger entity

**Return type**

[dtlpy.entities.trigger.Trigger](#)

Example:

```
service.triggers.update(trigger='trigger_entity')
```

## 2.11 Executions

**class Executions**(*client\_api*: [ApiClient](#), *service*: [Optional](#)[[Service](#)] = None, *project*: [Optional](#)[[Project](#)] = None)

Bases: [object](#)

Service Executions Repository

The Executions class allows the users to manage executions (executions of services) and their properties. See our documentation for more information about [executions](#).

**create**(*service\_id*: [Optional](#)[[str](#)] = None, *execution\_input*: [Optional](#)[[list](#)] = None, *function\_name*: [Optional](#)[[str](#)] = None, *resource*: [Optional](#)[[PackageInputType](#)] = None, *item\_id*: [Optional](#)[[str](#)] = None, *dataset\_id*: [Optional](#)[[str](#)] = None, *annotation\_id*: [Optional](#)[[str](#)] = None, *project\_id*: [Optional](#)[[str](#)] = None, *sync*: [bool](#) = False, *stream\_logs*: [bool](#) = False, *return\_output*: [bool](#) = False, *return\_curl\_only*: [bool](#) = False, *timeout*: [Optional](#)[[int](#)] = None) → [Execution](#)

Execute a function on an existing service

**Prerequisites:** You must be in the role of an *owner* or *developer*. You must have a service.

**Parameters**

- **service\_id** ([str](#)) – service id to execute on
- **execution\_input** ([List](#)[[FunctionIO](#)] or [dict](#)) – input dictionary or list of [FunctionIO](#) entities
- **function\_name** ([str](#)) – function name to run
- **resource** ([str](#)) – input type.
- **item\_id** ([str](#)) – optional - item id as input to function
- **dataset\_id** ([str](#)) – optional - dataset id as input to function
- **annotation\_id** ([str](#)) – optional - annotation id as input to function
- **project\_id** ([str](#)) – resource's project

- **sync** (*bool*) – if true, wait for function to end
- **stream\_logs** (*bool*) – prints logs of the new execution. only works with sync=True
- **return\_output** (*bool*) – if True and sync is True - will return the output directly
- **return\_curl\_only** (*bool*) – return the cURL of the creation WITHOUT actually do it
- **timeout** (*int*) – int, seconds to wait until TimeoutError is raised. if <=0 - wait until done  
- by default wait take the service timeout

**Returns**

execution object

**Return type**

*dtlpy.entities.execution.Execution*

**Example:**

```
service.executions.create(function_name='function_name', item_id='item_id',
↪project_id='project_id')
```

**get**(*execution\_id: Optional[str] = None, sync: bool = False*) → *Execution*

Get Service execution object

**Prerequisites:** You must be in the role of an *owner* or *developer*. You must have a service.

**Parameters**

- **execution\_id** (*str*) – execution id
- **sync** (*bool*) – if true, wait for the execution to finish

**Returns**

Service execution object

**Return type**

*dtlpy.entities.execution.Execution*

**Example:**

```
service.executions.get(execution_id='execution_id')
```

**increment**(*execution: Execution*)

Increment the number of attempts that an execution is allowed to attempt to run a service that is not responding.

**Prerequisites:** You must be in the role of an *owner* or *developer*. You must have a service.

**Parameters**

**execution** (*dtlpy.entities.execution.Execution*) –

**Returns**

int

**Return type**

int

**Example:**

```
service.executions.increment(execution='execution_entity')
```

**list**(filters: *Optional*[Filters] = None) → *PagedEntities*

List service executions

**Prerequisites:** You must be in the role of an *owner* or *developer*. You must have a service.

**Parameters**

**filters** (dtlpy.entities.filters.Filters) – dl.Filters entity to filters items

**Returns**

Paged entity

**Return type**

*dtlpy.entities.paged\_entities.PagedEntities*

**Example:**

```
service.executions.list()
```

**logs**(execution\_id: *str*, follow: *bool* = True, until\_completed: *bool* = True)

executions logs

**Prerequisites:** You must be in the role of an *owner* or *developer*. You must have a service.

**Parameters**

- **execution\_id** (*str*) – execution id
- **follow** (*bool*) – if true, keep stream future logs
- **until\_completed** (*bool*) – if true, wait until completed

**Returns**

executions logs

**Example:**

```
service.executions.logs(execution_id='execution_id')
```

**progress\_update**(execution\_id: *str*, status: *Optional*[ExecutionStatus] = None, percent\_complete: *Optional*[int] = None, message: *Optional*[*str*] = None, output: *Optional*[*str*] = None, service\_version: *Optional*[*str*] = None)

Update Execution Progress.

**Prerequisites:** You must be in the role of an *owner* or *developer*. You must have a service.

**Parameters**

- **execution\_id** (*str*) – execution id
- **status** (*str*) – ExecutionStatus
- **percent\_complete** (*int*) – percent work done
- **message** (*str*) – message
- **output** (*str*) – the output of the execution
- **service\_version** (*str*) – service version

**Returns**

Service execution object

**Return type**

*dtlpy.entities.execution.Execution*



**Example:**

```
service.executions.progress_update(execution_id='execution_id', status='complete
↪', percent_complete=100)
```

**rerun**(*execution*: [Execution](#), *sync*: *bool* = *False*)

Rerun execution

**Prerequisites:** You must be in the role of an *owner* or *developer*. You must have a service.

**Parameters**

- **execution** ([dtlpy.entities.execution.Execution](#)) –
- **sync** (*bool*) – wait for the execution to finish

**Returns**

Execution object

**Return type**

[dtlpy.entities.execution.Execution](#)

**Example:**

```
service.executions.rerun(execution='execution_entity')
```

**terminate**(*execution*: [Execution](#))

Terminate Execution

**Prerequisites:** You must be in the role of an *owner* or *developer*. You must have a service.

**Parameters**

- **execution** ([dtlpy.entities.execution.Execution](#)) –

**Returns**

execution object

**Return type**

[dtlpy.entities.execution.Execution](#)

**Example:**

```
service.executions.terminate(execution='execution_entity')
```

**update**(*execution*: [Execution](#)) → [Execution](#)

Update execution changes to platform

**Prerequisites:** You must be in the role of an *owner* or *developer*. You must have a service.

**Parameters**

- **execution** ([dtlpy.entities.execution.Execution](#)) – execution entity

**Returns**

Service execution object

**Return type**

[dtlpy.entities.execution.Execution](#)

**Example:**

```
service.executions.update(execution='execution_entity')
```

**wait**(*execution\_id*: *str*, *timeout*: *Optional[int]* = *None*)

Get Service execution object.

**Prerequisites:** You must be in the role of an *owner* or *developer*. You must have a service.

**Parameters**

- **execution\_id** (*str*) – execution id
- **timeout** (*int*) – seconds to wait until TimeoutError is raised. if <=0 - wait until done - by default wait take the service timeout

**Returns**

Service execution object

**Return type**

*dtlpy.entities.execution.Execution*

**Example:**

```
service.executions.wait(execution_id='execution_id')
```

## 2.12 Pipelines

**class Pipelines**(*client\_api*: *ApiClient*, *project*: *Optional[Project]* = *None*)

Bases: *object*

Pipelines Repository

The Pipelines class allows users to manage pipelines and their properties. See our documentation for more information on [pipelines](#).

**create**(*name*: *Optional[str]* = *None*, *project\_id*: *Optional[str]* = *None*, *pipeline\_json*: *Optional[dict]* = *None*) → *Pipeline*

Create a new pipeline.

**prerequisites:** You must be an *owner* or *developer* to use this method.

**Parameters**

- **name** (*str*) – pipeline name
- **project\_id** (*str*) – project id
- **pipeline\_json** (*dict*) – json containing the pipeline fields

**Returns**

Pipeline object

**Return type**

*dtlpy.entities.pipeline.Pipeline*

**Example:**

```
project.pipelines.create(name='pipeline_name')
```

**delete**(*pipeline*: *Optional[Pipeline]* = *None*, *pipeline\_name*: *Optional[str]* = *None*, *pipeline\_id*: *Optional[str]* = *None*)

Delete Pipeline object.

**prerequisites:** You must be an *owner* or *developer* to use this method.

#### Parameters

- **pipeline** (`dtlpy.entities.pipeline.Pipeline`) – pipeline entity
- **pipeline\_id** (`str`) – pipeline id
- **pipeline\_name** (`str`) – pipeline name

#### Returns

True if success

#### Return type

`bool`

#### Example:

```
project.pipelines.delete(pipeline_id='pipeline_id')
```

**execute** (*pipeline*: `Optional[Pipeline]` = `None`, *pipeline\_id*: `Optional[str]` = `None`, *pipeline\_name*: `Optional[str]` = `None`, *execution\_input*=`None`)

Execute a pipeline and return the pipeline execution as an object.

**prerequisites:** You must be an *owner* or *developer* to use this method.

#### Parameters

- **pipeline** (`dtlpy.entities.pipeline.Pipeline`) – pipeline entity
- **pipeline\_id** (`str`) – pipeline id
- **pipeline\_name** (`str`) – pipeline name
- **execution\_input** – list of the `dl.FunctionIO` or dict of pipeline input - example { 'item': 'item\_id' }

#### Returns

`entities.PipelineExecution` object

#### Return type

`dtlpy.entities.pipeline_execution.PipelineExecution`

#### Example:

```
project.pipelines.execute(pipeline='pipeline_entity', execution_input= {'item':
↪ 'item_id'} )
```

**get** (*pipeline\_name*=`None`, *pipeline\_id*=`None`, *fetch*=`None`) → `Pipeline`

Get Pipeline object to use in your code.

**prerequisites:** You must be an *owner* or *developer* to use this method.

You must provide at least ONE of the following params: `pipeline_name`, `pipeline_id`.

#### Parameters

- **pipeline\_id** (`str`) – pipeline id
- **pipeline\_name** (`str`) – pipeline name
- **fetch** – optional - fetch entity from platform, default taken from cookie

**Returns**

Pipeline object

**Return type***dtlpy.entities.pipeline.Pipeline***Example:**

```
project.pipelines.get(pipeline_id='pipeline_id')
```

**install**(*pipeline: Optional[Pipeline] = None*)

Install (start) a pipeline.

**prerequisites:** You must be an *owner* or *developer* to use this method.**Parameters****pipeline** (*dtlpy.entities.pipeline.Pipeline*) – pipeline entity**Returns**

Composition object

**Example:**

```
project.pipelines.install(pipeline='pipeline_entity')
```

**list**(*filters: Optional[Filters] = None, project\_id: Optional[str] = None*) → *PagedEntities*

List project pipelines.

**prerequisites:** You must be an *owner* or *developer* to use this method.**Parameters**

- **filters** (*dtlpy.entities.filters.Filters*) – Filters entity or a dictionary containing filters parameters
- **project\_id** (*str*) – project id

**Returns**

Paged entity

**Return type***dtlpy.entities.paged\_entities.PagedEntities***Example:**

```
project.pipelines.get()
```

**open\_in\_web**(*pipeline: Optional[Pipeline] = None, pipeline\_id: Optional[str] = None, pipeline\_name: Optional[str] = None*)

Open the pipeline in web platform.

**prerequisites:** Must be *owner* or *developer* to use this method.**Parameters**

- **pipeline** (*dtlpy.entities.pipeline.Pipeline*) – pipeline entity
- **pipeline\_id** (*str*) – pipeline id
- **pipeline\_name** (*str*) – pipeline name

**Example:**

```
project.pipelines.open_in_web(pipeline_id='pipeline_id')
```

**pause**(*pipeline: Optional[Pipeline] = None*)

Pause a pipeline.

**prerequisites:** You must be an *owner* or *developer* to use this method.

**Parameters**

**pipeline** (`dtlpy.entities.pipeline.Pipeline`) – pipeline entity

**Returns**

Composition object

**Example:**

```
project.pipelines.pause(pipeline='pipeline_entity')
```

**reset**(*pipeline: Optional[Pipeline] = None, pipeline\_id: Optional[str] = None, pipeline\_name: Optional[str] = None, stop\_if\_running: bool = False*)

Reset pipeline counters.

**prerequisites:** You must be an *owner* or *developer* to use this method.

**Parameters**

- **pipeline** (`dtlpy.entities.pipeline.Pipeline`) – pipeline entity - optional
- **pipeline\_id** (*str*) – pipeline\_id - optional
- **pipeline\_name** (*str*) – pipeline\_name - optional
- **stop\_if\_running** (*bool*) – If the pipeline is installed it will stop the pipeline and reset the counters.

**Returns**

bool

**Example:**

```
project.pipelines.reset(pipeline='pipeline_entity')
```

**stats**(*pipeline: Optional[Pipeline] = None, pipeline\_id: Optional[str] = None, pipeline\_name: Optional[str] = None*)

Get pipeline counters.

**prerequisites:** You must be an *owner* or *developer* to use this method.

**Parameters**

- **pipeline** (`dtlpy.entities.pipeline.Pipeline`) – pipeline entity - optional
- **pipeline\_id** (*str*) – pipeline\_id - optional
- **pipeline\_name** (*str*) – pipeline\_name - optional

**Returns**

PipelineStats

**Return type**

`dtlpy.entities.pipeline.PipelineStats`

**Example:**

```
project.pipelines.stats(pipeline='pipeline_entity')
```

**update**(*pipeline*: *Optional*[*Pipeline*] = *None*) → *Pipeline*

Update pipeline changes to platform.

**prerequisites:** You must be an *owner* or *developer* to use this method.

**Parameters**

**pipeline** (`dtlpy.entities.pipeline.Pipeline`) – pipeline entity

**Returns**

Pipeline object

**Return type**

`dtlpy.entities.pipeline.Pipeline`

**Example:**

```
project.pipelines.update(pipeline='pipeline_entity')
```

## 2.12.1 Pipeline Executions

**class PipelineExecutions**(*client\_api*: *ApiClient*, *project*: *Optional*[*Project*] = *None*, *pipeline*: *Optional*[*Pipeline*] = *None*)

Bases: `object`

PipelineExecutions Repository

The PipelineExecutions class allows users to manage pipeline executions. See our documentation for more information on [pipelines](#).

**create**(*pipeline\_id*: *Optional*[*str*] = *None*, *execution\_input*=*None*)

Execute a pipeline and return the execute.

**prerequisites:** You must be an *owner* or *developer* to use this method.

**Parameters**

- **pipeline\_id** – pipeline id
- **execution\_input** – list of the `dl.FunctionIO` or dict of pipeline input - example { 'item': 'item\_id' }

**Returns**

`entities.PipelineExecution` object

**Return type**

`dtlpy.entities.pipeline_execution.PipelineExecution`

**Example:**

```
pipeline.pipeline_executions.create(pipeline_id='pipeline_id', execution_input={
    ↪ 'item': 'item_id'})
```

**get**(*pipeline\_execution\_id*: *str*, *pipeline\_id*: *Optional*[*str*] = *None*) → *PipelineExecution*

Get Pipeline Execution object

**prerequisites:** You must be an *owner* or *developer* to use this method.

**Parameters**

- **pipeline\_execution\_id** (*str*) – pipeline execution id
- **pipeline\_id** (*str*) – pipeline id

**Returns**

PipelineExecution object

**Return type***dtlpy.entities.pipeline\_execution.PipelineExecution***Example:**

```
pipeline.pipeline_executions.get(pipeline_id='pipeline_id')
```

**list**(*filters: Optional[Filters] = None*) → *PagedEntities*

List project pipeline executions.

**prerequisites:** You must be an *owner* or *developer* to use this method.**Parameters**

**filters** (*dtlpy.entities.filters.Filters*) – Filters entity or a dictionary containing filters parameters

**Returns**

Paged entity

**Return type***dtlpy.entities.paged\_entities.PagedEntities***Example:**

```
pipeline.pipeline_executions.list()
```

## 2.13 General Commands

**class Commands**(*client\_api: ApiClient*)

Bases: *object*

Service Commands repository

**abort**(*command\_id: str*)

Abort Command

**Parameters**

**command\_id** (*str*) – command id

**Returns**

**get**(*command\_id: Optional[str] = None, url: Optional[str] = None*) → *Command*

Get Service command object

**Parameters**

- **command\_id** (*str*) –
- **url** (*str*) – command url

**Returns**

Command object

**list()**

List of commands

**Returns**

list of commands

**wait**(*command\_id*, *timeout=0*, *step=None*, *url=None*, *backoff\_factor=0.1*)

Wait for command to finish

*backoff\_factor*: A backoff factor to apply between attempts after the second try {backoff factor} \* (2 \*\* ({number of total retries} - 1)) seconds. If the *backoff\_factor* is 0.1, then `sleep()` will sleep for [0.0s, 0.2s, 0.4s, ...] between retries. It will never be longer than 8 sec

**Parameters**

- **command\_id** (*str*) – Command id to wait to
- **timeout** (*int*) – int, seconds to wait until `TimeoutError` is raised. if 0 - wait until done
- **step** (*int*) – int, seconds between polling
- **url** (*str*) – url to the command
- **backoff\_factor** (*float*) – A backoff factor to apply between attempts after the second try

**Returns**

Command object

### 2.13.1 Download Commands

### 2.13.2 Upload Commands



## 3.1 Organization

**class** `CacheAction`(*value*)

Bases: `str`, `Enum`

An enumeration.

**class** `MemberOrgRole`(*value*)

Bases: `str`, `Enum`

An enumeration.

**class** `Organization`(*members: list, groups: list, account: dict, created\_at, updated\_at, id, name, logo\_url, plan, owner, created\_by, client\_api: ApiClient, repositories=NOTHING*)

Bases: `BaseEntity`

Organization entity

**add\_member**(*email, role: ~dlpy.entities.organization.MemberOrgRole = <enum 'MemberOrgRole'>*)

Add members to your organization. Read about members and groups [here](<https://dataloop.ai/docs/org-members-groups>).

Prerequisites: To add members to an organization, you must be in the role of an “owner” in that organization.

### Parameters

- **email** (*str*) – the member’s email
- **role** (*str*) – `MemberOrgRole.ADMIN`, `MemberOrgRole.OWNER`, `MemberOrgRole.MEMBER`

### Returns

True if successful or error if unsuccessful

### Return type

`bool`

**cache\_action**(*mode=CacheAction.APPLY, pod\_type=PodType.SMALL*)

Open the organizations in web platform

### Parameters

- **mode** (*str*) – `dl.CacheAction.APPLY` or `dl.CacheAction.DESTROY`
- **pod\_type** (*dl.PodType*) – `dl.PodType.SMALL`, `dl.PodType.MEDIUM`, `dl.PodType.HIGH`

#### Returns

True if success

#### Return type

`bool`

**delete\_member**(*user\_id*: `str`, *sure*: `bool` = `False`, *really*: `bool` = `False`)

Delete member from the Organization.

Prerequisites: Must be an organization “owner” to delete members.

#### Parameters

- **user\_id** (`str`) – user id
- **sure** (`bool`) – Are you sure you want to delete?
- **really** (`bool`) – Really really sure?

#### Returns

True if success and error if not

#### Return type

`bool`

**classmethod from\_json**(*\_json*, *client\_api*, *is\_fetched*=`True`)

Build a Project entity object from a json

#### Parameters

- **is\_fetched** (`bool`) – is Entity fetched from Platform
- **\_json** (`dict`) – \_json response from host
- **client\_api** (`dl.ApiClient`) – ApiClient entity

#### Returns

Organization object

#### Return type

`dtlpy.entities.organization.Organization`

**list\_groups**()

List all organization groups (groups that were created within the organization).

Prerequisites: You must be an organization “owner” to use this method.

#### Returns

groups list

#### Return type

`list`

**list\_members**(*role*: `Optional[MemberOrgRole]` = `None`)

List all organization members.

Prerequisites: You must be an organization “owner” to use this method.

#### Parameters

**role** (`str`) – MemberOrgRole.ADMIN, MemberOrgRole.OWNER, MemberOrgRole.MEMBER

#### Returns

projects list

**Return type**`list`**open\_in\_web()**

Open the organizations in web platform

**to\_json()**

Returns platform \_json format of object

**Returns**

platform json format of object

**Return type**`dict`**update(plan: str)**

Update Organization.

Prerequisites: You must be an Organization **superuser** to update an organization.

**Parameters**

**plan** (`str`) – OrganizationsPlans.FREEMIUM, OrganizationsPlans.PREMIUM

**Returns**

organization object

**update\_member(email: str, role: MemberOrgRole = MemberOrgRole.MEMBER)**

Update member role.

Prerequisites: You must be an organization “owner” to update a member’s role.

**Parameters**

- **email** (`str`) – the member’s email
- **role** (`str`) – MemberOrgRole.ADMIN, MemberOrgRole.OWNER, MemberOrgRole.MEMBER

**Returns**

json of the member fields

**Return type**`dict`**class OrganizationsPlans(value)**

Bases: `str`, `Enum`

An enumeration.

**class PodType(value)**

Bases: `str`, `Enum`

An enumeration.

### 3.1.1 Integration

**class** **Integration**(*id, name, type, org, created\_at, created\_by, update\_at, client\_api: ApiClient, project=None*)

Bases: `BaseEntity`

Integration object

**delete**(*sure: bool = False, really: bool = False*)  $\rightarrow$  `bool`

Delete integrations from the Organization

**Parameters**

- **sure** (*bool*) – are you sure you want to delete?
- **really** (*bool*) – really really?

**Returns**

`True`

**Return type**

`bool`

**classmethod** **from\_json**(*\_json: dict, client\_api: ApiClient, is\_fetched=True*)

Build a Integration entity object from a json

**Parameters**

- **\_json** – \_json response from host
- **client\_api** – ApiClient entity
- **is\_fetched** – is Entity fetched from Platform

**Returns**

Integration object

**to\_json**()

Returns platform \_json format of object

**Returns**

platform json format of object

**Return type**

`dict`

**update**(*new\_name: str*)

Update the integrations name

**Parameters**

**new\_name** (*str*) – new name

## 3.2 Project

**class** **MemberRole**(*value*)

Bases: `str, Enum`

An enumeration.

```
class Project(contributors, created_at, creator, id, name, org, updated_at, role, account, is_blocked,  
              feature_constraints, client_api: ApiClient, repositories=NOTHING)
```

Bases: BaseEntity

Project entity

```
add_member(email, role: MemberRole = MemberRole.DEVELOPER)
```

Add a member to the project.

#### Parameters

**email** (*str*) – member email

::param role: dl.MemberRole.OWNER, dl.MemberRole.DEVELOPER, dl.MemberRole.ANNOTATOR,  
dl.MemberRole.ANNOTATION\_MANAGER :return: dict that represent the user :rtype: dict

```
checkout()
```

Checkout the project

```
delete(sure=False, really=False)
```

Delete the project forever!

#### Parameters

- **sure** (*bool*) – are you sure you want to delete?
- **really** (*bool*) – really really?

#### Returns

True

#### Return type

bool

```
classmethod from_json(_json, client_api, is_fetched=True)
```

Build a Project entity object from a json

#### Parameters

- **is\_fetched** (*bool*) – is Entity fetched from Platform
- **\_json** (*dict*) – \_json response from host
- **client\_api** (*dl.ApiClient*) – ApiClient entity

#### Returns

Project object

#### Return type

*dtlpy.entities.project.Project*

```
list_members(role: Optional[MemberRole] = None)
```

List the project members.

#### Parameters

**role** – dl.MemberRole.OWNER, dl.MemberRole.DEVELOPER,  
dl.MemberRole.ANNOTATOR, dl.MemberRole.ANNOTATION\_MANAGER

#### Returns

list of the project members

#### Return type

list

**open\_in\_web()**

Open the project in web platform

**remove\_member(email)**

Remove a member from the project.

**Parameters**

**email** (*str*) – member email

**Returns**

dict that represent the user

**Return type**

*dict*

**to\_json()**

Returns platform \_json format of object

**Returns**

platform json format of object

**Return type**

*dict*

**update(system\_metadata=False)**

Update the project

**Parameters**

**system\_metadata** (*bool*) – to update system metadata

**Returns**

Project object

**Return type**

*dtlpy.entities.project.Project*

**update\_member(email, role: MemberRole = MemberRole.DEVELOPER)**

Update member's information/details from the project.

**Parameters**

- **email** (*str*) – member email
- **role** – `dl.MemberRole.OWNER`, `dl.MemberRole.DEVELOPER`, `dl.MemberRole.ANNOTATOR`, `dl.MemberRole.ANNOTATION_MANAGER`

**Returns**

dict that represent the user

**Return type**

*dict*

### 3.2.1 User

**class User**(*created\_at, updated\_at, name, last\_name, username, avatar, email, role, type, org, id, project, client\_api=None, users=None*)

Bases: BaseEntity

User entity

**classmethod from\_json**(*\_json, project, client\_api, users=None*)

Build a User entity object from a json

**Parameters**

- **\_json** (*dict*) – \_json response from host
- **project** (*dtlpy.entities.project.Project*) – project entity
- **client\_api** – ApiClient entity
- **users** – Users repository

**Returns**

User object

**Return type**

*dtlpy.entities.user.User*

**to\_json**()

Returns platform \_json format of object

**Returns**

platform json format of object

**Return type**

*dict*

## 3.3 Dataset

**class Dataset**(*id, url, name, annotated, creator, projects, items\_count, metadata, directoryTree, export, expiration\_options, index\_driver, created\_at, items\_url, readable\_type, access\_level, driver, readonly, client\_api: ApiClient, project=None, datasets=None, repositories=NOTHING, ontology\_ids=None, labels=None, directory\_tree=None, recipe=None, ontology=None*)

Bases: BaseEntity

Dataset object

**add\_label**(*label\_name, color=None, children=None, attributes=None, display\_label=None, label=None, recipe\_id=None, ontology\_id=None, icon\_path=None*)

Add single label to dataset

**Prerequisites:** You must have a dataset with items that are related to the annotations. The relationship between the dataset and annotations is shown in the name. You must be in the role of an *owner* or *developer*.

**Parameters**

- **label\_name** (*str*) – str - label name
- **color** (*tuple*) – color
- **children** – children (sub labels)

- **attributes** (*list*) – attributes
- **display\_label** (*str*) – display\_label
- **label** (*dtlpy.entities.label.Label*) – label
- **recipe\_id** (*str*) – optional recipe id
- **ontology\_id** (*str*) – optional ontology id
- **icon\_path** (*str*) – path to image to be display on label

**Returns**

label entity

**Return type**

dtlpy.entities.label.Label

**Example:**

```
dataset.add_label(label_name='person', color=(34, 6, 231), attributes=['big',  
↪ 'small'])
```

**add\_labels**(*label\_list*, *ontology\_id=None*, *recipe\_id=None*)

Add labels to dataset

**Prerequisites:** You must have a dataset with items that are related to the annotations. The relationship between the dataset and annotations is shown in the name. You must be in the role of an *owner* or *developer*.

**Parameters**

- **label\_list** (*list*) – label list
- **ontology\_id** (*str*) – optional ontology id
- **recipe\_id** (*str*) – optional recipe id

**Returns**

label entities

**Example:**

```
dataset.add_labels(label_list=label_list)
```

**checkout**()

Checkout the dataset

**clone**(*clone\_name*, *filters=None*, *with\_items\_annotations=True*, *with\_metadata=True*,  
*with\_task\_annotations\_status=True*)

Clone dataset

**Prerequisites:** You must be in the role of an *owner* or *developer*.

**Parameters**

- **clone\_name** (*str*) – new dataset name
- **filters** (*dtlpy.entities.filters.Filters*) – Filters entity or a query dict
- **with\_items\_annotations** (*bool*) – clone all item's annotations
- **with\_metadata** (*bool*) – clone metadata
- **with\_task\_annotations\_status** (*bool*) – clone task annotations status



**Returns**

dataset object

**Return type***dtlpy.entities.dataset.Dataset***Example:**

```
dataset.clone(dataset_id='dataset_id',
              clone_name='dataset_clone_name',
              with_metadata=True,
              with_items_annotations=False,
              with_task_annotations_status=False)
```

**delete**(*sure=False, really=False*)

Delete a dataset forever!

**Prerequisites:** You must be an *owner* or *developer* to use this method.**Parameters**

- **sure** (*bool*) – are you sure you want to delete?
- **really** (*bool*) – really really?

**Returns**

True is success

**Return type***bool***Example:**

```
dataset.delete(sure=True, really=True)
```

**delete\_attributes**(*keys: list, recipe\_id: Optional[str] = None, ontology\_id: Optional[str] = None*)

Delete a bulk of attributes

**Parameters**

- **recipe\_id** (*str*) – recipe id
- **ontology\_id** (*str*) – ontology id
- **keys** (*list*) – Keys of attributes to delete

**Returns**

True if success

**Return type***bool***delete\_labels**(*label\_names*)

Delete labels from dataset's ontologies

**Prerequisites:** You must be in the role of an *owner* or *developer*.**Parameters****label\_names** – label object/ label name / list of label objects / list of label names**Example:**

```
dataset.delete_labels(label_names=['myLabel1', 'Mylabel2'])
```

**download**(*filters=None, local\_path=None, file\_types=None, annotation\_options: Optional[ViewAnnotationOptions] = None, annotation\_filters=None, overwrite=False, to\_items\_folder=True, thickness=1, with\_text=False, without\_relative\_path=None, alpha=1, export\_version=ExportVersion.V1*)

Download dataset by filters. Filtering the dataset for items and save them local Optional - also download annotation, mask, instance and image mask of the item

**Prerequisites:** You must be in the role of an *owner* or *developer*.

#### Parameters

- **filters** (`dtlpy.entities.filters.Filters`) – Filters entity or a dictionary containing filters parameters
- **local\_path** (*str*) – local folder or filename to save to.
- **file\_types** (*list*) – a list of file type to download. e.g ['video/webm', 'video/mp4', 'image/jpeg', 'image/png']
- **annotation\_options** (*list*(`dtlpy.entities.annotation.ViewAnnotationOptions`)) – download annotations options: list(`dl.ViewAnnotationOptions`) not relevant for JSON option
- **annotation\_filters** (`dtlpy.entities.filters.Filters`) – Filters entity to filter annotations for download not relevant for JSON option
- **overwrite** (*bool*) – optional - default = False
- **to\_items\_folder** (*bool*) – Create 'items' folder and download items to it
- **thickness** (*int*) – optional - line thickness, if -1 annotation will be filled, default = 1
- **with\_text** (*bool*) – optional - add text to annotations, default = False
- **without\_relative\_path** (*bool*) – bool - download items without the relative path from platform
- **alpha** (*float*) – opacity value [0 1], default 1
- **export\_version** (*str*) – V2 - exported items will have original extension in filename, V1 - no original extension in filenames

#### Returns

List of local\_path per each downloaded item

#### Example:

```
dataset.download(local_path='local_path',
                  annotation_options=[dl.ViewAnnotationOptions.JSON, dl.
↳ ViewAnnotationOptions.MASK],
                  overwrite=False,
                  thickness=1,
                  with_text=False,
                  alpha=1,
                  save_locally=True
                  )
```

**download\_annotations**(*local\_path=None, filters=None, annotation\_options: Optional[ViewAnnotationOptions] = None, annotation\_filters=None, overwrite=False, thickness=1, with\_text=False, remote\_path=None, include\_annotations\_in\_output=True, export\_png\_files=False, filter\_output\_annotations=False, alpha=1, export\_version=ExportVersion.V1*)

Download dataset by filters. Filtering the dataset for items and save them local Optional - also download annotation, mask, instance and image mask of the item

**Prerequisites:** You must be in the role of an *owner* or *developer*.

#### Parameters

- **local\_path** (*str*) – local folder or filename to save to.
- **filters** (`dtlpy.entities.filters.Filters`) – Filters entity or a dictionary containing filters parameters
- **annotation\_options** (`(list(dtlpy.entities.annotation.ViewAnnotationOptions))`) – download annotations options: `list(dl.ViewAnnotationOptions)`
- **annotation\_filters** (`dtlpy.entities.filters.Filters`) – Filters entity to filter annotations for download
- **overwrite** (*bool*) – optional - default = False
- **thickness** (*int*) – optional - line thickness, if -1 annotation will be filled, default = 1
- **with\_text** (*bool*) – optional - add text to annotations, default = False
- **remote\_path** (*str*) – DEPRECATED and ignored
- **include\_annotations\_in\_output** (*bool*) – default - False , if export should contain annotations
- **export\_png\_files** (*bool*) – default - if True, semantic annotations should be exported as png files
- **filter\_output\_annotations** (*bool*) – default - False, given an export by filter - determine if to filter out annotations
- **alpha** (*float*) – opacity value [0 1], default 1
- **export\_version** (*str*) – exported items will have original extension in filename, *V1* - no original extension in filenames

#### Returns

local\_path of the directory where all the downloaded item

#### Return type

*str*

#### Example:

```
dataset.download_annotations(dataset='dataset_entity',
                             local_path='local_path',
                             annotation_options=[dl.ViewAnnotationOptions.JSON,
                             ↪dl.ViewAnnotationOptions.MASK],
                             overwrite=False,
                             thickness=1,
                             with_text=False,
```

(continues on next page)

(continued from previous page)

```
        alpha=1
    )
```

**download\_partition**(*partition*, *local\_path*=None, *filters*=None, *annotation\_options*=None)

Download a specific partition of the dataset to *local\_path* This function is commonly used with `dl.ModelAdapter` which implements the convert to specific model structure

**Parameters**

- **partition** (`dl.SnapshotPartitionType`) – `dl.SnapshotPartitionType` name of the partition
- **local\_path** (`str`) – local path directory to download the data
- **filters** (`dtlpy.entities.filters.Filters`) – `dl.entities.Filters` to add the specific partitions constraint to

:return List *str* of the new downloaded path of each item

**classmethod from\_json**(*project*: `Project`, *\_json*: `dict`, *client\_api*: `ApiClient`, *datasets*=None, *is\_fetched*=True)

Build a Dataset entity object from a json

**Parameters**

- **project** – dataset's project
- **\_json** (`dict`) – \_json response from host
- **client\_api** – `ApiClient` entity
- **datasets** – Datasets repository
- **is\_fetched** (`bool`) – is Entity fetched from Platform

**Returns**

Dataset object

**Return type**

`dtlpy.entities.dataset.Dataset`

**get\_partitions**(*partitions*, *filters*=None, *batch\_size*: `Optional[int]` = None)

Returns PagedEntity of items from one or more partitions

**Parameters**

- **partitions** – `dl.entities.SnapshotPartitionType` or a list. Name of the partitions
- **filters** (`dtlpy.entities.filters.Filters`) – `dl.Filters` to add the specific partitions constraint to
- **batch\_size** – `int` how many items per page

**Returns**

`dl.PagedEntities` of `dl.Item` preforms items.list()

**get\_recipe\_ids**()

Get dataset recipe Ids

**Returns**

list of recipe ids

**Return type**

list

**open\_in\_web()**

Open the dataset in web platform

**static serialize\_labels(labels\_dict)**

Convert hex color format to rgb

**Parameters****labels\_dict** (*dict*) – dict of labels**Returns**

dict of converted labels

**set\_partition(partition, filters=None)**

Updates all items returned by filters in the dataset to specific partition

**Parameters**

- **partition** – *dl.entities.SnapshotPartitionType* to set to
- **filters** (*dtlpy.entities.filters.Filters*) – *dl.entities.Filters* to add the specific partitions constraint to

**Returns***dl.PagedEntities***set\_readonly(state: bool)**

Set dataset readonly mode

**Prerequisites:** You must be in the role of an *owner* or *developer*.**Parameters****state** (*bool*) – state**Example:**

```
dataset.set_readonly(state=True)
```

**switch\_recipe(recipe\_id=None, recipe=None)**

Switch the recipe that linked to the dataset with the given one

**Parameters**

- **recipe\_id** (*str*) – recipe id
- **recipe** (*dtlpy.entities.recipe.Recipe*) – recipe entity

**Example:**

```
dataset.switch_recipe(recipe_id='recipe_id')
```

**sync(wait=True)**

Sync dataset with external storage

**Prerequisites:** You must be in the role of an *owner* or *developer*.**Parameters****wait** (*bool*) – wait for the command to finish**Returns**

True if success

**Return type**`bool`**Example:**

```
dataset.sync()
```

**to\_json()**

Returns platform \_json format of object

**Returns**

platform json format of object

**Return type**`dict`**update(system\_metadata=False)**

Update dataset field

**Prerequisites:** You must be an *owner* or *developer* to use this method.

**Parameters**

**system\_metadata** (`bool`) – bool - True, if you want to change metadata system

**Returns**

Dataset object

**Return type**`dtlpy.entities.dataset.Dataset`**Example:**

```
dataset.update()
```

**update\_attributes**(title: `str`, key: `str`, attribute\_type, recipe\_id: `Optional[str]` = None, ontology\_id: `Optional[str]` = None, scope: `Optional[list]` = None, optional: `Optional[bool]` = None, multi: `Optional[bool]` = None, values: `Optional[list]` = None, attribute\_range=None)

ADD a new attribute or update if exist

**Parameters**

- **ontology\_id** (`str`) – ontology\_id
- **title** (`str`) – attribute title
- **key** (`str`) – the key of the attribute must be unique
- **attribute\_type** (`AttributesTypes`) – dl.AttributesTypes your attribute type
- **scope** (`list`) – list of the labels or \* for all labels
- **optional** (`bool`) – optional attribute
- **multi** (`bool`) – if can get multiple selection
- **values** (`list`) – list of the attribute values ( for checkbox and radio button)
- **attribute\_range** (`dict` or `AttributesRange`) – dl.AttributesRange object

**Returns**

true in success

**Return type**`bool`

**Example:**

```
dataset.update_attributes(ontology_id='ontology_id',
                        key='1',
                        title='checkbox',
                        attribute_type=dl.AttributesTypes.CHECKBOX,
                        values=[1,2,3])
```

**update\_label**(*label\_name*, *color=None*, *children=None*, *attributes=None*, *display\_label=None*, *label=None*, *recipe\_id=None*, *ontology\_id=None*, *upsert=False*, *icon\_path=None*)

Add single label to dataset

**Prerequisites:** You must have a dataset with items that are related to the annotations. The relationship between the dataset and annotations is shown in the name. You must be in the role of an *owner* or *developer*.

**Parameters**

- **label\_name** (*str*) – str - label name
- **color** (*tuple*) – color
- **children** – children (sub labels)
- **attributes** (*list*) – attributes
- **display\_label** (*str*) – display\_label
- **label** (*dtlpy.entities.label.Label*) – label
- **recipe\_id** (*str*) – optional recipe id
- **ontology\_id** (*str*) – optional ontology id
- **icon\_path** (*str*) – path to image to be display on label

**Returns**

label entity

**Return type**

dtlpy.entities.label.Label

**Example:**

```
dataset.update_label(label_name='person', color=(34, 6, 231), attributes=['big',
↪ 'small'])
```

**update\_labels**(*label\_list*, *ontology\_id=None*, *recipe\_id=None*, *upsert=False*)

Add labels to dataset

**Prerequisites:** You must have a dataset with items that are related to the annotations. The relationship between the dataset and annotations is shown in the name. You must be in the role of an *owner* or *developer*.

**Parameters**

- **label\_list** (*list*) – label list
- **ontology\_id** (*str*) – optional ontology id
- **recipe\_id** (*str*) – optional recipe id
- **upsert** (*bool*) – if True will add in case it does not existing

**Returns**

label entities

**Return type**

dtlpy.entities.label.Label

**Example:**

```
dataset.update_labels(label_list=label_list)
```

**upload\_annotations**(*local\_path*, *filters=None*, *clean=False*, *remote\_root\_path='/'*,  
*export\_version=ExportVersion.V1*)

Upload annotations to dataset.

**Prerequisites:** You must have a dataset with items that are related to the annotations. The relationship between the dataset and annotations is shown in the name. You must be in the role of an *owner* or *developer*.

**Parameters**

- **local\_path** (*str*) – str - local folder where the annotations files is.
- **filters** (*dtlpy.entities.filters.Filters*) – Filters entity or a dictionary containing filters parameters
- **clean** (*bool*) – bool - if True it remove the old annotations
- **remote\_root\_path** (*str*) – str - the remote root path to match remote and local items
- **export\_version** (*str*) – V2 - exported items will have original extension in filename, V1 - no original extension in filenames

For example, if the item filepath is a/b/item and remote\_root\_path is /a the start folder will be b instead of a

**Example:**

```
dataset.upload_annotations(dataset='dataset_entity',  
                           local_path='local_path',  
                           clean=False,  
                           export_version=dl.ExportVersion.V1  
                           )
```

**class ExpirationOptions**(*item\_max\_days: Optional[int] = None*)

Bases: `object`

ExpirationOptions object

**class IndexDriver**(*value*)

Bases: `str`, `Enum`

An enumeration.

### 3.3.1 Driver

**class Driver**(*bucket\_name*, *creator*, *allow\_external\_delete*, *allow\_external\_modification*, *created\_at*, *region*,  
*path*, *type*, *integration\_id*, *metadata*, *name*, *id*, *client\_api: ApiClient*)

Bases: `BaseEntity`

Driver entity

**classmethod from\_json**(*\_json*, *client\_api*, *is\_fetched=True*)

Build a Driver entity object from a json

**Parameters**



- **\_json** – \_json response from host
- **client\_api** – ApiClient entity
- **is\_fetched** – is Entity fetched from Platform

**Returns**

Driver object

**to\_json()**

Returns platform \_json format of object

**Returns**

platform json format of object

**Return type**

dict

**class ExternalStorage**(*value*)

Bases: `str`, `Enum`

An enumeration.

## 3.4 Item

**class ExportMetadata**(*value*)

Bases: `Enum`

An enumeration.

**class Item**(*annotations\_link, dataset\_url, thumbnail, created\_at, dataset\_id, annotated, metadata, filename, stream, name, type, url, id, hidden, dir, spec, creator, annotations\_count, client\_api: ApiClient, platform\_dict, dataset, project, project\_id, repositories=NOTHING*)

Bases: `BaseEntity`

Item object

**clone**(*dst\_dataset\_id=None, remote\_filepath=None, metadata=None, with\_annotations=True, with\_metadata=True, with\_task\_annotations\_status=False, allow\_many=False, wait=True*)

Clone item

**Parameters**

- **dst\_dataset\_id** (*str*) – destination dataset id
- **remote\_filepath** (*str*) – complete filepath
- **metadata** (*dict*) – new metadata to add
- **with\_annotations** (*bool*) – clone annotations
- **with\_metadata** (*bool*) – clone metadata
- **with\_task\_annotations\_status** (*bool*) – clone task annotations status
- **allow\_many** (*bool*) – *bool* if True, using multiple clones in single dataset is allowed, (default=False)
- **wait** (*bool*) – wait for the command to finish

**Returns**

Item object

**Return type***dtlpy.entities.item.Item***Example:**

```
item.clone(item_id='item_id',
           dst_dataset_id='dist_dataset_id',
           with_metadata=True,
           with_task_annotations_status=False,
           with_annotations=False)
```

**delete()**

Delete item from platform

**Returns**

True

**Return type**

bool

**download**(*local\_path=None, file\_types=None, save\_locally=True, to\_array=False, annotation\_options: Optional[ViewAnnotationOptions] = None, overwrite=False, to\_items\_folder=True, thickness=1, with\_text=False, annotation\_filters=None, alpha=1, export\_version=ExportVersion.V1*)

Download dataset by filters. Filtering the dataset for items and save them local Optional - also download annotation, mask, instance and image mask of the item

**Parameters**

- **local\_path** (*str*) – local folder or filename to save to.
- **file\_types** (*list*) – a list of file type to download. e.g ['video/webm', 'video/mp4', 'image/jpeg', 'image/png']
- **save\_locally** (*bool*) – bool. save to disk or return a buffer
- **to\_array** (*bool*) – returns Narray when True and local\_path = False
- **annotation\_options** (*list*) – download annotations options: list(dl.ViewAnnotationOptions)
- **annotation\_filters** (*dtlpy.entities.filters.Filters*) – Filters entity to filter annotations for download
- **overwrite** (*bool*) – optional - default = False
- **to\_items\_folder** (*bool*) – Create 'items' folder and download items to it
- **thickness** (*int*) – optional - line thickness, if -1 annotation will be filled, default = 1
- **with\_text** (*bool*) – optional - add text to annotations, default = False
- **alpha** (*float*) – opacity value [0 1], default 1
- **export\_version** (*str*) – exported items will have original extension in filename, *V1* - no original extension in filenames

**Returns**

generator of local\_path per each downloaded item

**Return type**

generator or single item

**Example:**

```

item.download(local_path='local_path',
              annotation_options=dl.ViewAnnotationOptions.MASK,
              overwrite=False,
              thickness=1,
              with_text=False,
              alpha=1,
              save_locally=True
            )

```

**classmethod** `from_json(_json, client_api, dataset=None, project=None, is_fetched=True)`

Build an item entity object from a json

**Parameters**

- **project** (`dtlpy.entities.project.Project`) – project entity
- **\_json** (*dict*) – \_json response from host
- **dataset** (`dtlpy.entities.dataset.Dataset`) – dataset in which the annotation's item is located
- **.client\_api** (*dlApiClient*) – ApiClient entity
- **is\_fetched** (*bool*) – is Entity fetched from Platform

**Returns**

Item object

**Return type**

*dtlpy.entities.item.Item*

**move**(*new\_path*)

Move item from one folder to another in Platform If the directory doesn't exist it will be created

**Parameters**

**new\_path** (*str*) – new full path to move item to.

**Returns**

True if update successfully

**Return type**

*bool*

**open\_in\_web**()

Open the items in web platform

**Returns**

**set\_description**(*text: str*)

Update Item description

**Parameters**

**text** (*str*) – if None or "" description will be deleted

:return

**to\_json**()

Returns platform \_json format of object

**Returns**

platform json format of object

### Return type

dict

**update**(*system\_metadata=False*)

Update items metadata

### Parameters

**system\_metadata** (*bool*) – bool - True, if you want to change metadata system

### Returns

Item object

### Return type

*dtlpy.entities.item.Item*

**update\_status**(*status: str, clear: bool = False, assignment\_id: Optional[str] = None, task\_id: Optional[str] = None*)

update item status

### Parameters

- **status** (*str*) – “completed” ,”approved” ,”discard”
- **clear** (*bool*) – if true delete status
- **assignment\_id** (*str*) – assignment id
- **task\_id** (*str*) – task id

:return :True/False :rtype: bool

### Example:

```
item.update_status(status='complete',
                   operation='created',
                   task_id='task_id')
```

**class ItemStatus**(*value*)

Bases: *str*, *Enum*

An enumeration.

**class ModalityRefTypeEnum**(*value*)

Bases: *str*, *Enum*

State enum

**class ModalityTypeEnum**(*value*)

Bases: *str*, *Enum*

State enum

### 3.4.1 Item Link

**class** `LinkTypeEnum(value)`

Bases: `str`, `Enum`

State enum

## 3.5 Annotation

**class** `Annotation(id, url, item_url, item, item_id, creator, created_at, updated_by, updated_at, type, source, dataset_url, platform_dict, metadata, fps, hash=None, dataset_id=None, status=None, object_id=None, automated=None, item_height=None, item_width=None, label_suggestions=None, annotation_definition: Optional[BaseAnnotationDefinition] = None, frames=None, current_frame=0, end_frame=0, end_time=0, start_frame=0, start_time=0, dataset=None, datasets=None, annotations=None, Annotation__client_api=None, items=None, recipe_2_attributes=None)`

Bases: `BaseEntity`

Annotations object

**add\_frame**(*annotation\_definition*, *frame\_num*=None, *fixed*=True, *object\_visible*=True)

Add a frame state to annotation

#### Parameters

- **annotation\_definition** – annotation type object - must be same type as annotation
- **frame\_num** (*int*) – frame number
- **fixed** (*bool*) – is fixed
- **object\_visible** (*bool*) – does the annotated object is visible

#### Returns

True if success

#### Return type

`bool`

#### Example:

```
annotation.add_frame(frame_num=10,
                     annotation_definition=dl.Box(top=10, left=10, bottom=100,
↪right=100, label='labelName'))
                     )
```

**add\_frames**(*annotation\_definition*, *frame\_num*=None, *end\_frame\_num*=None, *start\_time*=None, *end\_time*=None, *fixed*=True, *object\_visible*=True)

Add a frames state to annotation

**Prerequisites:** Any user can upload annotations.

#### Parameters

- **annotation\_definition** – annotation type object - must be same type as annotation
- **frame\_num** (*int*) – first frame number
- **end\_frame\_num** (*int*) – last frame number

- **start\_time** – starting time for video
- **end\_time** – ending time for video
- **fixed** (*bool*) – is fixed
- **object\_visible** (*bool*) – does the annotated object is visible

**Returns****Example:**

```
annotation.add_frames(frame_num=10,  
                      annotation_definition=dl.Box(top=10,left=10,bottom=100,↵  
↵right=100,label='labelName'))  
)
```

**delete()**

Remove an annotation from item

**Prerequisites:** You must have an item that has been annotated. You must have the role of an *owner* or *developer* or be assigned a task that includes that item as an *annotation manager* or *annotator*.

**Returns**

True if success

**Return type**

*bool*

**Example:**

```
annotation.delete()
```

**download**(filepath: *str*, annotation\_format: *ViewAnnotationOptions* = *ViewAnnotationOptions.MASK*,  
height: *Optional[float]* = *None*, width: *Optional[float]* = *None*, thickness: *int* = 1, with\_text: *bool*  
= *False*, alpha: *float* = 1)

Save annotation to file

**Prerequisites:** Any user can upload annotations.

**Parameters**

- **filepath** (*str*) – local path to where annotation will be downloaded to
- **annotation\_format** (*list*) – options: list(dl.ViewAnnotationOptions)
- **height** (*float*) – image height
- **width** (*float*) – image width
- **thickness** (*int*) – thickness
- **with\_text** (*bool*) – get mask with text
- **alpha** (*float*) – opacity value [0 1], default 1

**Returns**

filepath

**Return type**

*str*

**Example:**

```
annotation.download(filepath='filepath', annotation_format=dl.  
↳ViewAnnotationOptions.MASK)
```

**classmethod** `from_json(_json, item=None, client_api=None, annotations=None, is_video=None, fps=None, item_metadata=None, dataset=None, is_audio=None)`

Create an annotation object from platform json

#### Parameters

- `_json` (*dict*) – platform json
- `item` (`dtlpy.entities.item.Item`) – item
- `client_api` – ApiClient entity
- `annotations` –
- `is_video` (*bool*) – is video
- `fps` – video fps
- `item_metadata` – item metadata
- `dataset` – dataset entity
- `is_audio` (*bool*) – is audio

#### Returns

annotation object

#### Return type

`dtlpy.entities.annotation.Annotation`

**classmethod** `new(item=None, annotation_definition=None, object_id=None, automated=True, metadata=None, frame_num=None, parent_id=None, start_time=None, item_height=None, item_width=None)`

Create a new annotation object annotations

**Prerequisites:** Any user can upload annotations.

#### Parameters

- `item` (`dtlpy.entities.item.Items`) – item to annotate
- `annotation_definition` – annotation type object
- `object_id` (*str*) – object\_id
- `automated` (*bool*) – is automated
- `metadata` (*dict*) – metadata
- `frame_num` (*int*) – optional - first frame number if video annotation
- `parent_id` (*str*) – add parent annotation ID
- `start_time` – optional - start time if video annotation
- `item_height` (*float*) – annotation item's height
- `item_width` (*float*) – annotation item's width

#### Returns

annotation object

**Return type***dtlpy.entities.annotation.Annotation***Example:**

```
annotation.new(item='item_entity',
               annotation_definition=dl.Box(top=10,left=10,bottom=100,
               ↪right=100,label='labelName'))
               )
```

**set\_frame**(*frame*)

Set annotation to frame state

**Prerequisites:** Any user can upload annotations.**Parameters****frame** (*int*) – frame number**Returns**

True if success

**Return type***bool***Example:**

```
annotation.set_frame(frame=10)
```

**show**(*image=None, thickness=None, with\_text=False, height=None, width=None, annotation\_format: ViewAnnotationOptions = ViewAnnotationOptions.MASK, color=None, label\_instance\_dict=None, alpha=1, frame\_num=None*)

Show annotations mark the annotation of the image array and return it

**Prerequisites:** Any user can upload annotations.**Parameters**

- **image** – empty or image to draw on
- **thickness** (*int*) – line thickness
- **with\_text** (*bool*) – add label to annotation
- **height** (*float*) – height
- **width** (*float*) – width
- **annotation\_format** (*dl.ViewAnnotationOptions*) –  
list(dl.ViewAnnotationOptions)
- **color** (*tuple*) – optional - color tuple
- **label\_instance\_dict** – the instance labels
- **alpha** (*float*) – opacity value [0 1], default 1
- **frame\_num** (*int*) – for video annotation, show specific fame

**Returns**

list or single ndarray of the annotations

**Examples:**



```

annotation.show(image='ndarray',
                 thickness=1,
                 annotation_format=dl.VIEW_ANNOTATION_OPTIONS_MASK,
                 )

```

**to\_json()**

Convert annotation object to a platform json representation

**Returns**

platform json

**Return type**

dict

**update(system\_metadata=False)**

Update an existing annotation in host.

**Prerequisites:** You must have an item that has been annotated. You must have the role of an *owner* or *developer* or be assigned a task that includes that item as an *annotation manager* or *annotator*.

**Parameters**

**system\_metadata** – True, if you want to change metadata system

**Returns**

Annotation object

**Return type**

*dtlpy.entities.annotation.Annotation*

**Example:**

```

annotation.update()

```

**update\_status(status: AnnotationStatus = AnnotationStatus.ISSUE)**

Set status on annotation

**Prerequisites:** You must have an item that has been annotated. You must have the role of an *owner* or *developer* or be assigned a task that includes that item as an *annotation manager*.

**Parameters**

**status** (*str*) – can be AnnotationStatus.ISSUE, AnnotationStatus.APPROVED, AnnotationStatus.REVIEW, AnnotationStatus.CLEAR

**Returns**

Annotation object

**Return type**

*dtlpy.entities.annotation.Annotation*

**Example:**

```

annotation.update_status(status=dl.AnnotationStatus.ISSUE)

```

**upload()**

Create a new annotation in host

**Prerequisites:** Any user can upload annotations.

**Returns**

Annotation entity

**Return type***dtlpy.entities.annotation.Annotation***class AnnotationStatus**(*value*)Bases: `str`, `Enum`

An enumeration.

**class AnnotationType**(*value*)Bases: `str`, `Enum`

An enumeration.

**class ExportVersion**(*value*)Bases: `str`, `Enum`

An enumeration.

**class FrameAnnotation**(*annotation*, *annotation\_definition*, *frame\_num*, *fixed*, *object\_visible*,  
*recipe\_2\_attributes*=None, *interpolation*=False)Bases: `BaseEntity`

FrameAnnotation object

**classmethod from\_snapshot**(*annotation*, *\_json*, *fps*)

new frame state to annotation

**Parameters**

- **annotation** – annotation
- **\_json** – annotation type object - must be same type as annotation
- **fps** – frame number

**Returns**

FrameAnnotation object

**classmethod new**(*annotation*, *annotation\_definition*, *frame\_num*, *fixed*, *object\_visible*=True)

new frame state to annotation

**Parameters**

- **annotation** – annotation
- **annotation\_definition** – annotation type object - must be same type as annotation
- **frame\_num** – frame number
- **fixed** – is fixed
- **object\_visible** – does the annotated object is visible

**Returns**

FrameAnnotation object

**show**(*\*\*kwargs*)

Show annotation as ndarray :param kwargs: see annotation definition :return: ndarray of the annotation

**class ViewAnnotationOptions**(*value*)Bases: `str`, `Enum`

The Annotations file types to download (JSON, MASK, INSTANCE, ANNOTATION\_ON\_IMAGE, VTT, OBJECT\_ID).

State	Description
JSON	Dataloop json format
MASK	PNG file that contains drawing annotations on it
IN-STANCE	An image file that contains 2D annotations
AN-NO-TA-TION_ON_IMAGE	The source image with the annotations drawing in it
VTT	An text file contains supplementary information about a web video
OB-JECT_ID	An image file that contains 2D annotations

### 3.5.1 Collection of Annotation entities

**class AnnotationCollection**(*item=None, annotations=NOTHING, dataset=None, colors=None*)

Bases: BaseEntity

Collection of Annotation entity

**add**(*annotation\_definition, object\_id=None, frame\_num=None, end\_frame\_num=None, start\_time=None, end\_time=None, automated=True, fixed=True, object\_visible=True, metadata=None, parent\_id=None, model\_info=None*)

Add annotations to collection

#### Parameters

- **annotation\_definition** – dl.Polygon, dl.Segmentation, dl.Point, dl.Box etc
- **object\_id** – Object id (any id given by user). If video - must input to match annotations between frames
- **frame\_num** – video only, number of frame
- **end\_frame\_num** – video only, the end frame of the annotation
- **start\_time** – video only, start time of the annotation
- **end\_time** – video only, end time of the annotation
- **automated** –
- **fixed** – video only, mark frame as fixed
- **object\_visible** – video only, does the annotated object is visible
- **metadata** – optional- metadata dictionary for annotation
- **parent\_id** – set a parent for this annotation (parent annotation ID)
- **model\_info** – optional - set model on annotation { 'name',:', 'confidence':0}

#### Returns

**delete()**

Remove an annotation from item

**Prerequisites:** You must have an item that has been annotated. You must have the role of an *owner* or *developer* or be assigned a task that includes that item as an *annotation manager* or *annotator*.

**Returns**

True if success

**Return type**

bool

**Example:**

```
builder.delete()
```

**download**(filepath, img\_filepath=None, annotation\_format: `ViewAnnotationOptions` = `ViewAnnotationOptions.MASK`, height=None, width=None, thickness=1, with\_text=False, orientation=0, alpha=1)

Save annotations to file

**Prerequisites:** Any user can upload annotations.

**Parameters**

- **filepath** (*str*) – path to save annotation
- **img\_filepath** (*str*) – img file path - needed for img\_mask
- **annotation\_format** (`dl.ViewAnnotationOptions`) – how to show thw annotations.  
options: list(`dl.ViewAnnotationOptions`)
- **height** (*int*) – height
- **width** (*int*) – width
- **thickness** (*int*) – thickness
- **with\_text** (*bool*) – add a text to the image
- **orientation** (*int*) – the image orientation
- **alpha** (*float*) – opacity value [0 1], default 1

**Returns**

file path of the downlaod annotation

**Return type**

str

**Example:**

```
builder.download(filepath='filepath', annotation_format=dl.  
↳ViewAnnotationOptions.MASK)
```

**from\_instance\_mask**(mask, instance\_map=None)

convert annotation from instance mask format

**Parameters**

- **mask** – the mask annotation
- **instance\_map** – labels

**classmethod from\_json**(\_json: *list*, item=None, is\_video=None, fps=25, height=None, width=None, client\_api=None, is\_audio=None)

Create an annotation collection object from platform json

**Parameters**

- **\_json** (*dict*) – platform json
- **item** (*dtlpy.entities.item.Item*) – item
- **client\_api** – ApiClient entity
- **is\_video** (*bool*) – is video
- **fps** – video fps
- **height** (*float*) – height
- **width** (*float*) – width
- **is\_audio** (*bool*) – is audio

**Returns**

annotation object

**Return type**

*dtlpy.entities.annotation.Annotation*

**from\_vtt\_file**(*filepath*)

convert annotation from vtt format

**Parameters**

**filepath** (*str*) – path to the file

**get\_frame**(*frame\_num*)

Get frame

**Parameters**

**frame\_num** (*int*) – frame num

**Returns**

AnnotationCollection

**print**(*to\_return=False, columns=None*)

**Parameters**

- **to\_return** –
- **columns** –

**show**(*image=None, thickness=None, with\_text=False, height=None, width=None, annotation\_format: ViewAnnotationOptions = ViewAnnotationOptions.MASK, label\_instance\_dict=None, color=None, alpha=1, frame\_num=None*)

Show annotations according to annotation\_format

**Prerequisites:** Any user can upload annotations.

**Parameters**

- **image** (*ndarray*) – empty or image to draw on
- **height** (*int*) – height
- **width** (*int*) – width
- **thickness** (*int*) – line thickness
- **with\_text** (*bool*) – add label to annotation
- **annotation\_format** (*dl.ViewAnnotationOptions*) – how to show thw annotations.  
options: list(*dl.ViewAnnotationOptions*)

- **label\_instance\_dict** (*dict*) – instance label map {'Label': 1, 'More': 2}
- **color** (*tuple*) – optional - color tuple
- **alpha** (*float*) – opacity value [0 1], default 1
- **frame\_num** (*int*) – for video annotation, show specific frame

**Returns**

ndarray of the annotations

**Example:**

```
builder.show(image='ndarray',
             thickness=1,
             annotation_format=dl.VIEW_ANNOTATION_OPTIONS_MASK,
             )
```

**to\_json()**

Convert annotation object to a platform json representation

**Returns**

platform json

**Return type**

*dict*

**update(system\_metadata=True)**

Update an existing annotation in host.

**Prerequisites:** You must have an item that has been annotated. You must have the role of an *owner* or *developer* or be assigned a task that includes that item as an *annotation manager* or *annotator*.

**Parameters**

**system\_metadata** – True, if you want to change metadata system

**Returns**

Annotation object

**Return type**

*dtlpy.entities.annotation.Annotation*

**Example:**

```
builder.update()
```

**upload()**

Create a new annotation in host

**Prerequisites:** Any user can upload annotations.

**Returns**

Annotation entity

**Return type**

*dtlpy.entities.annotation.Annotation*

**Example:**

```
builder.upload()
```

### 3.5.2 Annotation Definition

#### Box Annotation Definition

**class** **Box**(*left=None, top=None, right=None, bottom=None, label=None, attributes=None, description=None, angle=None*)

Bases: BaseAnnotationDefinition

Box annotation object Can create a box using 2 point using: “top”, “left”, “bottom”, “right” (to form a box [(left, top), (right, bottom)]) For rotated box add the “angel”

**classmethod** **from\_segmentation**(*mask, label, attributes=None*)

Convert binary mask to Polygon

#### Parameters

- **mask** – binary mask (0,1)
- **label** – annotation label
- **attributes** – annotations list of attributes

#### Returns

Box annotations list to each separated segmentation

**show**(*image, thickness, with\_text, height, width, annotation\_format, color, alpha=1*)

Show annotation as ndarray :param image: empty or image to draw on :param thickness: :param with\_text: not required :param height: item height :param width: item width :param annotation\_format: options: list(dl.ViewAnnotationOptions) :param color: color :param alpha: opacity value [0 1], default 1 :return: ndarray

#### Classification Annotation Definition

**class** **Classification**(*label, attributes=None, description=None*)

Bases: BaseAnnotationDefinition

Classification annotation object

**show**(*image, thickness, with\_text, height, width, annotation\_format, color, alpha=1*)

Show annotation as ndarray :param image: empty or image to draw on :param thickness: :param with\_text: not required :param height: item height :param width: item width :param annotation\_format: options: list(dl.ViewAnnotationOptions) :param color: color :param alpha: opacity value [0 1], default 1 :return: ndarray

#### Cuboid Annotation Definition

**class** **Cube**(*label, front\_tl, front\_tr, front\_br, front\_bl, back\_tl, back\_tr, back\_br, back\_bl, angle=None, attributes=None, description=None*)

Bases: BaseAnnotationDefinition

Cube annotation object

**classmethod** **from\_boxes\_and\_angle**(*front\_left, front\_top, front\_right, front\_bottom, back\_left, back\_top, back\_right, back\_bottom, label, angle=0, attributes=None*)

Create cuboid by given front and back boxes with angle the angle calculate fom the center of each box

**show**(*image, thickness, with\_text, height, width, annotation\_format, color, alpha=1*)

Show annotation as ndarray :param image: empty or image to draw on :param thickness: :param with\_text: not required :param height: item height :param width: item width :param annotation\_format: options: list(dl.ViewAnnotationOptions) :param color: color :param alpha: opacity value [0 1], default 1 :return: ndarray

### Item Description Definition

**class Description**(*text, description=None*)

Bases: BaseAnnotationDefinition

Subtitle annotation object

### Ellipse Annotation Definition

**class Ellipse**(*x, y, rx, ry, angle, label, attributes=None, description=None*)

Bases: BaseAnnotationDefinition

Ellipse annotation object

**show**(*image, thickness, with\_text, height, width, annotation\_format, color, alpha=1*)

Show annotation as ndarray :param image: empty or image to draw on :param thickness: :param with\_text: not required :param height: item height :param width: item width :param annotation\_format: options: list(dl.ViewAnnotationOptions) :param color: color :param alpha: opacity value [0 1], default 1 :return: ndarray

### Note Annotation Definition

**class Message**(*msg\_id: Optional[str] = None, creator: Optional[str] = None, msg\_time=None, body: Optional[str] = None*)

Bases: object

Note message object

**class Note**(*left, top, right, bottom, label, attributes=None, messages=None, status='issue', assignee=None, create\_time=None, creator=None, description=None*)

Bases: Box

Note annotation object

### Point Annotation Definition

**class Point**(*x, y, label, attributes=None, description=None*)

Bases: BaseAnnotationDefinition

Point annotation object

**show**(*image, thickness, with\_text, height, width, annotation\_format, color, alpha=1*)

Show annotation as ndarray :param image: empty or image to draw on :param thickness: :param with\_text: not required :param height: item height :param width: item width :param annotation\_format: options: list(dl.ViewAnnotationOptions) :param color: color :param alpha: opacity value [0 1], default 1 :return: ndarray



## Polygon Annotation Definition

**class Polygon**(*geo, label, attributes=None, description=None*)

Bases: BaseAnnotationDefinition

Polygon annotation object

**classmethod from\_segmentation**(*mask, label, attributes=None, epsilon=None, max\_instances=1, min\_area=0*)

Convert binary mask to Polygon

### Parameters

- **mask** – binary mask (0,1)
- **label** – annotation label
- **attributes** – annotations list of attributes
- **epsilon** – from opencv: specifying the approximation accuracy. This is the maximum distance between the original curve and its approximation. if 0 all points are returns
- **max\_instances** – number of max instances to return. if None all wil be returned
- **min\_area** – remove polygons with area lower thn this threshold (pixels)

### Returns

Polygon annotation

**show**(*image, thickness, with\_text, height, width, annotation\_format, color, alpha=1*)

Show annotation as ndarray :param image: empty or image to draw on :param thickness: :param with\_text: not required :param height: item height :param width: item width :param annotation\_format: options: list(dl.ViewAnnotationOptions) :param color: color :param alpha: opacity value [0 1], default 1 :return: ndarray

## Polyline Annotation Definition

**class Polyline**(*geo, label, attributes=None, description=None*)

Bases: BaseAnnotationDefinition

Polyline annotation object

**show**(*image, thickness, with\_text, height, width, annotation\_format, color, alpha=1*)

Show annotation as ndarray :param image: empty or image to draw on :param thickness: :param with\_text: not required :param height: item height :param width: item width :param annotation\_format: options: list(dl.ViewAnnotationOptions) :param color: color :param alpha: opacity value [0 1], default 1 :return: ndarray

## Pose Annotation Definition

**class Pose**(*label, template\_id, instance\_id=None, attributes=None, points=None, description=None*)

Bases: BaseAnnotationDefinition

Classification annotation object

**show**(*image, thickness, with\_text, height, width, annotation\_format, color, alpha=1*)

Show annotation as ndarray :param image: empty or image to draw on :param thickness: :param with\_text: not required :param height: item height :param width: item width :param annotation\_format: options: list(dl.ViewAnnotationOptions) :param color: color :param alpha: opacity value [0 1], default 1 :return: ndarray

## Segmentation Annotation Definition

**class Segmentation**(*geo, label, attributes=None, description=None*)

Bases: BaseAnnotationDefinition

Segmentation annotation object

**classmethod from\_polygon**(*geo, label, shape, attributes=None*)

### Parameters

- **geo** – list of x,y coordinates of the polygon ([[x,y],[x,y]...])
- **label** – annotation's label
- **shape** – image shape (h,w)
- **attributes** –

### Returns

**show**(*image, thickness, with\_text, height, width, annotation\_format, color, alpha=1*)

Show annotation as ndarray :param image: empty or image to draw on :param thickness: :param with\_text: not required :param height: item height :param width: item width :param annotation\_format: options: list(dl.ViewAnnotationOptions) :param color: color :param alpha: opacity value [0 1], default 1 :return: ndarray

**to\_box**()

### Returns

Box annotations list to each separated segmentation

## Audio Annotation Definition

**class Subtitle**(*text, label, attributes=None, description=None*)

Bases: BaseAnnotationDefinition

Subtitle annotation object

## Undefined Annotation Definition

**class UndefinedAnnotationType**(*type, label, coordinates, attributes=None, description=None*)

Bases: BaseAnnotationDefinition

UndefinedAnnotationType annotation object

**show**(*image, thickness, with\_text, height, width, annotation\_format, color, alpha=1*)

Show annotation as ndarray :param image: empty or image to draw on :param thickness: :param with\_text: not required :param height: item height :param width: item width :param annotation\_format: options: list(dl.ViewAnnotationOptions) :param color: color :param alpha: opacity value [0 1], default 1 :return: ndarray

### 3.5.3 Similarity

```
class Collection(type: CollectionTypes, name, items=None)
    Bases: object
    Base Collection Entity
    add(ref, type: SimilarityTypeEnum = SimilarityTypeEnum.ID)
        Add item to collection :param ref: :param type: url, id
    pop(ref)

        Parameters
        ref –

    to_json()
        Returns platform _json format of object

        Returns
        platform json format of object

        Return type
        dict

class CollectionItem(type: SimilarityTypeEnum, ref)
    Bases: object
    Base CollectionItem

class CollectionTypes(value)
    Bases: str, Enum
    An enumeration.

class MultiView(name, items=None)
    Bases: Collection
    Multi Entity
    property items
        list of the collection items
    to_json()
        Returns platform _json format of object

        Returns
        platform json format of object

        Return type
        dict

class MultiViewItem(type, ref)
    Bases: CollectionItem
    Single multi view item

class Similarity(ref, name=None, items=None)
    Bases: Collection
    Similarity Entity
```

**property items**

list of the collection items

**property target**

Target item for similarity

**to\_json()**

Returns platform \_json format of object

**Returns**

platform json format of object

**Return type**

dict

**class SimilarityItem**(*type, ref, target=False*)

Bases: [CollectionItem](#)

Single similarity item

**class SimilarityTypeEnum**(*value*)

Bases: [str](#), [Enum](#)

State enum

## 3.6 Filter

**class Filters**(*field=None, values=None, operator: Optional[FiltersOperations] = None, method: Optional[FiltersMethod] = None, custom\_filter=None, resource: FiltersResource = FiltersResource.ITEM, use\_defaults=True, context=None, page\_size=None*)

Bases: [object](#)

Filters entity to filter items from pages in platform

**add**(*field, values, operator: Optional[FiltersOperations] = None, method: Optional[FiltersMethod] = None*)

Add filter

**Parameters**

- **field** ([str](#)) – Metadata field / attribute
- **values** ([str](#) or [list](#)) – field values
- **operator** ([dl.FiltersOperations](#)) – optional - in, gt, lt, eq, ne
- **method** ([dl.FiltersMethod](#)) – Optional - or/and

**Example:**

```
filter.add(field='metadata.user', values=['1','2'], operator=dl.  
↳FiltersOperations.IN)
```

**add\_join**(*field, values, operator: Optional[FiltersOperations] = None, method: FiltersMethod = FiltersMethod.AND*)

join a query to the filter

**Parameters**

- **field** ([str](#)) – Metadata field / attribute

- **values** (*str* or *list*) – field values
- **operator** (*dl.FiltersOperations*) – optional - in, gt, lt, eq, ne
- **method** – optional - str - FiltersMethod.AND, FiltersMethod.OR

**Example:**

```
filter.add_join(field='metadata.user', values=['1', '2'], operator=dl.
↳FiltersOperations.IN)
```

**generate\_url\_query\_params**(*url*)

generate url query params

**Parameters**

**url** (*str*) –

**has\_field**(*field*)

is filter has field

**Parameters**

**field** (*str*) – field to check

**Returns**

Ture is have it

**Return type**

*bool*

**open\_in\_web**(*resource*)

Open the filter in the platform data browser (in a new web browser)

**Parameters**

**resource** (*str*) – dl entity to apply filter on. currently only supports dl.Dataset

**platform\_url**(*resource*) → *str*

Build a url with filters param to open in web browser

**Parameters**

**resource** (*str*) – dl entity to apply filter on. currently only supports dl.Dataset

**Returns**

url string

**Return type**

*str*

**pop**(*field*)

Pop filed

**Parameters**

**field** (*str*) – field to pop

**pop\_join**(*field*)

Pop join

**Parameters**

**field** (*str*) – field to pop

**prepare**(*operation=None, update=None, query\_only=False, system\_update=None, system\_metadata=False*)

To dictionary for platform call

**Parameters**

- **operation** (*str*) – operation
- **update** – update
- **query\_only** (*bool*) – query only
- **system\_update** – system update
- **system\_metadata** – True, if you want to change metadata system

**Returns**

dict of the filter

**Return type**

dict

**sort\_by**(*field*, *value*: `FiltersOrderByDirection` = `FiltersOrderByDirection.ASCENDING`)

sort the filter

**Parameters**

- **field** (*str*) – field to sort by it
- **value** (*dl.FiltersOrderByDirection*) – `FiltersOrderByDirection.ASCENDING`, `FiltersOrderByDirection.DESENDING`

**Example:**

```
filter.sort_by(field='metadata.user', values=dl.FiltersOrderByDirection.  
↪ASCENDING)
```

**class FiltersKnownFields**(*value*)

Bases: `str`, `Enum`

An enumeration.

**class FiltersMethod**(*value*)

Bases: `str`, `Enum`

An enumeration.

**class FiltersOperations**(*value*)

Bases: `str`, `Enum`

An enumeration.

**class FiltersOrderByDirection**(*value*)

Bases: `str`, `Enum`

An enumeration.

**class FiltersResource**(*value*)

Bases: `str`, `Enum`

An enumeration.

## 3.7 Recipe

**class** `Recipe`(*id, creator, url, title, project\_ids, description, ontology\_ids, instructions, examples, custom\_actions, metadata, ui\_settings, client\_api: ApiClient, dataset=None, project=None, repositories=NOTHING*)

Bases: `BaseEntity`

Recipe object

**clone**(*shallow=False*)

Clone Recipe

**Parameters**

**shallow** (*bool*) – If True, link of existing ontology, clones all ontology that are link to the recipe as well

**Returns**

Cloned ontology object

**Return type**

*dtlpy.entities.recipe.Recipe*

**delete**(*force: bool = False*)

Delete recipe from platform

**Parameters**

**force** (*bool*) – force delete recipe

**Returns**

True

**Return type**

*bool*

**classmethod** `from_json`(*\_json, client\_api, dataset=None, project=None, is\_fetched=True*)

Build a Recipe entity object from a json

**Parameters**

- **\_json** (*dict*) – \_json response from host
- **Dataset** (*dtlpy.entities.dataset.Dataset*) – Dataset entity
- **project** (*dtlpy.entities.project.Project*) – project entity
- **client\_api** (*dtlpy.ApiClient*) – ApiClient entity
- **is\_fetched** (*bool*) – is Entity fetched from Platform

**Returns**

Recipe object

**get\_annotation\_template\_id**(*template\_name*)

Get annotation template id by template name

**Parameters**

**template\_name** (*str*) –

#### Returns

template id or None if does not exist

#### `open_in_web()`

Open the recipes in web platform

#### Returns

#### `to_json()`

Returns platform \_json format of object

#### Returns

platform json format of object

#### Return type

`dict`

#### `update(system_metadata=False)`

Update Recipe

#### Parameters

**system\_metadata** (*bool*) – bool - True, if you want to change metadata system

#### Returns

Recipe object

#### Return type

*dtlpy.entities.recipe.Recipe*

## 3.7.1 Ontology

```
class Ontology(client_api: ApiClient, id, creator, url, title, labels, metadata, attributes, recipe=None,
               dataset=None, project=None, repositories=NOTHING, instance_map=None, color_map=None)
```

Bases: BaseEntity

Ontology object

```
add_label(label_name, color=None, children=None, attributes=None, display_label=None, label=None,
           add=True, icon_path=None, update_ontology=False)
```

Add a single label to ontology

#### Parameters

- **label\_name** (*str*) – str - label name
- **color** (*tuple*) – color
- **children** – children (sub labels)
- **attributes** (*list*) – attributes
- **display\_label** (*str*) – display\_label
- **label** (*dtlpy.entities.label.Label*) – label
- **add** (*bool*) – to add or not
- **icon\_path** (*str*) – path to image to be display on label
- **update\_ontology** (*bool*) – update the ontology, default = False for backward compatible



**Returns**

Label entity

**Return type**

dtlpy.entities.label.Label

**Example:**

```
ontology.add_label(label_name='person', color=(34, 6, 231), attributes=['big',
↪ 'small'])
```

**add\_labels**(*label\_list*, *update\_ontology=False*)

Adds a list of labels to ontology

**Parameters**

- **label\_list** (*list*) – list of labels [{"value": {"tag": "tag", "displayLabel": "displayLabel", "color": "#color", "attributes": [attributes]}, "children": [children]}]
- **update\_ontology** (*bool*) – update the ontology, default = False for backward compatible

**Returns**

List of label entities added

**Example:**

```
ontology.add_labels(label_list=label_list)
```

**property color\_map**

rgb}

**Returns**

dict

**Return type***dict***Type**

Color mapping of labels, {label

**delete()**

Delete recipe from platform

**Returns**

True

**delete\_attributes**(*keys: list*)

Delete a bulk of attributes

**Parameters****keys** (*list*) – Keys of attributes to delete**Returns**

True if success

**Return type***bool***Example:**

```
ontology.delete_attributes(['1'])
```

**delete\_labels**(*label\_names*)

Delete labels from ontology

**Parameters**

**label\_names** – label object/ label name / list of label objects / list of label names

**Returns**

**classmethod from\_json**(*\_json, client\_api, recipe, dataset=None, project=None, is\_fetched=True*)

Build an Ontology entity object from a json

**Parameters**

- **is\_fetched** (*bool*) – is Entity fetched from Platform
- **project** ([dtlpy.entities.project.Project](#)) – project entity
- **dataset** ([dtlpy.entities.dataset.Dataset](#)) – dataset
- **\_json** (*dict*) – \_json response from host
- **recipe** ([dtlpy.entities.recipe.Recipe](#)) – ontology’s recipe
- **client\_api** (*dl.ApiClient*) – ApiClient entity

**Returns**

Ontology object

**Return type**

[dtlpy.entities.ontology.Ontology](#)

**property instance\_map**

instance mapping for creating instance mask

**Return dictionary {label**

map\_id}

**Return type**

*dict*

**to\_json**()

Returns platform \_json format of object

**Returns**

platform json format of object

**Return type**

*dict*

**update**(*system\_metadata=False*)

Update items metadata

**Parameters**

**system\_metadata** (*bool*) – bool - True, if you want to change metadata system

**Returns**

Ontology object

**update\_attributes**(*title: str, key: str, attribute\_type, scope: Optional[list] = None, optional: Optional[bool] = None, multi: Optional[bool] = None, values: Optional[list] = None, attribute\_range=None*)

ADD a new attribute or update if exist

**Parameters**

- **title** (*str*) – attribute title
- **key** (*str*) – the key of the attribute must be unique
- **attribute\_type** (*AttributesTypes*) – `dl.AttributesTypes` your attribute type
- **scope** (*list*) – list of the labels or `*` for all labels
- **optional** (*bool*) – optional attribute
- **multi** (*bool*) – if can get multiple selection
- **values** (*list*) – list of the attribute values ( for checkbox and radio button)
- **attribute\_range** (*dict* or *AttributesRange*) – `dl.AttributesRange` object

**Returns**

true in success

**Return type**

*bool*

**update\_label**(*label\_name*, *color=None*, *children=None*, *attributes=None*, *display\_label=None*, *label=None*, *add=True*, *icon\_path=None*, *upsert=False*, *update\_ontology=False*)

Update a single label to ontology

**Parameters**

- **label\_name** (*str*) – str - label name
- **color** (*tuple*) – color
- **children** – children (sub labels)
- **attributes** (*list*) – attributes
- **display\_label** (*str*) – display\_label
- **label** (*dtlpy.entities.label.Label*) – label
- **add** (*bool*) – to add or not
- **icon\_path** (*str*) – path to image to be display on label
- **update\_ontology** (*bool*) – update the ontology, default = False for backward compatible
- **upsert** (*bool*) – if True will add in case it does not existing

**Returns**

Label entity

**Return type**

`dtlpy.entities.label.Label`

**Example:**

```
ontology.update_label(label_name='person', color=(34, 6, 231), attributes=['big
→', 'small'])
```

**update\_labels**(*label\_list*, *upsert=False*, *update\_ontology=False*)

Update a list of labels to ontology

**Parameters**

- **label\_list** (*list*) – list of labels [{"value": {"tag": "tag", "displayLabel": "displayLabel", "color": "#color", "attributes": [attributes]}, "children": [children]}]

- **upsert** (*bool*) – if True will add in case it does not existing
- **update\_ontology** (*bool*) – update the ontology, default = False for backward compatible

**Returns**

List of label entities added

**Example:**

```
ontology.update_labels(label_list=label_list)
```

**Label**

## 3.8 Task

```
class Task(name, status, project_id, metadata, id, url, task_owner, item_status, creator, due_date, dataset_id,
            spec, recipe_id, query, assignmentIds, annotation_status, progress, for_review, issues, updated_at,
            created_at, available_actions, total_items, client_api, current_assignments=None, assignments=None,
            project=None, dataset=None, tasks=None, settings=None)
```

Bases: `object`

Task object

```
add_items(filters=None, items=None, assignee_ids=None, workload=None, limit=None, wait=True,
           query=None)
```

Add items to Task

**Parameters**

- **filters** (`dtlpy.entities.filters.Filters`) – Filters entity or a dictionary containing filters parameters
- **items** (*list*) – list of items to add to the task
- **assignee\_ids** (*list*) – list to assignee who works in the task
- **workload** (*list*) – list of the work load per assignee and work load
- **limit** (*int*) – task limit
- **wait** (*bool*) – wait for the command to finish
- **query** (*dict*) – query to filter the items use it

**Returns**

task entity

**Return type**

`dtlpy.entities.task.Task`

```
create_assignment(assignment_name, assignee_id, items=None, filters=None)
```

Create a new assignment

**Parameters**

- **assignment\_name** (*str*) – assignment name
- **assignee\_id** (*list*) – list of assignee for the assignment
- **items** (*list*) – items list for the assignment

- **filters** (`dtlpy.entities.filters.Filters`) – Filters entity or a dictionary containing filters parameters

**Returns**

Assignment object

**Return type**

`dtlpy.entities.assignment.Assignment` assignment

**Example:**

```
task.create_assignment(assignee_id='annotator1@dataloop.ai')
```

```
create_qa_task(due_date, assignee_ids, filters=None, items=None, query=None, workload=None,
               metadata=None, available_actions=None, wait=True, batch_size=None,
               max_batch_workload=None, allowed_assignees=None)
```

Create a new QA Task

**Parameters**

- **due\_date** (*float*) – date to when finish the task
- **assignee\_ids** (*list*) – list of assignee
- **filters** (*entities.Filters*) – filter to the task
- **items** (*List[entities.Item]*) – item to insert to the task
- **query** (*entities.Filters*) – filter to the task
- **workload** (*List[WorkloadUnit]*) – list WorkloadUnit for the task assignee
- **metadata** (*dict*) – metadata for the task
- **available\_actions** (*list*) – list of available actions to the task
- **wait** (*bool*) – wait for the command to finish
- **batch\_size** (*int*) – Pulling batch size (items) . Restrictions - Min 3, max 100
- **max\_batch\_workload** (*int*) – Max items in assignment . Restrictions - Min batchSize + 2 , max batchSize \* 2
- **allowed\_assignees** (*list*) – It's like the workload, but without percentage.

**Returns**

task object

**Return type**

`dtlpy.entities.task.Task`

**Example:**

```
task.create_qa_task(due_date = datetime.datetime(day= 1, month= 1, year= 2029).
↳ timestamp(),
                    assignee_ids =[ 'annotator1@dataloop.ai',
↳ 'annotator2@dataloop.ai'])
```

**delete**(*wait=True*)

Delete task from platform

**Parameters**

- **wait** (*bool*) – wait for the command to finish

**Returns**

True

**Return type**

bool

**get\_items**(*filters=None*)

Get the task items

**Parameters**

**filters** (`dtlpy.entities.filters.Filters`) – Filters entity or a dictionary containing filters parameters

**Returns**

list of the items or PagedEntity output of items

**Return type**

list or `dtlpy.entities.paged_entities.PagedEntities`

**open\_in\_web**()

Open the task in web platform

**Returns**

**remove\_items**(*filters: Optional[Filters] = None, query=None, items=None, wait=True*)

remove items from Task.

**Prerequisites:** You must be in the role of an *owner*, *developer*, or *annotation manager* who has been assigned to be *owner* of the annotation task.

**Parameters**

- **filters** (`dtlpy.entities.filters.Filters`) – Filters entity or a dictionary containing filters parameters
- **query** (*dict*) – query to filter the items use it
- **items** (*list*) – list of items to add to the task
- **wait** (*bool*) – wait for the command to finish

**Returns**

task entity

**Return type**

`dtlpy.entities.task.Task`

**set\_status**(*status: str, operation: str, item\_ids: List[str]*)

Update item status within task

**Parameters**

- **status** (*str*) – string that describes the status
- **operation** (*str*) – ‘create’ or ‘delete’
- **item\_ids** (*list*) – List[str] id items ids

**Returns**

True if success

**Return type**

bool

**to\_json()**

Returns platform \_json format of object

**Returns**

platform json format of object

**Return type**

dict

**update(system\_metadata=False)**

Update an Annotation Task

**Parameters**

**system\_metadata** (*bool*) – True, if you want to change metadata system

### 3.8.1 Assignment

**class Assignment**(*name, annotator, status, project\_id, metadata, id, url, task\_id, dataset\_id, annotation\_status, item\_status, total\_items, for\_review, issues, client\_api, task=None, assignments=None, project=None, dataset=None, datasets=None*)

Bases: BaseEntity

Assignment object

**get\_items**(*dataset=None, filters=None*)

Get all the items in the assignment

**Prerequisites:** You must be in the role of an *owner*, *developer*, or *annotation manager* who has been assigned as *owner* of the annotation task.

**Parameters**

- **dataset** (`dtlpy.entities.dataset.Dataset`) – dataset entity
- **filters** (`dtlpy.entities.filters.Filters`) – Filters entity or a dictionary containing filters parameters

**Returns**

pages of the items

**Return type**

`dtlpy.entities.paged_entities.PagedEntities`

**Example:**

```
task.assignments.get_items()
```

**open\_in\_web()**

Open the assignment in web platform

**Prerequisites:** You must be in the role of an *owner*, *developer*, or *annotation manager* who has been assigned as *owner* of the annotation task.

**Returns**

**Example:**

```
assignment.open_in_web()
```

**reassign**(*assignee\_id*, *wait=True*)

Reassign an assignment

**Prerequisites:** You must be in the role of an *owner*, *developer*, or *annotation manager* who has been assigned as *owner* of the annotation task.

**Parameters**

- **assignee\_id** (*str*) – the user that assignee the assignment to it
- **wait** (*bool*) – wait for the command to finish

**Returns**

Assignment object

**Return type**

*dtlpy.entities.assignment.Assignment*

**Example:**

```
assignment.reassign(assignee_ids='annotator1@dataloop.ai')
```

**redistribute**(*workload*, *wait=True*)

Redistribute an assignment

**Prerequisites:** You must be in the role of an *owner*, *developer*, or *annotation manager* who has been assigned as *owner* of the annotation task.

**Parameters**

- **workload** (*dtlpy.entities.assignment.Workload*) – workload object that contain the assignees and the work load
- **wait** (*bool*) – wait for the command to finish

**Returns**

Assignment object

**Return type**

*dtlpy.entities.assignment.Assignment* assignment

**Example:**

```
assignment.redistribute(workload=dl.Workload([dl.WorkloadUnit(assignee_id=
↪ "annotator1@dataloop.ai", load=50),
                                                    dl.WorkloadUnit(assignee_id=
↪ "annotator2@dataloop.ai", load=50)]))
```

**set\_status**(*status: str*, *operation: str*, *item\_id: str*)

Set item status within assignment

**Prerequisites:** You must be in the role of an *owner*, *developer*, or *annotation manager* who has been assigned as *owner* of the annotation task.

**Parameters**

- **status** (*str*) – status
- **operation** (*str*) – created/deleted
- **item\_id** (*str*) – item id

**Returns**

True id success



**Return type**`bool`**Example:**

```
assignment.set_status(status='complete',
                      operation='created',
                      item_id='item_id')
```

**to\_json()**

Returns platform \_json format of object

**Returns**

platform json format of object

**Return type**`dict`**update(system\_metadata=False)**

Update an assignment

**Prerequisites:** You must be in the role of an *owner*, *developer*, or *annotation manager* who has been assigned as *owner* of the annotation task.

**Parameters****system\_metadata** (`bool`) – True, if you want to change metadata system**Returns**

Assignment object

**Return type**`dtlpy.entities.assignment.Assignment` assignment**Example:**

```
assignment.update(system_metadata=False)
```

**class Workload(workload: list = NOTHING)**Bases: `object`

Workload object

**add(assignee\_id)**

add a assignee

**Parameters****assignee\_id** –**classmethod generate(assignee\_ids, loads=None)**

generate the loads for the given assignee :param assignee\_ids: :param loads:

**class WorkloadUnit(assignee\_id: str, load: float = 0)**Bases: `object`

WorkloadUnit object

## 3.9 Package

```
class Package(id, url, version, created_at, updated_at, name, codebase, modules, slots: list, ui_hooks, creator,  
             is_global, type, service_config, project_id, project, client_api: ApiClient, revisions=None,  
             repositories=NOTHING, artifacts=None, codebases=None, requirements=None)
```

Bases: BaseEntity

Package object

**checkout()**

Checkout as package

**Returns**

**delete()**

Delete Package object

**Returns**

True

```
deploy(service_name=None, revision=None, init_input=None, runtime=None, sdk_version=None,  
      agent_versions=None, verify=True, bot=None, pod_type=None, module_name=None,  
      run_execution_as_process=None, execution_timeout=None, drain_time=None, on_reset=None,  
      max_attempts=None, force=False, secrets: Optional[list] = None, **kwargs)
```

Deploy package

**Parameters**

- **service\_name** (*str*) – service name
- **revision** (*str*) – package revision - default=latest
- **init\_input** – config to run at startup
- **runtime** (*dict*) – runtime resources
- **sdk\_version** (*str*) –
  - optional - string - sdk version
- **agent\_versions** (*dict*) –
  - dictionary - - optional -versions of sdk, agent runner and agent proxy
- **bot** (*str*) – bot email
- **pod\_type** (*str*) – pod type dl.InstanceCatalog
- **verify** (*bool*) – verify the inputs
- **module\_name** (*str*) – module name
- **run\_execution\_as\_process** (*bool*) – run execution as process
- **execution\_timeout** (*int*) – execution timeout
- **drain\_time** (*int*) – drain time
- **on\_reset** (*str*) – on reset
- **max\_attempts** (*int*) – Maximum execution retries in-case of a service reset
- **force** (*bool*) – optional - terminate old replicas immediately
- **secrets** (*list*) – list of the integrations ids

**Returns**

Service object

**Return type***dtlpy.entities.service.Service***Example:**

```

package.deploy(service_name=package_name,
                execution_timeout=3 * 60 * 60,
                module_name=module.name,
                runtime=dl.KubernetesRuntime(
                    concurrency=10,
                    pod_type=dl.InstanceCatalog.REGULAR_S,
                    autoscaler=dl.KubernetesRabbitmqAutoscaler(
                        min_replicas=1,
                        max_replicas=20,
                        queue_length=20
                    )
                )
            )
    )

```

**classmethod from\_json**(*\_json*, *client\_api*, *project*, *is\_fetched=True*)

Turn platform representation of package into a package entity

**Parameters**

- **\_json** (*dict*) – platform representation of package
- **client\_api** (*dl.ApiClient*) – ApiClient entity
- **project** (*dtlpy.entities.project.Project*) – project entity
- **is\_fetched** – is Entity fetched from Platform

**Returns**

Package entity

**Return type***dtlpy.entities.package.Package*

**open\_in\_web**()

Open the package in web platform

**pull**(*version=None*, *local\_path=None*)

Pull local package

**Parameters**

- **version** (*str*) – version
- **local\_path** (*str*) – local path

**Example:**

```
package.pull(local_path='local_path')
```

**push**(*codebase: Optional[Union[GitCodebase, ItemCodebase]] = None*, *src\_path: Optional[str] = None*, *package\_name: Optional[str] = None*, *modules: Optional[list] = None*, *checkout: bool = False*, *revision\_increment: Optional[str] = None*, *service\_update: bool = False*, *service\_config: Optional[dict] = None*)

Push local package

#### Parameters

- **codebase** (*dtlpy.entities.codebase.Codebase*) – PackageCode object - defines how to store the package code
- **checkout** (*bool*) – save package to local checkout
- **src\_path** (*str*) – location of package codebase folder to zip
- **package\_name** (*str*) – name of package
- **modules** (*list*) – list of PackageModule
- **revision\_increment** (*str*) – optional - str - version bumping method - major/minor/patch - default = None
- **service\_update** (*bool*) – optional - bool - update the service
- **service\_config** (*dict*) – optional - json of service - a service that have config from the main service if wanted

#### Returns

package entity

#### Return type

*dtlpy.entities.package.Package*

#### Example:

```
packages.push(package_name='package_name',
              modules=[module],
              version='1.0.0',
              src_path=os.getcwd()
            )
```

```
test(cwd=None, concurrency=None, module_name='default_module', function_name='run',
     class_name='ServiceRunner', entry_point='main.py')
```

Test local package in local environment.

#### Parameters

- **cwd** (*str*) – path to the file
- **concurrency** (*int*) – the concurrency of the test
- **module\_name** (*str*) – module name
- **function\_name** (*str*) – function name
- **class\_name** (*str*) – class name
- **entry\_point** (*str*) – the file to run like main.py

#### Returns

list created by the function that tested the output

#### Return type

*list*

#### Example:

```
package.test(cwd='path_to_package',
             function_name='run')
```

**to\_json()**

Turn Package entity into a platform representation of Package

**Returns**

platform json of package

**Return type**

dict

**update()**

Update Package changes to platform

**Returns**

Package entity

**class RequirementOperator(value)**

Bases: `str`, `Enum`

An enumeration.

### 3.9.1 Package Function

```
class PackageFunction(outputs=NOTHING, name=NOTHING, description="", inputs=NOTHING,
                       display_name=None, display_icon=None)
```

Bases: `BaseEntity`

Webhook object

**class PackageInputType(value)**

Bases: `str`, `Enum`

An enumeration.

### 3.9.2 Package Module

```
class PackageModule(name=NOTHING, init_inputs=NOTHING, entry_point='main.py',
                     class_name='ServiceRunner', functions=NOTHING)
```

Bases: `BaseEntity`

PackageModule object

**add\_function(function)****Parameters**

**function** –

### 3.9.3 Slot

```
class PackageSlot(module_name='default_module',function_name='run',display_name=None,  
                  display_scopes: Optional[list] = None,display_icon=None,post_action: SlotPostAction =  
                  NOTHING,default_inputs: Optional[list] = None,input_options: Optional[list] = None)
```

Bases: BaseEntity

Webhook object

```
class SlotDisplayScopeResource(value)
```

Bases: str, Enum

An enumeration.

```
class SlotPostActionType(value)
```

Bases: str, Enum

An enumeration.

```
class UiBindingPanel(value)
```

Bases: str, Enum

An enumeration.

### 3.9.4 Codebase

## 3.10 Service

```
class InstanceCatalog(value)
```

Bases: str, Enum

The Service Pode size.

State	Description
REG-U-LAR_XS	regular pod with extra small size
REG-U-LAR_S	regular pod with small size
REG-U-LAR_M	regular pod with medium size
REG-U-LAR_L	regular pod with large size
REG-U-LAR_XL	regular pod with extra large size
HIGH-MEM_XS	highmem pod with extra small size
HIGH-MEM_S	highmem pod with small size
HIGH-MEM_M	highmem pod with medium size
HIGH-MEM_L	highmem pod with large size
HIGH-MEM_XL	highmem pod with extra large size
GPU_K80G8U	GPU pod with small size
GPU_K80G16U	GPU pod with medium size

**class KubernetesAutoscalerType**(*value*)

Bases: `str`, `Enum`

The Service Autoscaler Type (RABBITMQ, CPU).

State	Description
RAB-BITMQ	Service Autoscaler will be in RABBITMQ
CPU	Service Autoscaler will be in in local CPU

**class OnResetAction**(*value*)

Bases: `str`, `Enum`

The Execution action when the service reset (RERUN, FAILED).

State	Description
RE-RUN	When the service resting rerun the execution
FAILED	When the service resting fail the execution

**class RuntimeType**(*value*)

Bases: `str`, `Enum`

Service culture Runtime (KUBERNETES).

State	Description
KU-BER-NETES	Service run in kubernetes culture

```
class Service(created_at, updated_at, creator, version, package_id, package_revision, bot, use_user_jwt,
               init_input, versions, module_name, name, url, id, active, driver_id, secrets, runtime,
               queue_length_limit, run_execution_as_process: bool, execution_timeout, drain_time, on_reset:
               OnResetAction, project_id, is_global, max_attempts, package, client_api: ApiClient,
               revisions=None, project=None, repositories=NOTHING)
```

Bases: BaseEntity

Service object

```
activate_slots(project_id: Optional[str] = None, task_id: Optional[str] = None, dataset_id:
               Optional[str] = None, org_id: Optional[str] = None, user_email: Optional[str] = None,
               slots=None, role=None, prevent_override: bool = True, visible: bool = True, icon: str =
               'fas fa-magic', **kwargs) → object
```

Activate service slots

#### Parameters

- **project\_id** (*str*) – project id
- **task\_id** (*str*) – task id
- **dataset\_id** (*str*) – dataset id
- **org\_id** (*str*) – org id
- **user\_email** (*str*) – user email
- **slots** (*list*) – list of entities.PackageSlot
- **role** (*str*) – user role MemberOrgRole.ADMIN, MemberOrgRole.owner, MemberOrgRole.MEMBER
- **prevent\_override** (*bool*) – True to prevent override
- **visible** (*bool*) – visible
- **icon** (*str*) – icon
- **kwargs** – all additional arguments

#### Returns

list of user setting for activated slots

#### Return type

*list*

#### Example:

```
service.activate_slots(project_id='project_id',
                       slots=List[entities.PackageSlot],
                       icon='fas fa-magic')
```



**checkout()**

Checkout

**Returns****delete()**

Delete Service object

**Returns**

True

**Return type**

bool

**execute**(*execution\_input=None, function\_name=None, resource=None, item\_id=None, dataset\_id=None, annotation\_id=None, project\_id=None, sync=False, stream\_logs=True, return\_output=True*)

Execute a function on an existing service

**Parameters**

- **execution\_input** (*List[FunctionIO]* or *dict*) – input dictionary or list of FunctionIO entities
- **function\_name** (*str*) – function name to run
- **resource** (*str*) – input type.
- **item\_id** (*str*) – optional - item id as input to function
- **dataset\_id** (*str*) – optional - dataset id as input to function
- **annotation\_id** (*str*) – optional - annotation id as input to function
- **project\_id** (*str*) – resource's project
- **sync** (*bool*) – if true, wait for function to end
- **stream\_logs** (*bool*) – prints logs of the new execution. only works with sync=True
- **return\_output** (*bool*) – if True and sync is True - will return the output directly

**Returns**

execution object

**Return type***dtlpy.entities.execution.Execution***Example:**

```
service.execute(function_name='function_name', item_id='item_id', project_id=
↪ 'project_id')
```

**classmethod from\_json**(*\_json: dict, client\_api: ApiClient, package=None, project=None, is\_fetched=True*)

Build a service entity object from a json

**Parameters**

- **\_json** (*dict*) – platform json
- **client\_api** (*dtlpy.ApiClient*) – ApiClient entity
- **package** (*dtlpy.entities.package.Package*) – package entity
- **project** (*dtlpy.entities.project.Project*) – project entity

- **is\_fetched** (*bool*) – is Entity fetched from Platform

**Returns**

service object

**Return type**

*dtlpy.entities.service.Service*

**log**(*size=None, checkpoint=None, start=None, end=None, follow=False, text=None, execution\_id=None, function\_name=None, replica\_id=None, system=False, view=True, until\_completed=True*)

Get service logs

**Parameters**

- **size** (*int*) – size
- **checkpoint** (*dict*) – the information from the 1st point checked in the service
- **start** (*str*) – iso format time
- **end** (*str*) – iso format time
- **follow** (*bool*) – if true, keep stream future logs
- **text** (*str*) – text
- **execution\_id** (*str*) – execution id
- **function\_name** (*str*) – function name
- **replica\_id** (*str*) – replica id
- **system** (*bool*) – system
- **view** (*bool*) – if true, print out all the logs
- **until\_completed** (*bool*) – wait until completed

**Returns**

ServiceLog entity

**Return type**

*ServiceLog*

**Example:**

```
service.log()
```

**open\_in\_web()**

Open the service in web platform

**Returns****pause()****Returns****resume()****Returns****status()**

Get Service status

**Returns**

status json

**Return type**`dict`**to\_json()**

Returns platform \_json format of object

**Returns**

platform json format of object

**Return type**`dict`**update(*force=False*)**

Update Service changes to platform

**Parameters****force** (*bool*) – force update**Returns**

Service entity

**Return type***dtlpy.entities.service.Service*

### 3.10.1 Bot

**class Bot**(*created\_at, updated\_at, name, last\_name, username, avatar, email, role, type, org, id, project, client\_api=None, users=None, bots=None, password=None*)

Bases: *User*

Bot entity

**delete()**

Delete the bot

**Returns**

True

**Return type**`bool`**classmethod from\_json(\_json, project, client\_api, bots=None)**

Build a Bot entity object from a json

**Parameters**

- **\_json** – \_json response from host
- **project** – project entity
- **client\_api** – ApiClient entity
- **bots** – Bots repository

**Returns**

User object

**to\_json()**

Returns platform \_json format of object

**Returns**

platform json format of object

**Return type**

dict

## 3.11 Trigger

```
class BaseTrigger(id, url, created_at, updated_at, creator, name, active, type, scope, is_global, input,  
                  function_name, service_id, webhook_id, pipeline_id, special, project_id, spec, service,  
                  project, client_api: ApiClient, op_type='service', repositories=NOTHING)
```

Bases: BaseEntity

Trigger Entity

**delete()**

Delete Trigger object

**Returns**

True

```
classmethod from_json(_json, client_api, project, service=None)
```

Build a trigger entity object from a json

**Parameters**

- **\_json** (*dict*) – platform json
- **client\_api** (*dl.ApiClient*) – ApiClient entity
- **project** (*dtlpy.entities.project.Project*) – project entity
- **service** (*dtlpy.entities.service.Service*) – service entity

**Returns****to\_json()**

Returns platform \_json format of object

**Returns**

platform json format of object

**Return type**

dict

**update()**

Update Trigger object

**Returns**

Trigger entity

```
class CronTrigger(id, url, created_at, updated_at, creator, name, active, type, scope, is_global, input,  
                  function_name, service_id, webhook_id, pipeline_id, special, project_id, spec, service,  
                  project, client_api: ApiClient, op_type='service', repositories=NOTHING, start_at=None,  
                  end_at=None, cron=None)
```

Bases: BaseTrigger

```
classmethod from_json(_json, client_api, project, service=None)
```

Build a trigger entity object from a json

**Parameters**

- **\_json** – platform json

- **client\_api** – ApiClient entity
- **project** – project entity
- **service** – service entity

**Returns****to\_json()**

Returns platform \_json format of object

**Returns**

platform json format of object

**Return type**

dict

```
class Trigger(id, url, created_at, updated_at, creator, name, active, type, scope, is_global, input, function_name,
               service_id, webhook_id, pipeline_id, special, project_id, spec, service, project, client_api:
               ApiClient, op_type='service', repositories=NOTHING, filters=None,
               execution_mode=TriggerExecutionMode.ONCE, actions=TriggerAction.CREATED,
               resource=TriggerResource.ITEM)
```

Bases: [BaseTrigger](#)

Trigger Entity

```
classmethod from_json(_json, client_api, project, service=None)
```

Build a trigger entity object from a json

**Parameters**

- **\_json** – platform json
- **client\_api** – ApiClient entity
- **project** ([dtlpy.entities.project.Project](#)) – project entity
- **service** ([dtlpy.entities.service.Service](#)) – service entity

**Returns****to\_json()**

Returns platform \_json format of object

**Returns**

platform json format of object

**Return type**

dict

```
class TriggerAction(value)
```

Bases: [str](#), [Enum](#)

An enumeration.

```
class TriggerExecutionMode(value)
```

Bases: [str](#), [Enum](#)

An enumeration.

```
class TriggerResource(value)
```

Bases: [str](#), [Enum](#)

An enumeration.

```
class TriggerType(value)
```

Bases: `str`, `Enum`

An enumeration.

## 3.12 Execution

```
class Execution(id, url, creator, created_at, updated_at, input, output, feedback_queue, status, status_log,
                sync_reply_to, latest_status, function_name, duration, attempts, max_attempts, to_terminate:
                bool, trigger_id, service_id, project_id, service_version, package_id, package_name, client_api:
                ApiClient, service, project=None, repositories=NOTHING, pipeline: Optional[dict] = None)
```

Bases: `BaseEntity`

Service execution entity

```
classmethod from_json(_json, client_api, project=None, service=None, is_fetched=True)
```

### Parameters

- **\_json** (*dict*) – platform json
- **client\_api** (*dtlpy.ApiClient*) – ApiClient entity
- **project** (*dtlpy.entities.project.Project*) – project entity
- **service** (*dtlpy.entities.service.Service*) –
- **is\_fetched** – is Entity fetched from Platform

```
increment()
```

Increment attempts

### Returns

```
logs(follow=False)
```

Print logs for execution

### Parameters

**follow** – keep stream future logs

```
progress_update(status: Optional[ExecutionStatus] = None, percent_complete: Optional[int] = None,
                 message: Optional[str] = None, output: Optional[str] = None, service_version:
                 Optional[str] = None)
```

Update Execution Progress

### Parameters

- **status** (*str*) – ExecutionStatus
- **percent\_complete** (*int*) – percent complete
- **message** (*str*) – message to update the progress state
- **output** (*str*) – output
- **service\_version** (*str*) – service version

### Returns

Service execution object

**rerun()**

Re-run

**Returns**

Execution object

**terminate()**

Terminate execution

**Returns**

execution object

**to\_json()**

Returns platform \_json format of object

**Returns**

platform json format of object

**Return type**

`dict`

**update()**

Update execution changes to platform

**Returns**

execution entity

**wait()**

Wait for execution

**Returns**

Service execution object

**class ExecutionStatus**(*value*)

Bases: `str`, `Enum`

An enumeration.

## 3.13 Pipeline

```
class Pipeline(id, name, creator, org_id, connections, created_at, updated_at, start_nodes, project_id,  
               composition_id, url, preview, description, revisions, project, client_api: ApiClient,  
               repositories=NOTHING)
```

Bases: `BaseEntity`

Package object

**delete()**

Delete pipeline object

**Returns**

`True`

**execute**(*execution\_input=None*)

execute a pipeline and return the execute

**Parameters**

**execution\_input** – list of the `dl.FunctionIO` or dict of pipeline input - example { 'item': 'item\_id' }

### Returns

entities.PipelineExecution object

**classmethod** `from_json(_json, client_api, project, is_fetched=True)`

Turn platform representation of pipeline into a pipeline entity

### Parameters

- `_json` (*dict*) – platform representation of package
- `client_api` (*dl.ApiClient*) – ApiClient entity
- `project` (*dtlpy.entities.project.Project*) – project entity
- `is_fetched` (*bool*) – is Entity fetched from Platform

### Returns

Pipeline entity

### Return type

*dtlpy.entities.pipeline.Pipeline*

**install()**

install pipeline

### Returns

Composition entity

**open\_in\_web()**

Open the pipeline in web platform

### Returns

**pause()**

pause pipeline

### Returns

Composition entity

**reset**(*stop\_if\_running: bool = False*)

Resets pipeline counters

### Parameters

**stop\_if\_running** (*bool*) – If the pipeline is installed it will stop the pipeline and reset the counters.

### Returns

bool

**set\_start\_node**(*node: PipelineNode*)

Set the start node of the pipeline

### Parameters

**node** (*PipelineNode*) – node to be the start node

**stats()**

Get pipeline counters

### Returns

PipelineStats

### Return type

*dtlpy.entities.pipeline.PipelineStats*



**to\_json()**

Turn Package entity into a platform representation of Package

**Returns**

platform json of package

**Return type**

`dict`

**update()**

Update pipeline changes to platform

**Returns**

pipeline entity

### 3.13.1 Pipeline Execution

**class PipelineExecution**(*id, nodes, executions, status, created\_at, updated\_at, pipeline\_id, max\_attempts, pipeline, client\_api: ApiClient, repositories=NOTHING*)

Bases: BaseEntity

Package object

**classmethod from\_json**(*\_json, client\_api, pipeline, is\_fetched=True*)

Turn platform representation of pipeline\_execution into a pipeline\_execution entity

**Parameters**

- **\_json** (*dict*) – platform representation of package
- **client\_api** (*dl.ApiClient*) – ApiClient entity
- **pipeline** (*dtlpy.entities.pipeline.Pipeline*) – Pipeline entity
- **is\_fetched** (*bool*) – is Entity fetched from Platform

**Returns**

Pipeline entity

**Return type**

*dtlpy.entities.pipeline.Pipeline*

**to\_json()**

Turn Package entity into a platform representation of Package

**Returns**

platform json of package

**Return type**

`dict`

## 3.14 Other

### 3.14.1 Pages

```
class PagedEntities(client_api: ApiClient, page_offset, page_size, filters, items_repository,  
                   has_next_page=False, total_pages_count=0, items_count=0, service_id=None,  
                   project_id=None, order_by_type=None, order_by_direction=None,  
                   execution_status=None, execution_resource_type=None, execution_resource_id=None,  
                   execution_function_name=None, items=[])
```

Bases: `object`

Pages object

```
get_page(page_offset=None, page_size=None)
```

Get page

**Parameters**

- **page\_offset** – page offset
- **page\_size** – page size

```
go_to_page(page=0)
```

Brings specified page of items from host

**Parameters**

**page** – page number

**Returns**

```
next_page()
```

Brings the next page of items from host

**Returns**

```
prev_page()
```

Brings the previous page of items from host

**Returns**

```
process_result(result)
```

**Parameters**

**result** – json object

```
return_page(page_offset=None, page_size=None)
```

Return page

**Parameters**

- **page\_offset** – page offset
- **page\_size** – page size

### 3.14.2 Base Entity

### 3.14.3 Command

**class** `Command`(*id, url, status, created\_at, updated\_at, type, progress, spec, error, client\_api: ApiClient, repositories=NOTHING*)

Bases: `BaseEntity`

Com entity

**abort()**

abort command

**Returns**

**classmethod** `from_json`(*\_json, client\_api, is\_fetched=True*)

Build a Command entity object from a json

**Parameters**

- **\_json** – \_json response from host
- **client\_api** – ApiClient entity
- **is\_fetched** – is Entity fetched from Platform

**Returns**

Command object

**in\_progress()**

Check if command is still in one of the in progress statuses

**Returns**

True if command still in progress

**Return type**

`bool`

**to\_json()**

Returns platform \_json format of object

**Returns**

platform json format of object

**Return type**

`dict`

**wait**(*timeout=0, step=None, backoff\_factor=0.1*)

Wait for Command to finish

**Parameters**

- **timeout** (`int`) – int, seconds to wait until TimeoutError is raised. if 0 - wait until done
- **step** (`int`) – int, seconds between polling
- **backoff\_factor** (`float`) – A backoff factor to apply between attempts after the second try

**Returns**

Command object

**class** **CommandsStatus**(*value*)

Bases: `str`, `Enum`

An enumeration.

### 3.14.4 Directory Tree

**class** **DirectoryTree**(*\_json*)

Bases: `object`

Dataset DirectoryTree

**class** **SingleDirectory**(*value, directory\_tree, children=None*)

Bases: `object`

DirectoryTree single directory

## 4.1 converter

**class Converter**(*concurrency=6, return\_error\_filepath=False*)

Bases: `object`

Annotation Converter

**attach\_agent\_progress**(*progress: Progress, progress\_update\_frequency: Optional[int] = None*)

Attach agent progress.

**Parameters**

- **progress** (*Progress*) – the progress object that follows the work
- **progress\_update\_frequency** (*int*) – progress update frequency in percentages

**convert**(*annotations, from\_format: str, to\_format: str, conversion\_func=None, item=None*)

Convert annotation list or single annotation.

**Prerequisites:** You must be an *owner* or *developer* to use this method.

**Parameters**

- **item** (*dtlpy.entities.item.Item*) – item entity
- **annotations** (*list or AnnotationCollection*) – annotations list to convert
- **from\_format** (*str*) – AnnotationFormat to convert to – AnnotationFormat.COCO, AnnotationFormat.YOLO, AnnotationFormat.VOC, AnnotationFormat.DATALOOP
- **to\_format** (*str*) – AnnotationFormat to convert to – AnnotationFormat.COCO, AnnotationFormat.YOLO, AnnotationFormat.VOC, AnnotationFormat.DATALOOP
- **conversion\_func** (*Callable*) – Custom conversion service

**Returns**

the annotations

**convert\_dataset**(*dataset, to\_format: str, local\_path: str, conversion\_func=None, filters=None, annotation\_filter=None*)

Convert entire dataset.

**Prerequisites:** You must be an *owner* or *developer* to use this method.

**Parameters**

- **dataset** (*dtlpy.entities.dataet.Dataset*) – dataset entity

- **to\_format** (*str*) – AnnotationFormat to convert to – AnnotationFormat.COCO, AnnotationFormat.YOLO, AnnotationFormat.VOC, AnnotationFormat.DATALOOP
- **local\_path** (*str*) – path to save the result to
- **conversion\_func** (*Callable*) – Custom conversion service
- **filters** (`dtlpy.entities.filters.Filters`) – Filters entity or a dictionary containing filter parameters
- **annotation\_filter** (`dtlpy.entities.filters.Filters`) – Filter entity

**Returns**

the error log file path if there are errors and the coco json if the format is coco

**convert\_directory**(*local\_path*: *str*, *to\_format*: *AnnotationFormat*, *from\_format*: *AnnotationFormat*, *dataset*, *conversion\_func*=None)

Convert annotation files in entire directory.

**Prerequisites:** You must be an *owner* or *developer* to use this method.

**Parameters**

- **local\_path** (*str*) – path to the directory
- **to\_format** (*str*) – AnnotationFormat to convert to – AnnotationFormat.COCO, AnnotationFormat.YOLO, AnnotationFormat.VOC, AnnotationFormat.DATALOOP
- **from\_format** (*str*) – AnnotationFormat to convert from – AnnotationFormat.COCO, AnnotationFormat.YOLO, AnnotationFormat.VOC, AnnotationFormat.DATALOOP
- **dataset** (`dtlpy.entities.dataset.Dataset`) – dataset entity
- **conversion\_func** (*Callable*) – Custom conversion service

**Returns**

the error log file path if there are errors

**convert\_file**(*to\_format*: *str*, *from\_format*: *str*, *file\_path*: *str*, *save\_locally*: *bool* = False, *save\_to*: *Optional[str]* = None, *conversion\_func*=None, *item*=None, *pbar*=None, *upload*: *bool* = False, \*\*\_)

Convert file containing annotations.

**Prerequisites:** You must be an *owner* or *developer* to use this method.

**Parameters**

- **to\_format** (*str*) – AnnotationFormat to convert to – AnnotationFormat.COCO, AnnotationFormat.YOLO, AnnotationFormat.VOC, AnnotationFormat.DATALOOP
- **from\_format** (*str*) – AnnotationFormat to convert from – AnnotationFormat.COCO, AnnotationFormat.YOLO, AnnotationFormat.VOC, AnnotationFormat.DATALOOP
- **file\_path** (*str*) – path of the file to convert
- **pbar** (*tqdm*) – tqdm object that follows the work (progress bar)
- **upload** (*bool*) – if True upload
- **save\_locally** (*bool*) – If True, save locally
- **save\_to** (*str*) – path to save the result to
- **conversion\_func** (*Callable*) – Custom conversion service
- **item** (`dtlpy.entities.item.Item`) – item entity

**Returns**

annotation list, errors

**static custom\_format**(*annotation*, *conversion\_func*, *i\_annotation=None*, *annotations=None*,  
*from\_format=None*, *item=None*, *\*\*\_*)

Custom convert function.

**Prerequisites:** You must be an *owner* or *developer* to use this method.

**Parameters**

- **annotation** (`dtlpy.entities.annotation.Annotation` or *dict*) – annotations to convert
- **conversion\_func** (*Callable*) – Custom conversion service
- **i\_annotation** (*int*) – annotation index
- **annotations** (*list*) – list of annotations

param str from\_format: AnnotationFormat to convert to – AnnotationFormat.COCO, AnnotationFormat.YOLO, AnnotationFormat.VOC, AnnotationFormat.DATALOOP :param dtlpy.entities.item.Item item: item entity :return: converted Annotation

**from\_coco**(*annotation*, *\*\*kwargs*)

Convert from COCO format to DATALOOP format. Use this as *conversion\_func* param for functions that ask for this param.

**Prerequisites:** You must be an *owner* or *developer* to use this method.

**Parameters**

- **annotation** – annotations to convert
- **kwargs** – additional params

**Returns**

converted Annotation entity

**Return type**

`dtlpy.entities.annotation.Annotation`

**static from\_voc**(*annotation*, *\*\*\_*)

Convert from VOC format to DATALOOP format. Use this as *conversion\_func* for functions that ask for this param.

**Prerequisites:** You must be an *owner* or *developer* to use this method.

**Parameters**

**annotation** – annotations to convert

**Returns**

converted Annotation entity

**Return type**

`dtlpy.entities.annotation.Annotation`

**from\_yolo**(*annotation*, *item=None*, *\*\*kwargs*)

Convert from YOLO format to DATALOOP format. Use this as *conversion\_func* param for functions that ask for this param.

**Prerequisites:** You must be an *owner* or *developer* to use this method.

**Parameters**

- **annotation** – annotations to convert
- **item** (`dtlpy.entities.item.Item`) – item entity
- **kwargs** – additional params

**Returns**

converted Annotation entity

**Return type**

`dtlpy.entities.annotation.Annotation`

**save\_to\_file**(*save\_to*, *to\_format*, *annotations*, *item=None*)

Save annotations to a file.

**Prerequisites:** You must be an *owner* or *developer* to use this method.

**Parameters**

- **save\_to** (*str*) – path to save the result to
- **to\_format** – AnnotationFormat to convert to – AnnotationFormat.COCO, AnnotationFormat.YOLO, AnnotationFormat.VOC, AnnotationFormat.DATALOOP
- **annotations** (*list*) – annotation list to convert
- **item** (`dtlpy.entities.item.Item`) – item entity

**static to\_coco**(*annotation*, *item=None*, *\*\*\_*)

Convert from DATALOOP format to COCO format. Use this as *conversion\_func* param for functions that ask for this param.

**Prerequisites:** You must be an *owner* or *developer* to use this method.

**Parameters**

- **annotation** (`dtlpy.entities.annotation.Annotation` or *dict*) – annotations to convert
- **item** (`dtlpy.entities.item.Item`) – item entity
- **\*\*\_** – additional params

**Returns**

converted Annotation

**Return type**

*dict*

**static to\_voc**(*annotation*, *item=None*, *\*\*\_*)

Convert from DATALOOP format to VOC format. Use this as *conversion\_func* param for functions that ask for this param.

**Prerequisites:** You must be an *owner* or *developer* to use this method.

**Parameters**

- **annotation** (`dtlpy.entities.annotation.Annotation` or *dict*) – annotations to convert
- **item** (`dtlpy.entities.item.Item`) – item entity
- **\*\*\_** – additional params

**Returns**

converted Annotation



**Return type**`dict`**to\_yolo**(*annotation*, *item=None*, *\*\*\_*)

Convert from DATALOOP format to YOLO format. Use this as `conversion_func` param for functions that ask for this param.

**Prerequisites:** You must be an *owner* or *developer* to use this method.

**Parameters**

- **annotation** (`dtlpy.entities.annotation.Annotation` or `dict`) – annotations to convert
- **item** (`dtlpy.entities.item.Item`) – item entity
- **\*\*\_** – additional params

**Returns**

converted Annotation

**Return type**`tuple`

**upload\_local\_dataset**(*from\_format: AnnotationFormat*, *dataset*, *local\_items\_path: Optional[str] = None*, *local\_labels\_path: Optional[str] = None*, *local\_annotations\_path: Optional[str] = None*, *only\_bbox: bool = False*, *filters=None*, *remote\_items=None*)

Convert and upload local dataset to dataloop platform.

**Prerequisites:** You must be an *owner* or *developer* to use this method.

**Parameters**

- **from\_format** (`str`) – AnnotationFormat to convert to – AnnotationFormat.COCO, AnnotationFormat.YOLO, AnnotationFormat.VOC, AnnotationFormat.DATALOOP
- **dataset** (`dtlpy.entities.dataset.Dataset`) – dataset entity
- **local\_items\_path** (`str`) – path to items to upload
- **local\_annotations\_path** (`str`) – path to annotations to upload
- **local\_labels\_path** (`str`) – path to labels to upload
- **only\_bbox** (`bool`) – only for coco datasets, if True upload only bbox
- **filters** (`dtlpy.entities.filters.Filters`) – Filters entity or a dictionary containing filter parameters
- **remote\_items** (`list`) – list of the items to upload

**Returns**

the error log file path if there are errors



## TUTORIALS

### 5.1 Data Management Tutorial

Tutorials for data management

#### 5.1.1 Cloud Storage

Setup integration with GCS/S3/Azure

##### Create an External Dataset

Setup integration with GCS/S3/Azure

##### Connect Cloud Storage

If you already have your data managed and organized on a cloud storage service, such as GCS/S3/Azure, you may want to utilize that with Dataloop, and not upload the binaries and create duplicates.

##### Cloud Storage Integration

Access & Permissions - Creating an integration with GCS/S2/Azure cloud requires adding a key/secret with the following permissions:

List (Mandatory) - allowing Dataloop to list all of the items in the storage. Get (Mandatory) - get the items and perform pre-process functionalities like thumbnails, item info etc. Put / Write (Mandatory) - lets you upload your items directly to the external storage from the Dataloop platform. Delete - lets you delete your items directly from the external storage using the Dataloop platform.

## Create Integration With GCS

Creating an integration GCS requires having JSON file with GCS configuration.

```
import dtlpy as dl
if dl.token_expired():
    dl.login()
organization = dl.organizations.get(organization_name=org_name)
with open(r"C:\gcsfile.json", 'r') as f:
    gcs_json = json.load(f)
gcs_to_string = json.dumps(gcs_json)
organization.integrations.create(name='gcsintegration',
                                integrations_type=dl.ExternalStorage.GCS,
                                options={'key': '',
                                         'secret': '',
                                         'content': gcs_to_string})
```

## Create Integration With S3

```
import dtlpy as dl
if dl.token_expired():
    dl.login()
organization = dl.organizations.get(organization_name='my-org')
organization.integrations.create(name='S3integration', integrations_type=dl.
    ↪ ExternalStorage.S3,
                                options={'key': "my_key", 'secret': "my_secret"})
```

## Create Integration With Azure

```
import dtlpy as dl
if dl.token_expired():
    dl.login()
organization = dl.organizations.get(organization_name='my-org')
organization.integrations.create(name='azureintegration',
                                integrations_type=dl.ExternalStorage.AZUREBLOB,
                                options={'key': 'my_key',
                                         'secret': 'my_secret',
                                         'clientId': 'my_clientId',
                                         'tenantId': 'my_tenantId'})
```

## Storage Driver

Once you have an integration, you can set up a driver, which adds a specific bucket (and optionally with a specific path/folder) as a storage resource.

### Create Drivers in the Platform (browser)

```
# param name: the driver name
# param driver_type: ExternalStorage.S3, ExternalStorage.GCS , ExternalStorage.AZUREBLOB
# param integration_id: the integration id
# param bucket_name: the external bucket name
# param project_id:
# param allow_external_delete:
# param region: relevant only for s3 - the bucket region
# param storage_class: relevant only for s3
# param path: Optional. By default, path is the root folder. Path is case sensitive.
# return: driver object
import dtlpy as dl
project = dl.projects.get('project_name')
driver = project.drivers.create(name='driver_name',
                               driver_type=dl.ExternalStorage.S3,
                               integration_id='integration_id',
                               bucket_name='bucket_name',
                               allow_external_delete=True,
                               region='eu-west-1',
                               storage_class="",
                               path="")
```

Once the integration and drivers are ready, you can create a Dataloop Datasets and sync all the data:

```
# create a dataset from a driver name, you can also create by the driver ID
import dtlpy as dl
project = dl.Project
dataset = project.datasets.create(dataset_name=dataset_name,
                                  driver=driver)
dataset.sync()
```

## AWS Binding with Lambda

Create a Lambda to sync a Bucket with Dataloop's Dataset

### Create an AWS Lambda to Continuously Sync a Bucket with Dataloop's Dataset

If you want to catch events from the AWS bucket and update the Dataloop Dataset you need to set up a Lambda. The Lambda will catch the AWS bucket events and will reflect them into the Dataloop Platform.

We have prepared an environment zip file with our SDK for python3.8 so you don't need to create anything else to use dtlpy in the lambda.

NOTE: For any other custom use (e.g other python version or more packages) try creating your own layer (We used [this](#) tutorial and the python:3.8 docker image).

### Create the Lambda

1. Create a new Lambda
2. **The default timeout is 3[s] so we'll need to change to 1[m]:**  
Configuration → General configuration → Edit → Timeout
3. Copy the following code:

```
import os
import urllib.parse
# Set dataloop path to tmp (to read/write from the lambda)
os.environ["DATALOOP_PATH"] = "/tmp"
import dtlpy as dl
DATASET_ID = ''
DTLPY_USERNAME = ''
DTLPY_PASSWORD = ''
def lambda_handler(event, context):
    dl.login_m2m(email=DTLPY_USERNAME, password=DTLPY_PASSWORD)
    dataset = dl.datasets.get(dataset_id=DATASET_ID,
                              fetch=False # to avoid GET the dataset each time
                              )
    for record in event['Records']:
        # Get the bucket name
        bucket = record['s3']['bucket']['name']
        # Get the file name
        filename = urllib.parse.unquote_plus(record['s3']['object']['key'], encoding=
        ↪ 'utf-8')
        if 'ObjectRemoved' in record['eventName']:
            # On delete event - delete the item from Dataloop
            try:
                dtlpy_filename = '/' + filename
                filters = dl.Filters(field='filename', values=dtlpy_filename)
                dataset.items.delete(filters=filters)
            except Exception as e:
                raise e
        elif 'ObjectCreated' in record['eventName']:
            # On create event - add a new item to the Dataset
```

(continues on next page)

(continued from previous page)

```

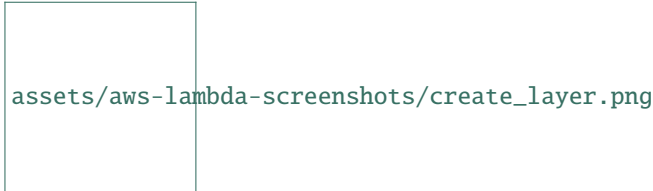
try:
    # upload the file
    path = 'external://' + filename
    # dataset.items.upload(local_path=path, overwrite=True) # if overwrite_
→ is required
    dataset.items.upload(local_path=path)
except Exception as e:
    raise e

```

## Add a Layer to the Lambda

We have created an AWS Layer with the Dataloop SDK ready. Click [here](#) to download the zip file. Because the layer's size is larger than 50MB you cannot use it directly (AWS restrictions), but need to upload it to a bucket first. Once uploaded, create a new layer for the dtlpy env:

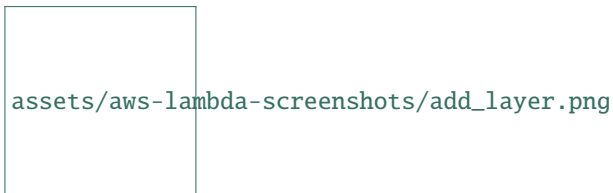
1. Go to the layers screen and “click Add Layer”.



2. Choose a name (dtlpy-env).
3. Use the link to the bucket layer.zip.
4. Select the env (x86\_64, python3.8).
5. Click “Create” and the bottom on the page.

Go back to your lambda and add the layer:

1. Select the “Add Layer”.



2. Choose “Custom layer” and select the Layer you’ve added and the version.
3. click “Add” at the bottom.

## Create the Bucket Events

Go to the bucket you are using, and create the event:

1. Go to Properties → Event notifications → Create event notification
2. Choose a name for the Event
3. For Event types choose: All object create events, All object delete events
4. Destination - Lambda function → Choose from your Lambda functions → choose the function you build → SAVE

Deploy and you're good to go!

## 5.1.2 Manage Datasets

Create and manage Datasets and connect them with your cloud storage

### Manage Datasets

Datasets are buckets in the dataloop system that hold a collection of data items of any type, regardless of their storage location (on Dataloop storage or external cloud storage).

### Create Dataset

You can create datasets within a project. There are no limits to the number of dataset a project can have, which correlates with data versioning where datasets can be cloned and merged.

```
dataset = project.datasets.create(dataset_name='my-dataset-name')
```

### Create Dataset With Cloud Storage Driver

If you've created an integration and driver to your cloud storage, you can create a dataset connected to that driver. A single integration (for example: S3) can have multiple drivers (per bucket or even per folder), so you need to specify that.

```
project = dl.projects.get(project_name='my-project-name')
# Get your drivers list
project.drivers.list().print()
# Create a dataset from a driver name. You can also create by the driver ID.
dataset = project.datasets.create(driver='my_driver_name', dataset_name="my_dataset_name
→")
```



## Retrieve Datasets

You can read all datasets that exist in a project, and then access the datasets by their ID (or name).

```
datasets = project.datasets.list()
dataset = project.datasets.get(dataset_id='my-dataset-id')
```

## Create Directory

A dataset can have multiple directories, allowing you to manage files by context, such as upload time, working batch, source, etc.

```
dataset.items.make_dir(directory="/directory/name")
```

## Hard-copy a Folder to Another Dataset

You can create a clone of a folder into a new dataset, but if you want to actually move between datasets a folder with files that are stored in the Dataloop system, you'll need to download the files and upload again to the destination dataset.

```
copy_annotations = True
flat_copy = False # if true, it copies all dir files and sub dir files to the
↳ destination folder without sub directories
source_folder = '/source_folder'
destination_folder = '/destination_folder'
source_project_name = 'source_project_name'
source_dataset_name = 'source_dataset_name'
destination_project_name = 'destination_project_name'
destination_dataset_name = 'destination_dataset_name'
# Get source project dataset
project = dl.projects.get(project_name=source_project_name)
dataset_from = project.datasets.get(dataset_name=source_dataset_name)
source_folder = source_folder.rstrip('/')
# Filter to get all files of a specific folder
filters = dl.Filters()
filters.add(field='filename', values=source_folder + '/*') # Get all items in folder
↳ (recursive)
pages = dataset_from.items.list(filters=filters)
# Get destination project and dataset
project = dl.projects.get(project_name=destination_project_name)
dataset_to = project.datasets.get(dataset_name=destination_dataset_name)
# Go over all projects and copy file from src to dst
for page in pages:
    for item in page:
        # Download item (without save to disk)
        buffer = item.download(save_locally=False)
        # Give the item's name to the buffer
        if flat_copy:
            buffer.name = item.name
        else:
            buffer.name = item.filename[len(source_folder) + 1:]
```

(continues on next page)

(continued from previous page)

```
# Upload item
print("Going to add {} to {} dir".format(buffer.name, destination_folder))
new_item = dataset_to.items.upload(local_path=buffer, remote_path=destination_
↪ folder)
if not isinstance(new_item, dl.Item):
    print('The file {} could not be upload to {}'.format(buffer.name, ↪
↪ destination_folder))
    continue
print("{} has been uploaded".format(new_item.filename))
if copy_annotations:
    new_item.annotations.upload(item.annotations.list())
```

## 5.1.3 Data Versioning

How to manage versions

### Data Versioning

Dataloop’s powerful data versioning provides you with unique tools for data management - clone, merge, slice & dice your files, to create multiple versions for various applications. Sample use cases include: Golden training sets management Reproducibility (dataset training snapshot) Experimentation (creating subsets from different kinds) Task/Assignment management Data Version “Snapshot” - Use our versioning feature as a way to save data (items, annotations, metadata) before any major process. For example, a snapshot can serve as a roll-back mechanism to original datasets in case of any error without losing the data.

### Clone Datasets

Cloning a dataset creates a new dataset with the same files as the original. Files are actually a reference to the original binary and not a new copy of the original, so your cloud data remains safe and protected. When cloning a dataset, you can add a destination dataset, remote file path, and more...

```
dataset = project.datasets.get(dataset_id='my-dataset-id')
dataset.clone(clone_name='clone-name',
              filters=None,
              with_items_annotations=True,
              with_metadata=True,
              with_task_annotations_status=True)
```

### Merge Datasets

Dataset merging outcome depends on how similar or different the datasets are.

- Cloned Datasets - items, annotations, and metadata will be merged. This means that you will see annotations from different datasets on the same item.
- Different datasets (not clones) with similar recipes - items will be summed up, which will cause duplication of similar items.
- Datasets with different recipes - Datasets with different default recipes cannot be merged. Use the ‘Switch recipe’ option on dataset level (3-dots action button) to match recipes between datasets and be able to merge them.

```
dataset_ids = ["dataset-1-id", "dataset-2-id"]
project_ids = ["dataset-1-project-id", "dataset-2-project-id"]
dataset_merge = dl.datasets.merge(merge_name="my_dataset-merge",
                                   project_ids=project_ids,
                                   dataset_ids=dataset_ids,
                                   with_items_annotations=True,
                                   with_metadata=False,
                                   with_task_annotations_status=False)
```

## 5.1.4 Upload and Manage Data and Metadata

Upload data items and metadata

### Upload & Manage Data & Metadata

#### Upload specific files

When you have specific files you want to upload, you can upload them all into a dataset using this script:

```
import dtlpy as dl
if dl.token_expired():
    dl.login()
project = dl.projects.get(project_name='project_name')
dataset = project.datasets.get(dataset_name='dataset_name')
dataset.items.upload(local_path=[r'C:/home/project/images/John Morris.jpg',
                                 r'C:/home/project/images/John Benton.jpg',
                                 r'C:/home/project/images/Liu Jinli.jpg'],
                    remote_path='/folder_name') # Remote path is optional, images will
↳ go to the main directory by default
```

#### Upload all files in a folder

If you want to upload all files from a folder, you can do that by just specifying the folder name:

```
import dtlpy as dl
if dl.token_expired():
    dl.login()
project = dl.projects.get(project_name='project_name')
dataset = project.datasets.get(dataset_name='dataset_name')
dataset.items.upload(local_path=r'C:/home/project/images',
                    remote_path='/folder_name') # Remote path is optional, images will
↳ go to the main directory by default
```

## Upload items from URL link

You can provide Dataloop with the link to the item, and not necessarily the item itself.

```
dataset = project.datasets.get(dataset_name='dataset_name')
url_path = 'http://ww.some_website/beautiful_flower.jpg'
# Create link
link = dl.UrlLink(ref=url_path, mimetype='image', name='file_name.jpg')
# Upload link
item = dataset.items.upload(local_path=link)
```

You can open an item uploaded to Dataloop by opening it in a viewer.

```
show
item.open_in_web()
```

## Additional upload options

Additional upload options include using buffer, pillow, openCV, and NdArray - see our complete documentation for code examples.

## Upload Items and Annotations Metadata

You can upload items as a table using a pandas data frame that will let you upload items with info (annotations, metadata such as confidence, filename, etc.) attached to it.

```
import pandas
import dtlpy as dl
dataset = dl.datasets.get(dataset_id='id') # Get dataset
to_upload = list()
# First item and info attached:
to_upload.append({'local_path': r"E:\TypesExamples\000000000064.jpg", # Item file path
                  'local_annotations_path': r"E:\TypesExamples\0000000000776.json", #
↳Annotations file path
                  'remote_path': "/first", # Dataset folder to upload the item to
                  'remote_name': 'f.jpg', # Dataset folder name
                  'item_metadata': {'user': {'dummy': 'fir'}}}) # Added user metadata
# Second item and info attached:
to_upload.append({'local_path': r"E:\TypesExamples\0000000000776.jpg", # Item file path
                  'local_annotations_path': r"E:\TypesExamples\0000000000776.json", #
↳Annotations file path
                  'remote_path': "/second", # Dataset folder to upload the item to
                  'remote_name': 's.jpg', # Dataset folder name
                  'item_metadata': {'user': {'dummy': 'sec'}}}) # Added user metadata
df = pandas.DataFrame(to_upload) # Make data into table
items = dataset.items.upload(local_path=df,
                             overwrite=True) # Upload table to platform
```

## 5.1.5 Upload and Manage Annotations

Upload annotations into data items

### Upload & Manage Annotations

```
import dtlpy as dl
item = dl.items.get(item_id='')
annotation = item.annotations.get(annotation_id='')
annotation.metadata["user"] = True
annotation.update()
```

### Upload User Metadata

To upload annotations from JSON and include the user metadata, add the parameter `local_annotation_path` to the `dataset.items.upload` function, like so:

```
project = dl.projects.get(project_name='project_name')
dataset = project.datasets.get(dataset_name='dataset_name')
dataset.items.upload(local_path=r'<items path>',
                    local_annotations_path=r'<annotation json file path>',
                    item_metadata=dl.ExportMetadata.FROM_JSON,
                    overwrite=True)
```

### Convert Annotations To COCO Format

```
converter = dl.Converter()
converter.upload_local_dataset(
    from_format=dl.AnnotationFormat.COCO,
    dataset=dataset,
    local_items_path=r'C:/path/to/items',
    # Please make sure the names of the items are the same as written in the COCO JSON
    ↪ file
    local_annotations_path=r'C:/path/to/annotations/file/coco.json'
)
```

### Upload Entire Directory and their Corresponding Dataloop JSON Annotations

```
# Local path to the items folder
# If you wish to upload items with your directory tree use : r'C:/home/project/images_
↪ folder'
local_items_path = r'C:/home/project/images_folder/*'
# Local path to the corresponding annotations - make sure the file names fit
local_annotations_path = r'C:/home/project/annotations_folder'
dataset.items.upload(local_path=local_items_path,
                    local_annotations_path=local_annotations_path)
```

## Upload Annotations To Video Item

Uploading annotations to video items needs to consider spanning between frames, and toggling visibility (occlusion). In this example, we will use the following CSV file. In this file there is a single ‘person’ box annotation that begins on frame number 20, disappears on frame number 41, reappears on frame number 51 and ends on frame number 90.

Video\_annotations\_example.CSV

```
import pandas as pd
# Read CSV file
df = pd.read_csv(r'C:/file.csv')
# Get item
item = dataset.items.get(item_id='my_item_id')
builder = item.annotations.builder()
# Read line by line from the csv file
for i_row, row in df.iterrows():
    # Create box annotation from csv rows and add it to a builder
    builder.add(annotation_definition=dl.Box(top=row['top'],
                                             left=row['left'],
                                             bottom=row['bottom'],
                                             right=row['right'],
                                             label=row['label']),
               object_visible=row['visible'], # Support hidden annotations on the
↪ visible row
               object_id=row['annotation id'], # Numbering system that separates
↪ different annotations
               frame_num=row['frame'])
# Upload all created annotations
item.annotations.upload(annotations=builder)
```

## Set Attributes On Annotations

You can set attributes on annotations in the platform using the SDK. Since Dataloop deprecated a legacy attributes mechanism, attributes are referred to as ‘2.0’ version and need to be set as such first.

### Free Text Attribute

```
dl.use_attributes_2(True)
annotation.attributes.update({"ID of the attribute": "value of the attribute"})
annotation = annotation.update(True)
```

### Range Attributes (Slider in UI)

```
dl.use_attributes_2(True)
annotation.attributes.update({"<attribute-id>": number_on_range})
annotation = annotation.update(system_metadata=True)
```

### CheckBox Attribute (Multiple choice)

```
dl.use_attributes_2(True)
annotation.attributes.update({"<attribute-id>": ["selection", "selection"]})
annotation = annotation.update(system_metadata=True)
```

### Radio Button Attribute (Single Choice)

```
dl.use_attributes_2(True)
annotation.attributes.update({"<attribute-id>": "selection"})
annotation = annotation.update(system_metadata=True)
```

### Yes/No Attribute

```
dl.use_attributes_2(True)
annotation.attributes.update({"<attribute-id>": True / False})
annotation = annotation.update(system_metadata=True)
```

### Show Annotations Over Image

After uploading items and annotations with their metadata, you might want to see some of them and perform visual validation.

To see only the annotations, use the annotation type *show* option.

```
# Use the show function for all annotation types
box = dl.Box()
# Must provide all inputs
box.show(image='',
          thickness='',
          with_text='',
          height='',
          width='',
          annotation_format='',
          color='')

```

To see the item itself with all annotations, use the Annotations option.

```
# Must input an image or height and width
annotation.show(image='',
                 height='', width='',
                 annotation_format='dl.ViewAnnotationOptions.*',
                 thickness='',
                 with_text='')

```

## Download Data, Annotations & Metadata

The item ID for a specific file can be found in the platform UI - Click BROWSE for a dataset, click on the selected file, and the file information will be displayed in the right-side panel. The item ID is detailed, and can be copied in a single click.

## Download Items and Annotations

Download dataset items and annotations to your computer folder in two separate folders. See all annotation options [here](#).

```
dataset.download(local_path=r'C:/home/project/images', # The default value is ".dataloop
↳ " folder
                  annotation_options=dl.VIEW_ANNOTATION_OPTIONS_JSON)
```

## Multiple Annotation Options

See all annotation options [here](#).

```
dataset.download(local_path=r'C:/home/project/images', # The default value is ".dataloop
↳ " folder
                  annotation_options=[dl.VIEW_ANNOTATION_OPTIONS_MASK,
                                      dl.VIEW_ANNOTATION_OPTIONS_JSON,
                                      dl.ViewAnnotationOptions.INSTANCE])
```

## Filter by Item and/or Annotation

- **Items filter** - download filtered items based on multiple parameters, like their directory. You can also download items based on different filters. Learn all about item filters [here](#).
- **Annotation filter** - download filtered annotations based on multiple parameters like their label. You can also download items annotations based on different filters, learn all about annotation filters [here](#). This example will download items and JSONS from a dog folder of the label 'dog'.

```
# Filter items from "folder_name" directory
item_filters = dl.Filters(resource='items', field='dir', values='/dog_name')
# Filter items with dog annotations
annotation_filters = dl.Filters(resource=dl.FiltersResource.ANNOTATION, field='label',
↳ values='dog')
dataset.download(local_path=r'C:/home/project/images', # The default value is ".dataloop
↳ " folder
                  filters=item_filters,
                  annotation_filters=annotation_filters,
                  annotation_options=dl.VIEW_ANNOTATION_OPTIONS_JSON)
```



## Filter by Annotations

- **Annotation filter** - download filtered annotations based on multiple parameters like their label. You can also download items annotations based on different filters, learn all about annotation filters [here](#).

```
item = dataset.items.get(item_id="item_id") # Get item from dataset to be able to view
↳ the dataset colors on Mask
# Filter items with dog annotations
annotation_filters = dl.Filters(resource='annotations', field='label', values='dog')
item.download(local_path=r'C:/home/project/images', # the default value is ".dataloop"
↳ folder
               annotation_filters=annotation_filters,
               annotation_options=dl.VIEW_ANNOTATION_OPTIONS_JSON)
```

## Download Annotations in COCO Format

- **Items filter** - download filtered items based on multiple parameters like their directory. You can also download items based on different filters, learn all about item filters [here](#).
- **Annotation filter** - download filtered annotations based on multiple parameters like their label. You can also download items annotations based on different filters, learn all about annotation filters [here](#).

This example will download COCO from a dog items folder of the label 'dog'.

```
# Filter items from "folder_name" directory
item_filters = dl.Filters(resource='items', field='dir', values='/dog_name')
# Filter items with dog annotations
annotation_filters = dl.Filters(resource='annotations', field='label', values='dog')
converter = dl.Converter()
converter.convert_dataset(dataset=dataset,
                        to_format='coco',
                        local_path=r'C:/home/coco_annotations',
                        filters=item_filters,
                        annotation_filters=annotation_filters)
```

## 5.1.6 Sort and Filters

DQL Filters a Pagination

### Advance SDK Filters

More complex filters on items and annotations

To access the filters entity click .

## Filter Operators

To understand more about filter operators please click .

When adding a filter, several operators are available for use:

### Equal

eq -> equal (or `dl.FiltersOperation.EQUAL`)

For example, filter items from a specific folder directory.

```
import dtlpy as dl
# Get project and dataset
project = dl.projects.get(project_name='project_name')
dataset = project.datasets.get(dataset_name='dataset_name')
# Create filters instance
filters = dl.Filters()
# Filter only items from a specific folder directory
filters.add(field='dir', values='/DatasetFolderName', operator=dl.FILTERS_OPERATIONS_
↳ EQUAL)
# optional - return results sorted by ascending file name
filters.sort_by(field='filename')
# Get filtered items list in a page object
pages = dataset.items.list(filters=filters)
# Count the items
print('Number of items in dataset: {}'.format(pages.items_count))
```

### Not Equal

ne -> not equal (or `dl.FiltersOperation.NOT_EQUAL`)

In this example, you will get all items that do not have ONLY a 'cat' label.

```
filters = dl.Filters()
# Filter ONLY a cat label
filters.add_join(field='label', values='cat', operator=dl.FILTERS_OPERATIONS_NOT_EQUAL)
# optional - return results sorted by ascending file name
filters.sort_by(field='filename')
# Get filtered items list in a page object
pages = dataset.items.list(filters=filters)
# Count the items
print('Number of items in the dataset: {}'.format(pages.items_count))
```

## Greater Than

gt -> greater than (or dl.FiltersOperation.GREATER\_THAN)

You will get items with a greater height (in pixels) than the given value in this example.

```
filters = dl.Filters()
# Filter images with a bigger height size
filters.add(field='metadata.system.height', values=height_number_in_pixels,
            operator=dl.FILTERS_OPERATIONS_GREATER_THAN)
# optional - return results sorted by ascending file name
filters.sort_by(field='filename')
# Get filtered items list in a page object
pages = dataset.items.list(filters=filters)
# Count the items
print('Number of items in dataset: {}'.format(pages.items_count))
```

## Less Than

lt -> less than (or dl.FiltersOperation.LESS\_THAN)

You will get items with a width (in pixels) less than the given value in this example.

```
filters = dl.Filters()
# Filter images with a bigger height size
filters.add(field='metadata.system.width', values=width_number_in_pixels, operator=dl.
FILTERS_OPERATIONS_LESS_THAN)
# optional - return results sorted by ascending file name
filters.sort_by(field='filename')
# Get filtered items list in a page object
pages = dataset.items.list(filters=filters)
# Count the items
print('Number of items in dataset: {}'.format(pages.items_count))
```

## In a List

in -> is in a list (when using this expression, values should be a list). (or dl.FiltersOperation.IN) In this example, you will get items with dog OR cat labels.

```
filters = dl.Filters()
# Filter items with dog OR cat labels
filters.add_join(field='label', values=['dog', 'cat'], operator=dl.FILTERS_OPERATIONS_IN)
# optional - return results sorted by ascending file name
filters.sort_by(field='filename')
# Get filtered items list in a page object
pages = dataset.items.list(filters=filters)
# Count the items
print('Number of items in dataset: {}'.format(pages.items_count))
```

## Exist

The filter param `FILTERS_OPERATIONS_EXISTS` checks if an attribute exists. The following example checks if there is an item with user metadata:

```
filters = dl.Filters()
filters.add(field='metadata.user', values=True, operator=dl.FILTERS_OPERATIONS_EXISTS)
dataset.items.list(filters=filters)
```

## SDK defaults

Filters ignore SDK defaults like hidden items and directories or note annotations as issues. If you wish to change this behavior, you may do the following:

```
filters = dl.Filters(use_defaults=False)
```

## Hidden Items and Directories

If you wish to only show hidden items & directories in your filters use this code:

```
filters = dl.Filters()
filters.add(field='type', values='dir')
# or
filters.pop(field='type')
```

## Delete a Filter

```
filters = dl.Filters()
# For example, if you added the following filter:
filters.add(field='to-delete-field', values='value')
# Use this command to delete the filter
filters.pop(field='to-delete-field')
# or for items by their annotations
filters.pop_join(field='to-delete-annotation-field')
```

## Full Examples

### How to filter items that were created between specific dates?

In this example, you will get all of the items that were created in 2018.

```
import datetime, time
filters = dl.Filters()
# -- time filters -- must be in ISO format and in UTC (offset from local time).
↳ converting using datetime package as follows:
earlier_timestamp = datetime.datetime(year=2018, month=1, day=1, hour=0, minute=0,
↳ second=0,
```

(continues on next page)

(continued from previous page)

```

                                tzinfo=datetime.timezone(
                                    datetime.timedelta(seconds=-time.timezone))).
↪isoformat()
later_timestamp = datetime.datetime(year=2019, month=1, day=1, hour=0, minute=0,
↪second=0,
                                tzinfo=datetime.timezone(
                                    datetime.timedelta(seconds=-time.timezone))).
↪isoformat()
filters.add(field='createdAt', values=earlier_timestamp, operator=dl.FiltersOperations.
↪GREATER_THAN)
filters.add(field='createdAt', values=later_timestamp, operator=dl.FiltersOperations.
↪LESS_THAN)
# change method to OR
filters.method = dl.FiltersMethod.OR
# Get filtered items list in a page object
pages = dataset.items.list(filters=filters)
# Count the items
print('Number of items in dataset: {}'.format(pages.items_count))

```

### How to filter items that don't have a specific label?

In this example, you will get all items that do not have a 'cat' label AT ALL.

```

# Get all items
all_items = set([item.id for item in dataset.items.list().all()])
# Get all items WITH the label cat
filters = dl.Filters()
filters.add_join(field='label', values='cat')
cat_items = set([item.id for item in dataset.items.list(filters=filters).all()])
# Get the difference between the sets. This will give you a list of the items with no cat
no_cat_items = all_items.difference(cat_items)
print('Number of filtered items in dataset: {}'.format(len(no_cat_items)))
# Iterate through the ID's - Go over all ID's and print the matching item
for item_id in no_cat_items:
    print(dataset.items.get(item_id=item_id))

```

## **Annotation Level Filters**

Create filter on annotations, use DQL on an annotation level attributes

To access the filters entity click .

## **The Dataloop Query Language - DQL**

Using The , you may navigate through massive amounts of data.

You can *filter*, *sort*, and *update* your metadata with it.

## **Filters**

Using filters, you can filter items and get a generator of the filtered items. The filters entity is used to build such filters.

### **Filters - Field & Value**

Filter your items or annotations using the parameters in the JSON code that represent its data within our system. Access your item/annotation JSON using .

#### **Field**

Field refers to the attributes you filter by.

For example, “dir” would be used if you wish to filter items by their folder/directory.

#### **Value**

Value refers to the input by which you want to filter. For example, “/new\_folder” can be the directory/folder name where the items you wish to filter are located.

### **Sort - Field & Value**

#### **Field**

Field refers to the field you sort your items/annotations list by. For example, if you sort by filename, you will get the item list sorted in alphabetical order by filename. See the full list of the available fields .

## Value

Value refers to the list order direction. Either ascending or descending.

## Filter Annotations

Filter annotations by the annotations' JSON fields. In this example, you will get all of the note annotations in the dataset sorted by the label.

```
import dtlpy as dl
# Get project and dataset
project = dl.projects.get(project_name='project_name')
dataset = project.datasets.get(dataset_name='dataset_name')
# Create filters instance with annotation resource
filters = dl.Filters(resource=dl.FiltersResource.ANNOTATION)
# Filter example - only note annotations
filters.add(field='type', values='note')
# optional - return results sorted by descending label
filters.sort_by(field='label', value=dl.FiltersOrderByDirection.DESENDING)
pages = dataset.annotations.list(filters=filters)
# Count the annotations
print('Number of filtered annotations in dataset: {}'.format(pages.items_count))
# Iterate through the annotations - Go over all annotations and print the properties
for page in pages:
    for annotation in page:
        annotation.print()
```

## Filter Annotations by the Annotations' Item

- filter Annotations by the annotations' items' JSON fields. For example, filter only box annotations from image items.

```
# Create filters instance
filters = dl.Filters(resource=dl.FiltersResource.ANNOTATION)
# Filter all box annotations
filters.add(field='type', values='box')
# AND filter annotations by their items - only items that are of mimetype image
# Meaning you will get 'box' annotations of all image items
filters.add_join(field='metadata.system.mimetype', values="image*")
# optional - return results sorted by descending creation date
filters.sort_by(field='createdAt', value=dl.FILTERS_ORDERBY_DIRECTION_DESCENDING)
# Get filtered annotations list in a page object
pages = dataset.annotations.list(filters=filters)
# Count the annotations
print('Number of filtered annotations in dataset: {}'.format(pages.items_count))
```

## Filters Method - “Or” and “And”

### And

If you wish to filter annotations with the “and” logical operator, you can do so by specifying which filters will be checked with “and”.

```
{
  "id": "5f576f660bb2fb455d79ffdf",
  "datasetId": "5e368bee106a76a61cf05282",
  "type": "segment",
  "label": "Planet",
  "attributes": [],
  "coordinates": [
    [
      {
        "x": 856.25,
        "y": 1031.2499999999995
      },
      {
        "x": 1081.25,
        "y": 1631.2499999999995
      },
      {
        "x": 485.41666666666663,
        "y": 1735.4166666666665
      },
      {
        "x": 497.91666666666663,
        "y": 1172.9166666666665
      }
    ]
  ],
  "metadata": {
    "system": {
      "status": null,
      "startTime": 0,
      "endTime": 1,
      "frame": 0,
      "endFrame": 1,
      "snapshots_": [
        {
          "fixed": true,
          "type": "transition",
          "frame": 0,
          "objectVisible": true,
          "data": [
            [
              {
                "x": 856.25,
                "y": 1031.2499999999995
              },
              {

```

(continues on next page)



(continued from previous page)

```

        "x": 1081.25,
        "y": 1631.2499999999995
    },
    {
        "x": 485.41666666666663,
        "y": 1735.4166666666665
    },
    {
        "x": 497.91666666666663,
        "y": 1172.9166666666665
    }
    ]
    ],
    "label": "Planet",
    "attributes": []
    }
    ],
    "automated": false,
    "isOpen": false,
    "system": false
    },
    "user": {}
    },
    "creator": "user@dataloop.ai",
    "createdAt": "2020-09-08T11:47:50.576Z",
    "updatedAt": "2020-09-08T11:47:50.576Z",
    "updatedBy": "user@dataloop.ai",
    "updatedAt": "2020-09-08T11:47:50.576Z",
    "itemId": "5f572f4423a69b8c83408f12",
    "url": "https://gate.dataloop.ai/api/v1/annotations/5f576f660bb2fb455d79ffdf",
    "item": "https://gate.dataloop.ai/api/v1/items/5f572f4423a69b8c83408f12",
    "dataset": "https://gate.dataloop.ai/api/v1/datasets/5e368bee106a76a61cf05282",
    "hash": "11fdc816804faf0f7266b40d1cb67aff38e5c10d"
}

```

## Full Examples

### How to filter annotations by their label?

```

filters = dl.Filters()
# set resource
filters.resource = dl.FiltersResource.ANNOTATION
filters.add(field='label', values='your_label_value')
pages = dataset.annotations.list(filters=filters)
# Count the annotations
print('Number of filtered annotations in dataset: {}'.format(pages.items_count))

```

## Advanced Filtering Operators

Explore advanced filtering options on [dtlpy](#).

### Item Level

Create filter on items, use DQL on an item level attributes

To access the filters entity click [here](#).

## The Dataloop Query Language - DQL

Using The [DQL](#), you may navigate through massive amounts of data.

You can *filter*, *sort*, and *update* your metadata with it.

### Filters

Using filters, you can filter items and get a generator of the filtered items. The filters entity is used to build such filters.

### Filters - Field & Value

Filter your items or annotations using the parameters in the JSON code that represent its data within our system. Access your item/annotation JSON using [dtlpy](#).

#### Field

Field refers to the attributes you filter by.

For example, “dir” would be used if you wish to filter items by their folder/directory.

#### Value

Value refers to the input by which you want to filter. For example, “/new\_folder” can be the directory/folder name where the items you wish to filter are located.

### Sort - Field & Value

#### Field

Field refers to the field you sort your items/annotations list by. For example, if you sort by filename, you will get the item list sorted in alphabetical order by filename. See the full list of the available fields [here](#).

## Value

Value refers to the list order direction. Either ascending or descending.

## Filter Items

Filter items by the item's JSON fields. In this example, you will get all annotated items in a dataset sorted by the filename.

```
import dtlpy as dl
# Get project and dataset
project = dl.projects.get(project_name='project_name')
dataset = project.datasets.get(dataset_name='dataset_name')
# Create filters instance
filters = dl.Filters()
# Filter only annotated items
filters.add(field='annotated', values=True)
# optional - return results sorted by ascending file name
filters.sort_by(field="filename")
# Get filtered items list in a page object
pages = dataset.items.list(filters=filters)
# Count the items
print('Number of items in dataset: {}'.format(pages.items_count))
```

## Filter Items by the Items' Annotations

- filter items by the items' annotations JSON fields. For example, filter only items with 'box' annotations.

```
filters = dl.Filters()
# Filter all approved items
filters.add(field='metadata.system.annotationStatus', values="approved")
# AND filter items by their annotation - only items with 'box' annotations
# Meaning you will get approved items with 'box' annotations
filters.add_join(field='type', values='box')
# optional - return results sorted by descending creation date
filters.sort_by(field='createdAt', value=dl.FILTERS_ORDERBY_DIRECTION_DESCENDING)
# Get filtered items list in a page object
pages = dataset.items.list(filters=filters)
# Count the items
print('Number of items in dataset: {}'.format(pages.items_count))
```

## Filters Method - “Or” and “And”

### And

If you wish to filter annotations with the “and” logical operator, you can do so by specifying which filters will be checked with “and”.

```
{
  "id": "5f4b60848ced1d50c3df114a",
  "datasetId": "5f4b603d9825b9f191bbd3b3",
  "createdAt": "2020-08-30T08:17:08.000Z",
  "dir": "/new_folder",
  "filename": "/new_folder/optional.jpg",
  "type": "file",
  "hidden": false,
  "metadata": {
    "system": {
      "originalname": "file",
      "size": 3290035,
      "encoding": "7bit",
      "mimetype": "image/jpeg",
      "annotationStatus": [
        "completed"
      ],
    },
    "refs": [
      {
        "type": "task",
        "id": "5f4b61f8f81ab6238c331bd2"
      },
      {
        "type": "assignment",
        "id": "5f4b61f8f81ab60508331bd3"
      }
    ],
    "executionLogs": {
      "image-metadata-extractor": {
        "default_module": {
          "run": {
            "5f4b60841b892d82eaa2d95b": {
              "progress": 100,
              "status": "success"
            }
          }
        }
      }
    },
    "exif": {},
    "height": 2734,
    "width": 4096,
    "statusLog": [
      {
        "status": "completed",
        "timestamp": "2020-08-30T14:54:17.014Z",
```

(continues on next page)

(continued from previous page)

```

        "creator": "user@dataloop.ai",
        "action": "created"
    }
],
    "isBinary": true
}
},
    "name": "optional.jpg",
    "url": "https://gate.dataloop.ai/api/v1/items/5f4b60848ced1d50c3df114a",
    "dataset": "https://gate.dataloop.ai/api/v1/datasets/5f4b603d9825b9f191bbd3b3",
    "annotationsCount": 18,
    "annotated": "discarded",
    "stream": "https://gate.dataloop.ai/api/v1/items/5f4b60848ced1d50c3df114a/stream",
    "thumbnail": "https://gate.dataloop.ai/api/v1/items/5f4b60848ced1d50c3df114a/
↪thumbnail",
    "annotations": "https://gate.dataloop.ai/api/v1/items/5f4b60848ced1d50c3df114a/
↪annotations"
}

```

## Full Examples

### How to filter items by their annotations label?

```

filters = dl.Filters()
filters.add_join(field='label', values='your_label_value')
pages = dataset.items.list(filters=filters)
# Count the items
print('Number of filtered items in dataset: {}'.format(pages.items_count))

```

### How to filter items by completed and approved status?

```

filters = dl.Filters()
filters.add(field='metadata.system.annotationStatus', values=["completed", "approved"])
pages = dataset.items.list(filters=filters)
# Count the items
print('Number of items in dataset: {}'.format(pages.items_count))

```

### How to filter items by completed status (with items who are approved as well)?

```

filters = dl.Filters()
# set resource
filters.add(field='metadata.system.annotationStatus', values="completed")
pages = dataset.items.list(filters=filters)
# Count the items
print('Number of items in dataset: {}'.format(pages.items_count))

```

### How to filter items by only completed status?

```
filters = dl.Filters()
filters.add(field='metadata.system.annotationStatus', values=["completed"])
pages = dataset.items.list(filters=filters)
# Count the items
print('Number of items in dataset: {}'.format(pages.items_count))
```

### How to filter unassigned items?

```
filters = dl.Filters()
filters.add(field='metadata.system.refs', values=[])
pages = dataset.items.list(filters=filters)
# Count the items
print('Number of items in dataset: {}'.format(pages.items_count))
```

### How to filter items by a specific folder?

```
filters = dl.Filters()
filters.add(field='dir', values="/folderName")
pages = dataset.items.list(filters=filters)
# Count the items
print('Number of items in dataset: {}'.format(pages.items_count))
```

### Get all items named foo.bar

```
filters = dl.Filters()
filters.add(field='name', values='foo.bar.*')
# Get filtered item list in a page object
pages = dataset.items.list(filters=filters)
# Count the items
print('Number of filtered items in dataset: {}'.format(pages.items_count))
```

### Sort files of size 0-5 MB by name, in ascending order

```
filters = dl.Filters()
filters.add(field='metadata.system.size', values='0', operator='gt')
filters.add(field='metadata.system.size', values='5242880', operator='lt')
filters.sort_by(field='filename', value=dl.FILTERS_ORDERBY_DIRECTION_ASCENDING)
# Get filtered item list in a page object
pages = dataset.items.list(filters=filters)
# Count the items
print('Number of filtered items in dataset: {}'.format(pages.items_count))
```

## Sort with multiple fields: Sort Items by labels ascending and createdAt descending

```
filters = dl.Filters()
# set annotation resource
filters.resource = dl.FiltersResource.ANNOTATION
# return results sorted by descending label
filters.sort_by(field='label', value=dl.FILTERS_ORDERBY_DIRECTION_ASCENDING)
filters.sort_by(field='createdAt', value=dl.FILTERS_ORDERBY_DIRECTION_DESCENDING)
# Get filtered item list in a page object
pages = dataset.items.list(filters=filters)
# Count the items
print('Number of filtered items in dataset: {}'.format(pages.items_count))
```

## Advanced Filtering Operators

Explore advanced filtering options on [dtlpy](#).

## Response to DQL Query

A typical response to a DQL query will look like the following:

```
{
  "totalItemsCount": number,
  "items": Array,
  "totalPagesCount": number,
  "hasNextPage": boolean,
}
# A possible result:
{
  "totalItemsCount": 2,
  "totalPagesCount": 1,
  "hasNextPage": false,
  "items": [
    {
      "id": "5d0783852dbc15306a59ef6c",
      "createdAt": "2019-06-18T23:29:15.775Z",
      "filename": "/5546670769_8df950c6b6.jpg",
      "type": "file"
      // ...
    },
    {
      "id": "5d0783852dbc15306a59ef6d",
      "createdAt": "2019-06-19T23:29:15.775Z",
      "filename": "/5551018983_3ce908ac98.jpg",
      "type": "file"
      // ...
    }
  ]
}
```

## Pagination

How to use pages and iteration over items

## Pagination

### Pages

We use pages instead of a list when we have an object that contains a lot of information.

The page object divides a large list into pages (with a default of 1000 items) in order to save time when going over the items.

It is the same as we display it in the annotation platform, see example .

You can redefine the number of items on a page with the `page_size` attribute. When we go over the items we use nested loops to first go to the pages and then go over the items for each page.

### Iterator of Items

You can create a generator of items with different filters.

```
import dtlpy as dl
# Get the project
project = dl.projects.get(project_name='project_name')
# Get the dataset
dataset = project.datasets.get(dataset_name='dataset_name')
# Get items in pages (1000 item per page)
filters = dl.Filters()
filters.add(field='filename', values='/your/file/path.mimetype')
pages = dataset.items.list(filters=filters)
# Count the items
print('Number of items in dataset: {}'.format(pages.items_count))
# Go over all item and print the properties
for i_page, page in enumerate(pages):
    print('{} items in page {}'.format(len(page), i_page))
    for item in page:
        item.print()
```

A Page entity iterator also allows reverse iteration for cases in which you want to change items during the iteration:

```
# Go over all item and print the properties
for i_page, page in enumerate(reversed(pages)):
    print('{} items in page {}'.format(len(page), i_page))
```

If you want to iterate through all items within your filter, you can also do so without going through them page by page:

```
for item in pages.all():
    print(item.name)
```

If you are planning to do some process on each item, it's faster to use multi-threads (or multi-process) for parallel computation. The following uses `ThreadPoolExecutor` with 32 workers to process parallel batches of 32 items:



```

from concurrent.futures import ThreadPoolExecutor
def single_item(item):
    # do some work on item
    print(item.filename)
    return True
with ThreadPoolExecutor(max_workers=32) as executor:
    executor.map(single_item, pages.all())

```

Lets compare the runtime to see that now the process is faster:

```

from concurrent.futures import ThreadPoolExecutor
import time
tic = time.time()
for item in pages.all():
    # do stuff on item
    time.sleep(1)
print('One by one took {:.2f}[s]'.format(time.time() - tic))
def single_item(item):
    # do stuff on item
    time.sleep(1)
    return True
tic = time.time()
with ThreadPoolExecutor(max_workers=32) as executor:
    executor.map(single_item, pages.all())
print('Using threads took {:.2f}[s]'.format(time.time() - tic))

```

Visualizing the progress with tqdm progress bar:

```

import tqdm
pbar = tqdm.tqdm(total=pages.items_count)
def single_item(item):
    # do stuff on item
    time.sleep(1)
    pbar.update()
    return True
with ThreadPoolExecutor(max_workers=32) as executor:
    executor.map(single_item, pages.all())

```

## Set page\_size

The following example sets the page\_size to 50:

```

# Create filters instance
filters = dl.Filters()
# Get filtered item list in a page object, where the starting page is 1
pages = dataset.items.list(filters=filters, page_offset=1, page_size=50)
# Count the items
print('Number of filtered items in dataset: {}'.format(pages.items_count))
# Print items from page 1
print('Length of first page: {}'.format(len(pages.items)))

```

## 5.1.7 Working with Metadata

Working with Item's metadata

### Working with Metadata

```
import dtlpy as dl
# Get project and dataset
project = dl.projects.get(project_name='project_name')
dataset = project.datasets.get(dataset_name='dataset_name')
```

### User Metadata

As a powerful tool to manage data based on your categories and information, you can add any keys and values to both the item's and annotations' user-metadata sections using the Dataloop SDK. Then, you can use your user-metadata for data filtering, sorting, etc.

### Metadata Data Types

Metadata is a dictionary attribute used with items, annotations, and other entities of the Dataloop system (task, recipe, and more). As such, it can be used with string, number, boolean, list or null types.

#### String

```
item.metadata['user']['MyKey'] = 'MyValue'
annotation.metadata['user']['MyKey'] = 'MyValue'
```

#### Number

```
item.metadata['user']['MyKey'] = 3
annotation.metadata['user']['MyKey'] = 3
```

#### Boolean

```
item.metadata['user']['MyKey'] = True
annotation.metadata['user']['MyKey'] = True
```

### Null – add metadata with no information

```
item.metadata['user']['MyKey'] = None
annotation.metadata['user']['MyKey'] = None
```

### List

```
# add metadata of a list (can contain elements of different types).
item.metadata['user']['MyKey'] = ["A", 2, False]
annotation.metadata['user']['MyKey'] = ["A", 2, False]
```

### Add new metadata to a list without losing existing data

```
item.metadata['user']['MyKey'].append(3)
item = item.update()
annotation.metadata['user']['MyKey'].append(3)
annotation = annotation.update()
```

### Add metadata to an item's user metadata

```
# upload and claim item
item = dataset.items.upload(local_path=r'C:/home/project/images/item.mimetype')
# or get item
item = dataset.items.get(item_id='write-your-id-number')
# modify metadata
item.metadata['user'] = dict()
item.metadata['user']['MyKey'] = 'MyValue'
# update and reclaim item
item = item.update()
```

### Modify an existing user metadata field

```
# upload and claim item
item = dataset.items.upload(local_path=r'C:/home/project/images/item.mimetype')
# or get item
item = dataset.items.get(item_id='write-your-id-number')
# modify metadata
if 'user' not in item.metadata:
    item.metadata['user'] = dict()
item.metadata['user']['MyKey'] = 'MyValue'
# update and reclaim item
item = item.update()
```

## Add metadata to annotations' user metadata

```
# Get annotation
annotation = dl.annotations.get(annotation_id='my-annotation-id')
# modify metadata
annotation.metadata['user'] = dict()
item.metadata['user']['red'] = True
# update and reclaim annotation
annotation = annotation.update()
```

## Filter items by user metadata

### 1. Get your dataset

```
project = dl.projects.get(project_name='project_name')
dataset = project.datasets.get(dataset_name='dataset_name')
```

### 2. Add metadata to an item

You can also

```
# upload and claim item
item = dataset.items.upload(local_path=r'C:/home/project/images/item.mimetype')
# or get item
item = dataset.items.get(item_id='write-your-id-number')
# modify metadata
item.metadata['user'] = dict()
item.metadata['user']['MyKey'] = 'MyValue'
# update and reclaim item
item = item.update()
```

### 3. Create a filter

```
filters = dl.Filters()
# set resource - optional - default is item
filters.resource = dl.FiltersResource.ITEM
```

### 4. Filter by your written key

```
filters.add(field='metadata.user.Key', values='Value')
```

## 5. Get filtered items

```
pages = dataset.items.list(filters=filters)
# Go over all item and print the properties
for page in pages:
    for item in page:
        item.print()
```

## 5.2 FaaS Tutorial

Tutorials for FaaS

### 5.2.1 FaaS Interactive Tutorial – Using Python & Dataloop SDK

FaaS Interactive Tutorial

#### FaaS Interactive Tutorial – Using Python & Dataloop SDK

##### Concept

Dataloop Function-as-a-Service (FaaS) is a compute service that automatically runs your code based on time patterns or in response to trigger events.

You can use Dataloop FaaS to extend other Dataloop services with custom logic. Altogether, FaaS serves as a super flexible unit that provides you with increased capabilities in the Dataloop platform and allows achieving any need while automating processes.

With Dataloop FaaS, you simply upload your code and create your functions. Following that, you can define a time interval or specify a resource event for triggering the function. When a trigger event occurs, the FaaS platform launches and manages the compute resources, and executes the function.

You can configure the compute settings according to your preferences (machine types, concurrency, timeout, etc.) or use the default settings.

##### Use Cases

**Pre annotation processing:** Resize, video assembler, video dissembler

**Post annotation processing:** Augmentation, crop box-annotations, auto-parenting

**ML models:** Auto-detection

**QA models:** Auto QA, consensus model, majority vote model

## 5.2.2 Introduction

Getting started with FaaS.

### Introduction

This tutorial will help you get started with FaaS.

1. Prerequisites
2. Basic use case: Single function
  - Deploy a function as a service
  - Execute the service manually and view the output
1. Advance use case: Multiple functions
  - Deploy several functions as a package
  - Deploy a service of the package
  - Set trigger events to the functions
  - Execute the functions and view the output and logs

First, log in to the platform by running the following Python code in the terminal or your IDE:

```
import dtlpy as dl
if dl.token_expired():
    dl.login()
```

Your browser will open a login screen, allowing you to enter your credentials or log in with Google. Once the “Login Successful” tab appears, you are allowed to close it.

This tutorial requires a project. You can create a new project, or alternatively use an existing one:

```
# Create a new project
project = dl.projects.create(project_name='project-sdk-tutorial')
```

```
# Use an existing project
project = dl.projects.get(project_name='project-sdk-tutorial')
```

Let’s create a dataset to work with and upload a sample item to it:

```
dataset = project.datasets.create(dataset_name='dataset-sdk-tutorial')
item = dataset.items.upload(
    local_path=[
        'https://raw.githubusercontent.com/dataloop-ai/tiny_coco/master/images/train2017/
↪000000184321.jpg'],
    remote_path='/folder_name')
# Remote path is optional, images will go to the main directory by default
```

### 5.2.3 Run Your First Function

Create and run your first FaaS in the Dataloop platform

#### Basic Use Case: Single Function

##### Create and Deploy a Sample Function

Below is an image-manipulation function in Python to use for converting an RGB image to a grayscale image. The function receives a single item, which later can be used as a trigger to invoke the function:

```
def rgb2gray(item: dl.Item):
    """
    Function to convert RGB image to GRAY
    Will also add a modality to the original item
    :param item: dl.Item to convert
    :return: None
    """
    import numpy as np
    import cv2
    buffer = item.download(save_locally=False)
    bgr = cv2.imdecode(np.frombuffer(buffer.read(), np.uint8), -1)
    gray = cv2.cvtColor(bgr, cv2.COLOR_BGR2GRAY)
    bgr_equalized_item = item.dataset.items.upload(local_path=gray,
                                                    remote_path='/gray' + item.dir,
                                                    remote_name=item.name)

    # add modality
    item.modalities.create(name='gray',
                           ref=bgr_equalized_item.id)
    item.update(system_metadata=True)
```

You can now deploy the function as a service using Dataloop SDK. Once the service is ready, you may execute the available function on any input:

```
project = dl.projects.get(project_name='project-sdk-tutorial')
service = project.services.deploy(func=rgb2gray,
                                  service_name='grayscale-item-service')
```

#### Execute the function

An execution means running the function on a service with specific inputs (arguments). The execution input will be provided to the function that the execution runs.

Now that the service is up, it can be executed manually (on-demand) or automatically, based on a set trigger (time/event). As part of this tutorial, we will demonstrate how to manually run the “RGB to Gray” function.

To see the item we uploaded, run the following code:

```
item.open_in_web()
```

## 5.2.4 Multiple Functions and Modules

Create a Package with multiple functions and modules

### Multiple Functions

#### Create and Deploy a Package of Several Functions

First, login to the Dataloop platform:

```
import dtlpy as dl
if dl.token_expired():
    dl.login()
```

Let's define the project and dataset you will work with in this tutorial. create a new project and dataset:

```
project = dl.projects.create(project_name='project-sdk-tutorial')
project.datasets.create(dataset_name='dataset-sdk-tutorial')
```

To use an existing project and dataset:

```
project = dl.projects.get(project_name='project-sdk-tutorial')
dataset = project.datasets.get(dataset_name='dataset-sdk-tutorial')
```

### Write your code

The following code consists of two image-manipulation methods:

- RGB to grayscale over an image
- CLAHE Histogram Equalization over an image - Contrast Limited Adaptive Histogram Equalization (CLAHE) to equalize images

To proceed with this tutorial, copy the following code and save it as a main.py file.

```
import dtlpy as dl
import cv2
import numpy as np
class ImageProcess(dl.BaseServiceRunner):
    @staticmethod
    def rgb2gray(item: dl.Item):
        """
        Function to convert RGB image to GRAY
        Will also add a modality to the original item
        :param item: dl.Item to convert
        :return: None
        """
        buffer = item.download(save_locally=False)
        bgr = cv2.imdecode(np.frombuffer(buffer.read(), np.uint8), -1)
        gray = cv2.cvtColor(bgr, cv2.COLOR_BGR2GRAY)
        gray_item = item.dataset.items.upload(local_path=gray,
                                              remote_path='/gray' + item.dir,
                                              remote_name=item.filename)
```

(continues on next page)



(continued from previous page)

```

    # add modality
    item.modalities.create(name='gray',
                           ref=gray_item.id)
    item.update(system_metadata=True)
    @staticmethod
    def clahe_equalization(item: dl.Item):
        """
        Function to perform histogram equalization (CLAHE)
        Will add a modality to the original item
        Based on opencv https://docs.opencv.org/4.x/d5/daf/tutorial\_py\_histogram\_
        ↪equalization.html
        :param item: dl.Item to convert
        :return: None
        """
        buffer = item.download(save_locally=False)
        bgr = cv2.imdecode(np.frombuffer(buffer.read(), np.uint8), -1)
        # create a CLAHE object (Arguments are optional).
        lab = cv2.cvtColor(bgr, cv2.COLOR_BGR2LAB)
        lab_planes = cv2.split(lab)
        clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8, 8))
        lab_planes[0] = clahe.apply(lab_planes[0])
        lab = cv2.merge(lab_planes)
        bgr_equalized = cv2.cvtColor(lab, cv2.COLOR_LAB2BGR)
        bgr_equalized_item = item.dataset.items.upload(local_path=bgr_equalized,
                                                         remote_path='/equ' + item.dir,
                                                         remote_name=item.filename)

        # add modality
        item.modalities.create(name='equ',
                               ref=bgr_equalized_item.id)
        item.update(system_metadata=True)

```

## Define the module

Multiple functions may be defined in a single package under a “module” entity. This way you will be able to use a single codebase for various services.

Here, we will create a module containing the two functions we discussed. The “main.py” file you downloaded is defined as the module entry point. Later, you will specify its directory file path.

```

modules = [dl.PackageModule(name='image-processing-module',
                             entry_point='main.py',
                             class_name='ImageProcess',
                             functions=[dl.PackageFunction(name='rgb2gray',
                                                            description='Converting RGB to_
        ↪gray',
                                                            inputs=[dl.FunctionIO(type=dl.
        ↪PackageInputType.ITEM,
                                                            name=
        ↪'item')]]),
                             dl.PackageFunction(name='clahe_equalization',
                                                  description='CLAHE histogram_
        ↪equalization',

```

(continues on next page)

(continued from previous page)

```
inputs=[dl.FunctionIO(type=dl.  
↪PackageInputType.ITEM,  
↪'item'))]]  
]]]
```

## Push the package

When you deployed the service in the previous tutorial (“Single Function”), a module and a package were automatically generated.

Now we will explicitly create and push the module as a package in the Dataloop FaaS library (application hub). For that, please specify the source path (`src_path`) of the “main.py” file you downloaded, and then run the following code:

```
src_path = 'functions/opencv_functions'  
project = dl.projects.get(project_name='project-sdk-tutorial')  
package = project.packages.push(package_name='image-processing',  
                                modules=modules,  
                                src_path=src_path)
```

## Deploy a service

Now that the package is ready, it can be deployed to the Dataloop platform as a service. To create a service from a package, you need to define which module the service will serve. Notice that a service can only contain a single module. All the module functions will be automatically added to the service.

Multiple services can be deployed from a single package. Each service can get its own configuration: a different module and settings (computing resources, triggers, UI slots, etc.).

In our example, there is only one module in the package. Let’s deploy the service:

```
service = package.services.deploy(service_name='image-processing',  
                                  runtime=dl.KubernetesRuntime(concurrency=32),  
                                  module_name='image-processing-module')
```

## Trigger the service

Once the service is up, we can configure a trigger to automatically run the service functions. When you bind a trigger to a function, that function will execute when the trigger fires. The trigger is defined by a given time pattern or by an event in the Dataloop system.

Event based trigger is related to a combination of resource and action. A resource can be any entity in our system (item, dataset, annotation, etc.) and the associated action will define a change in the resource that will prompt the trigger (update, create, delete). You can only have one resource per trigger.

The resource object that triggered the function will be passed as the function’s parameter (input).

Let’s set a trigger in the event a new item is created:

```
filters = dl.Filters()
filters.add(field='datasetId', values=dataset.id)
trigger = service.triggers.create(name='image-processing2',
                                  function_name='clahe_equalization',
                                  execution_mode=dl.TriggerExecutionMode.ONCE,
                                  resource=dl.TriggerResource.ITEM,
                                  actions=dl.TriggerAction.CREATED,
                                  filters=filters)
```

In the defined filters we specified a dataset. Once a new item is uploaded (created) in this dataset, the CLAHE function will be executed for this item. You can also add filters to specify the item type (image, video, JSON, directory, etc.) or a certain format (jpeg, jpg, WebM, etc.).

A separate trigger must be set for each function in your service. Now, we will define a trigger for the second function in the module `rgb2gray`. Each time an item is updated, invoke the `rgb2gray` function:

```
trigger = service.triggers.create(name='image-processing-rgb',
                                  function_name='rgb2gray',
                                  execution_mode=dl.TriggerExecutionMode.ALWAYS,
                                  resource=dl.TriggerResource.ITEM,
                                  actions=dl.TriggerAction.UPDATED,
                                  filters=filters)
```

To trigger the function only once (only on the first item update), set `TriggerExecutionMode.ONCE` instead of `TriggerExecutionMode.ALWAYS`.

## Execute the function

Now we can upload (“create”) an image to our dataset to trigger the service. The function `clahe_equalization` will be invoked:

```
item = dataset.items.upload(
    local_path=['https://raw.githubusercontent.com/dataloop-ai/tiny_coco/master/images/
    ↪train2017/0000000463730.jpg'])
```

To see the original item, please click [here](#).

## Review the function’s logs

You can review the execution log history to check that your execution succeeded:

```
service.log()
```

The transformed image will be saved in your dataset. Once you see in the log that the execution succeeded, you may open the item to see its transformation:

```
item.open_in_web()
```

### Pause the service:

We recommend pausing the service you created for this tutorial so it will not be triggered:

```
service.pause()
```

Congratulations! You have successfully created, deployed, and tested Dataloop functions!

### Multiple Modules

You can define multiple different modules in a package. A typical use-case for multiple-modules is to have a single code base that can be used by a number of services (for different applications). For example, having a single YoloV4 codebase, but creating different modules for training, inference, etc.

When creating a service from that package, you will need to define which module the service will serve (a service can only serve a single module with all its functions). For example, to push a 2 module package, you will need to have 2 entry points, one for each module, and this is how you define the modules:

```
modules = [
    dl.PackageModule(
        name='first-module',
        entry_point='first_module_main.py',
        functions=[
            dl.PackageFunction(
                name='run',
                inputs=[dl.FunctionIO(name='item',
                                     type=dl.PackageInputType.ITEM)]
            )
        ]
    ),
    dl.PackageModule(
        name='second-module',
        entry_point='second_module_main.py',
        functions=[
            dl.PackageFunction(
                name='run',
                inputs=[dl.FunctionIO(name='item',
                                     type=dl.PackageInputType.ITEM)]
            )
        ]
    )
]
```

Create the package with your modules

```
package = project.packages.push(package_name='two-modules-test',
                                modules=modules,
                                src_path='<path to where the entry point is located>'
                                )
```

You will pass these modules as a param to `packages.push()` After that, when you deploy the package, you will need to specify the module name: Note: A service can only implement one module.

```
service = package.deploy(
    module_name='first-module',
    service_name='first-module-test-service'
)
```

## 5.2.5 Execution Control

Kill and Timeout on an Execution

### Executions Control

#### Execution Termination

Sometimes when we run long term executions, such as model training, we need the option to terminate the execution. This is facilitated using terminate at Checkpoint. To stop an execution set the code checkpoints to check if this execution received a termination and if it did, raise the Termination Exception. This allows you to save some work that was already done before terminating. For example:

```
class ServiceRunner(dl.BaseServiceRunner):
    def detect(self, item: dl.Item):
        # Do some work
        foo = 0
        self.kill_event()
        # Do some more work
        bar = 1
        self.kill_event()
        # Sleep for a while
        import time
        time.sleep(1)
        # And... done!
        return
```

Each time there is a “kill\_event” the service runner checks to see if this execution received a termination request. To kill such execution we use

```
execution.terminate()
```

#### Execution Timeout

You can tell an execution to stop after a given number of seconds with the timeout parameter (the default time is 1 hour). In case a service reset, such as in timeout or service update, If there are running executions the service will wait for the execution timeout before resetting. The number have to be a natural number (int).

```
service.execution_timeout = 60 # 1 minute
```

You can decide what to do to executions that have experienced a timeout. There are 2 options of timeout handling:

1. Mark execution as failed
2. Retry

```
service.on_reset = 'failed'
service.on_reset = 'rerun'
# The service must be updated after changing these attributes
service.update()
```

## 5.3 Task Workflows

Tutorials for workforce management

### 5.3.1 Tasks and Assignment

Getting started with Task and Assignments.

#### Create Annotation Task

Getting started with Annotation Tasks.

#### Create a Task

To reach the tasks and assignments repositories go to and .

To reach the tasks and assignments entities go to and .

#### Creating a Task with Assignments

There are a couple of ways to create a task with assignments.

##### 1. By Folder Directory

This example will create a task for items that match a filter. The items will be divided equally between annotator's assignments:

```
import dtlpy as dl
import datetime
if dl.token_expired():
    dl.login()
project = dl.projects.get(project_name='<project_name>')
dataset = project.datasets.get(dataset_name='<dataset_name>')
filters = dl.Filters(field='<dir>', values='</my/folder/directory>') # filter by ↵
↵directory
task = dataset.tasks.create(
    task_name='<task_name>',
    due_date=datetime.datetime(day=1, month=1, year=2029).timestamp(),
    assignee_ids=['<annotator1@dataloop.ai>', '<annotator2@dataloop.ai>'],
    # The items will be divided equally between assignments
    filters=filters # filter by folder directory or use other filters
)
```

## 2. By Filters

This example will create a task for items that match a filter. The items will be divided equally between the annotator's assignments:

```
import dtlpy as dl
import datetime
if dl.token_expired():
    dl.login()
project = dl.projects.get(project_name='<project_name>')
dataset = project.datasets.get(dataset_name='<dataset_name>')
# filter items without annotations
filters = dl.Filters(field='<annotated>', values=False)
task = dataset.tasks.create(
    task_name='<task_name>',
    due_date=datetime.datetime(day=1, month=1, year=2029).timestamp(),
    assignee_ids=['<annotator1@dataloop.ai>', '<annotator2@dataloop.ai>'],
    # The items will be divided equally between assignments
    filters=filters # filter items without annotations or use other filters
)
```

## 3. List of Items

Create a task from a list of items. The items will be divided equally between annotator's assignments:

```
import dtlpy as dl
import datetime
if dl.token_expired():
    dl.login()
project = dl.projects.get(project_name='<project_name>')
dataset = project.datasets.get(dataset_name='<dataset_name>')
items = dataset.items.list()
items_list = [item for item in items.all()]
task = dataset.tasks.create(
    task_name='<task_name>',
    due_date=datetime.datetime(day=1, month=1, year=2029).timestamp(),
    assignee_ids=['<annotator1@dataloop.ai>', '<annotator2@dataloop.ai>'],
    # The items will be divided equally between assignments
    items=items_list
)
```

## 4. Full Dataset

Create a task from all of the items in the dataset. The items will be divided equally between annotator's assignments:

```
import dtlpy as dl
import datetime
if dl.token_expired():
    dl.login()
project = dl.projects.get(project_name='<project_name>')
```

(continues on next page)

(continued from previous page)

```
dataset = project.datasets.get(dataset_name='<dataset_name>')
task = dataset.tasks.create(
    task_name='<task_name>',
    due_date=datetime.datetime(day=1, month=1, year=2029).timestamp(),
    assignee_ids=['<annotator1@dataloop.ai>', '<annotator2@dataloop.ai>']
    # The items will be divided equally between assignments
)
```

## Add items to an existing task

Adding items to an existing task will create new assignments (for new assignee/s).

### 1. By Filters

```
import dtlpy as dl
import datetime
if dl.token_expired():
    dl.login()
project = dl.projects.get(project_name='<project_name>')
dataset = project.datasets.get(dataset_name='<dataset_name>')
filters = dl.Filters(field='<metadata.system.refs>', values=[]) # filter on unassigned_
↳ items
task.add_items(
    filters=filters, # filter by folder directory or use other filters
    assignee_ids=['<annotator1@dataloop.ai>', '<annotator2@dataloop.ai>'])
```

### 2. Single Item

```
import dtlpy as dl
import datetime
if dl.token_expired():
    dl.login()
project = dl.projects.get(project_name='<project_name>')
dataset = project.datasets.get(dataset_name='<dataset_name>')
item = dataset.items.get(item_id='<my-item-id>')
task.add_items(
    items=[item],
    assignee_ids=['<annotator1@dataloop.ai>', '<annotator2@dataloop.ai>'])
```



### 3. List of Items

```
import dtlpy as dl
import datetime
if dl.token_expired():
    dl.login()
project = dl.projects.get(project_name='<project_name>')
dataset = project.datasets.get(dataset_name='<dataset_name>')
items = dataset.items.list()
items_list = [item for item in items.all()]
task.add_items(
    items=items_list,
    assignee_ids=['<annotator1@dataloop.ai>', '<annotator2@dataloop.ai>']
)
```

### Create Annotation Assignment

Getting started with Annotation Assignment.

### Task Assignment

To reach the tasks and assignments repositories go to and .

To reach the tasks and assignments entities go to and .

### Item Review

The Annotation Studio is built for realtime review, task assignment and feedback.

Each item can be classified in 3 ways:

- **Discarded:** Items that are not relevant for labeling
- **Complete** (or an alternate custom status created by the task creator): Items after an annotation process
- **Approved** (or an alternate custom status created by the task creator): Completed items after a QA process ####  
Prep

```
import dtlpy as dl
if dl.token_expired():
    dl.login()
project = dl.projects.get(project_name='<project_name>')
dataset = project.datasets.get(dataset_name='<dataset_name>')
```

## Single status update

```
# Mark single item as completed
item = dataset.items.get(item_id='<my-item-id>')
item.update_status(status=dl.ItemStatus.COMPLETED)
# In the same way you can update to another status
item.update_status(status=dl.ItemStatus.APPROVED)
item.update_status(status=dl.ItemStatus.DISCARDED)
```

## Clear status

```
# Clear status for completed/approved/discarded
item.update_status(dl.ITEM_STATUS_COMPLETED, clear=True)
```

## Bulk status update

```
# With items list
filters = dl.Filters(field='<annotated>', values=True)
items = dataset.items.list(filters=filters)
dataset.items.update_status(status=dl.ItemStatus.APPROVED, items=items)
# With filters
filters = dl.Filters(field='<annotated>', values=True)
dataset.items.update_status(status=dl.ItemStatus.DISCARDED, filters=filters)
# With list of item ids
item_ids = ['<id1>', '<id2>', '<id3>']
dataset.items.update_status(status=dl.ItemStatus.COMPLETED, item_ids=item_ids)
```

## Example

To mark an entire task as completed use the following:

```
task = dataset.tasks.get(task_name='<my-task-name>')
dataset.items.update_status(status=dl.ItemStatus.COMPLETED, items=task.get_items())
```

## Redistribute and Reassign

Redistribute and reassign items from tasks and assignments

## Redistributing and Reassigning a Task

To reach the tasks and assignments repositories go to and .

To reach the tasks and assignments entities go to and .

## Get Task and Assignments

### Get Task

#### Get by ID

```
task = dl.tasks.get(task_id='<my-task-id>')
```

#### Get by name – in a

```
project = dl.projects.get(project_name='<project_name>')
task = project.tasks.get(task_name='<my-task-name>')
```

#### Get by name – in a

```
dataset = project.datasets.get(dataset_name='<dataset_name>')
task = project.tasks.get(task_name='<my-task-name>')
```

#### Get list – in a

```
dataset = project.datasets.get(dataset_name='<dataset_name>')
task = project.tasks.get(task_name='<my-task-name>')
```

#### Get list – in a

```
tasks = project.tasks.list()
```

### Get Task Items

```
tasks = dataset.tasks.list()
```

## Get Assignments

### Get by ID

```
assignment = dl.assignments.get(assignment_id='<my-assignment-id>')
```

### Get by name – in a

```
project = dl.projects.get(project_name='<project_name>')
assignment = project.assignments.get(assignment_name='<my-assignment-name>')
```

### Get by name – in a

```
dataset = project.datasets.get(dataset_name='<dataset_name>')
assignment = dataset.assignments.get(assignment_name='<my-assignment-name>')
```

### Get by name – in a

```
task = project.tasks.get(task_name='<my-task-name>')
assignment = task.assignments.get(assignment_name='<my-assignment-name>')
```

### Get list – in a

```
assignments = project.assignments.list()
```

### Get list – in a

```
assignments = dataset.assignments.list()
```

### Get list – in a

```
assignments = task.assignments.list()
```

## Get Assignment Items

```
assignment_items = assignment.get_items()
```

## Redistribute and Reassign the Assignment

### prep

```
import dtlpy as dl
import datetime
if dl.token_expired():
    dl.login()
project = dl.projects.get(project_name='<project_name>')
dataset = project.datasets.get(dataset_name='<dataset_name>')
task = dl.tasks.get(task_id='<my-task-id>')
assignment = task.assignments.get(assignment_name='<my-assignment-name>')
```

### Redistribute

```
# load is the workload percentage for each annotator
assignment.redistribute(dl.Workload([dl.WorkloadUnit(assignee_id='<annotator1@dataloop.
↪ai>', load=50),
                                dl.WorkloadUnit(assignee_id='<annotator2@dataloop.
↪ai>', load=50)]))
```

### Reassign

```
assignment.reassign(assignee_ids['<annotator1@dataloop.ai>'])
```

## Delete Task and Assignments

### Delete Task

```
task.delete()
```

## 5.3.2 QA Tasks Management

Create QA tasks and annotation-qa flows

## Create QA Task

Getting started with QA Tasks.

### Create a QA Task

To reach the tasks and assignments repositories go to and .

To reach the tasks and assignments entities go to and .

In Dataloop there are two ways to create a QA task:

1. You can create a QA task from the annotation task. This will collect all completed Items and create a QA Task.
2. You can create a standalone QA task. *### QA task from the annotation task #### prep*

```
import dtlpy as dl
import datetime
if dl.token_expired():
    dl.login()
project = dl.projects.get(project_name='<project_name>')
dataset = project.datasets.get(dataset_name='<dataset_name>')
# Get the annotation task, you can also get a task by name or from a list
task = project.tasks.get(task_id='<my-task-id>')
```

## 2. Create a QA Task

This action will collect all completed Items and create a QA Task under the annotation task.

```
# Add filter for completed items
filters = dl.Filters()
filters.add(field='<metadata.system.annotationStatus>', values='<completed>')
# create a QA task - fill in the due date and assignees.
QATask = dataset.tasks.create_qa_task(task=task,
                                       due_date=datetime.datetime(day=1, month=1,
↪year=2029).timestamp(),
                                       assignee_ids=['<annotator1@dataloop.ai>', '
↪<annotator2@dataloop.ai>'],
                                       filters=filters # this filter is for "completed_
↪items"
                                       )
```

## A standalone QA task

### prep

```
import dtlpy as dl
import datetime
if dl.token_expired():
    dl.login()
```

(continues on next page)

(continued from previous page)

```
project = dl.projects.get(project_name='<project_name>')
dataset = project.datasets.get(dataset_name='<dataset_name>')
```

## 2. Add filter by directory

```
filters = dl.Filters(field='<metadata.system.annotationStatus>', values='<completed>')
filters.add(field='<dir>', values='</my/folder/directory>')
```

## Create a QA Task

This action will collect all items on the folder and create a QA Task from them.

```
QAtask = dataset.tasks.create(
    task_type='<qa>',
    due_date=datetime.datetime(day=1, month=1, year=2029).timestamp(),
    assignee_ids=['<annotator1@dataloop.ai>', '<annotator2@dataloop.ai>'],
    filters=filters # filter by folder directory or use other filters
)
```

## Create QA Assignment

Getting started with QA Assignment.

## Note Annotation

Create Note annotation on items

## Note Annotation

To reach the tasks and assignments repositories go to and .

To reach the tasks and assignments entities go to and .

The Annotation Studio also enables real time dialog in the studio. The note annotation allows annotators and reviewers the option to add an issue directly to the item as an annotation.

## Prep

```
import dtlpy as dl
if dl.token_expired():
    dl.login()
```

## Init Note

With message inside and top, bottom, left, right positioning Using the annotations definitions classes you can create, edit, view and upload platform annotations.

```
annotation_definition = dl.Note(top=10, left=10, bottom=100, right=100, label='my-label')
annotation_definition.assignee = "user@dataloop.ai"
annotation_definition.add_message("this is a message 1")
annotation_definition.add_message("this is a message 2")
```

## Create Note Annotation

### 1. Get project and dataset

```
project = dl.projects.get(project_name='project_name')
dataset = project.datasets.get(dataset_name='dataset_name')
```

### 2. Get item from the platform

```
item = dataset.items.get(filepath='/your-image-file-path.jpg')
```

### 3. Create a builder instance

```
builder = item.annotations.builder()
```

### 4. Add a note

```
annotation_definition = dl.Note(top=10, left=10, bottom=100, right=100, label='my-label')
annotation_definition.assignee = "user@dataloop.ai"
annotation_definition.add_message("this is a message 1")
annotation_definition.add_message("this is a message 2")
builder.add(annotation_definition=annotation_definition)
```

### 5. Upload annotations to the item

```
item.annotations.upload(builder)
```



## QA on Annotation Level

Annotation level QA

## QA on Annotation Level

To reach the tasks and assignments repositories go to and .

To reach the tasks and assignments entities go to and .

## ItemAnnotations Review

The Annotation Studio also enables direct feedback for specific annotations. To enable a realtime review, a Reviewer can open an issue on an Annotation. The Annotator (person who annotated the issued Annotation) then receives the issue, fixes it and sends it back for a second review. The Reviewer may approve the fix or return it as an issue.

We also support a real-time dialog on items as an annotation, go to to learn more.

## Prep

```
import dtlpy as dl
if dl.token_expired():
    dl.login()
project = dl.projects.get(project_name='project_name')
dataset = project.datasets.get(dataset_name='dataset_name')
```

## Single status update

```
# Mark a single annotation with an open issue
item = dataset.items.get(item_id='my-item-id')
annotation = item.annotations.get(annotation_id='your-annotation-id-number')
annotation.update_status(dl.AnnotationStatus.ISSUE)
# In the same way you can update to another status
annotation.update_status(dl.AnnotationStatus.APPROVED)
annotation.update_status(dl.AnnotationStatus.REVIEW)
annotation.update_status(dl.AnnotationStatus.CLEAR) # Have the annotation without status
```

## Bulk status update

```
# Get Task
task = project.tasks.get(task_id='my_task_id')
# Add filters for items in the task who have annotations with issues
filters = dl.Filters()
filters.add_join(field='metadata.system.status', values='issue')
items = task.get_items(filters=filters)
# Go over all of the items
for page in items:
```

(continues on next page)

(continued from previous page)

```
for item in page:
    # Add filter for annotations with issues
    filters = dl.Filters()
    filters.resource = dl.FiltersResource.ANNOTATION
    filters.add(field='metadata.system.status', values='issue')
    annotations = item.annotations.list(filters=filters)
    # For every annotation that has issue in the item update the status to "for_
↪review"
    for annotation in annotations:
        ↪annotation.update_status(dl.AnnotationStatus.REVIEW)
```

## QA on Item Level

Item level QA

## QA on Item Level

To reach the tasks and assignments repositories go to and .

To reach the tasks and assignments entities go to and .

## Item Review

The Annotation Studio is built for realtime review, task assignment and feedback.

Each item can be classified in 3 ways:

- **Discarded:** Items that are not relevant for labeling
- **Complete** (or an alternate custom status created by the task creator): Items after an annotation process
- **Approved** (or an alternate custom status created by the task creator): Completed items after a QA process ####  
Prep

```
import dtlpy as dl
if dl.token_expired():
    dl.login()
project = dl.projects.get(project_name='<project_name>')
dataset = project.datasets.get(dataset_name='<dataset_name>')
```

## Single status update

```
# Mark single item as completed
item = dataset.items.get(item_id='<my-item-id>')
item.update_status(status=dl.ItemStatus.COMPLETED)
# In the same way you can update to another status
item.update_status(status=dl.ItemStatus.APPROVED)
item.update_status(status=dl.ItemStatus.DISCARDED)
```

## Clear status

```
# Clear status for completed/approved/discarded
item.update_status(dl.ITEM_STATUS_COMPLETED, clear=True)
```

## Bulk status update

```
# With items list
filters = dl.Filters(field='annotated', values=True)
items = dataset.items.list(filters=filters)
dataset.items.update_status(status=dl.ItemStatus.APPROVED, items=items)
# With filters
filters = dl.Filters(field='annotated', values=True)
dataset.items.update_status(status=dl.ItemStatus.DISCARDED, filters=filters)
# With list of item ids
item_ids = ['id1', 'id2', 'id3']
dataset.items.update_status(status=dl.ItemStatus.COMPLETED, item_ids=item_ids)
```

## Example

To mark an entire task as completed use this:

```
task = dataset.tasks.get(task_name='my-task-name')
dataset.items.update_status(status=dl.ItemStatus.COMPLETED, items=task.get_items())
```

## Redistribute and Reassign

Redistribute and reassign items from tasks and assignments

### Redistributing and Reassigning a QA Task

To reach the tasks and assignments repositories go to and .

To reach the tasks and assignments entities go to and .

### Get QA Task and Assignments

#### Get Task

#### Get by ID

```
QAtask = dl.tasks.get(task_id='<my-task-id>')
```

### Get by name – in a

```
project = dl.projects.get(project_name='<project_name>')
QAtask = project.tasks.get(task_name='<my-qa-task-name>')
```

### Get by name – in a

```
dataset = project.datasets.get(dataset_name='<dataset_name>')
QAtask = project.tasks.get(task_name='<my-qa-task-name>')
```

### Get list – in a

```
tasks = project.tasks.list()
```

### Get list – in a

```
tasks = dataset.tasks.list()
```

### Get Task Items

```
qa_task_items = QAtask.get_items()
```

### Get Assignments

#### Get by ID

```
assignment = dl.assignments.get(assignment_id='<my-assignment-id>')
```

#### Get by name – in a

```
project = dl.projects.get(project_name='<project_name>')
assignment = project.assignments.get(assignment_name='<my-assignment-name>')
```

**Get by name – in a**

```
dataset = project.datasets.get(dataset_name='<dataset_name>')
assignment = dataset.assignments.get(assignment_name='<my-assignment-name>')
```

**Get by name – in a**

```
task = project.tasks.get(task_name='<my-task-name>')
assignment = task.assignments.get(assignment_name='<my-assignment-name>')
```

**Get list – in a**

```
assignments = project.assignments.list()
```

**Get list – in a**

```
assignments = dataset.assignments.list()
```

**Get list – in a**

```
assignments = task.assignments.list()
```

**Get Assignment Items**

```
assignment_items = assignment.get_items()
```

**Redistribute and Reassign the QA Assignment**

```
import dtlpy as dl
import datetime
if dl.token_expired():
    dl.login()
project = dl.projects.get(project_name='<project_name>')
dataset = project.datasets.get(dataset_name='<dataset_name>')
QAtask = dl.tasks.get(task_id='<my-task-id>')
assignment = task.assignments.get(assignment_name='<my-assignment-name>')
```

## Redistribute

```
# load is the workload percentage for each annotator
assignment.redistribute(dl.Workload([dl.WorkloadUnit(assignee_id='<annotator1@dataloop.
↪ai>', load=50),
                                dl.WorkloadUnit(assignee_id='<annotator2@dataloop.
↪ai>', load=50)]))
```

## Reassign

```
assignment.reassign(assignee_ids['<annotator1@dataloop.ai>'])
```

## Delete Task and Assignments

### Delete Task

```
QAtask.delete()
```

### Delete Assignment

```
assignment.delete()
```

## 5.4 Image Annotations

Tutorials for creating all types of image annotations

### 5.4.1 Setup

Setup environment before starting

This tutorial guides you through the process using the Dataloop SDK to create and upload annotations into items. The tutorial includes chapters with different tools, and the last chapter includes various more advanced scripts

- [Classification Point & Pose](#)
- [Bounding Box & Cuboid](#)
- [Polygon & Polyline](#)
- [Ellipse & Item-Description](#)
- [Advanced Tutorials](#)
  - [Copy Annotations Between Items](#)
  - [Show Images & Annotations](#)
  - [Show Annotations from JSON file](#)
  - [Count the Total Number of Annotations in a Dataset](#)

- Parenting Annotations
- Change Annotation’s Label to a New Label

## Setup

```
import dtlpy as dl
if dl.token_expired():
    dl.login()
project = dl.projects.get(project_name='project_name')
dataset = project.datasets.get(dataset_name='dataset_name')
```

## Initiation

Using the annotation definitions classes you can create, edit, view and upload platform annotations. Each annotation init receives the coordinates for the specific type, label, and optional attributes.

## Optional Plotting

Before updating items with annotations, you can optionally plot the annotation you created and review it before uploading it. This applies to all annotations described in the following section.

```
import matplotlib.pyplot as plt
plt.figure()
plt.imshow(builder.show())
for annotation in builder:
    plt.figure()
    plt.imshow(annotation.show())
    plt.title(annotation.label)
```

## 5.4.2 Classification, Point and Pose

Classification, Point and Pose annotations types

### Classification

Classify a single item

```
# Get item from the platform
item = dataset.items.get(filepath='/your-image-file-path.jpg')
# Create a builder instance
builder = item.annotations.builder()
# Classify
builder.add(annotation_definition=dl.Classification(label=label))
# Upload classification to the item
item.annotations.upload(builder)
```

## Classify Multiple Items

Classifying multiple items requires using an Items entity with a filter.

```
# mutiple items classification using filter
...
```

## Create a Point Annotation

```
# Get item from the platform
item = dataset.items.get(filepath='/your-image-file-path.jpg')
# Create a builder instance
builder = item.annotations.builder()
# Create point annotation with label and attribute
builder.add(annotation_definition=dl.Point(x=100,
                                           y=100,
                                           label='my-label',
                                           attributes={'color': 'red'}))
# Upload point to the item
item.annotations.upload(builder)
```

## Pose Annotation

```
# Pose annotation is based on pose template. Create the pose template from the platform,
↳ UI and use it in the script by its ID
template_id = recipe.get_annotation_template_id(template_name="my_template_name")
# Get item
item = dataset.items.get(filepath='/your-image-file-path.jpg')
# Define the Pose parent annotation and upload it to the item
parent_annotation = item.annotations.upload(
    dl.Annotation.new(annotation_definition=dl.Pose(label='my_parent_label',
                                                  template_id=template_id,
                                                  # instance_id is optional
                                                  instance_id=None)))[0]
# Add child points
builder = item.annotations.builder()
builder.add(annotation_definition=dl.Point(x=x,
                                           y=y,
                                           label='my_point_label'),
           parent_id=parent_annotation.id)
builder.upload()
```



### 5.4.3 Bounding Box and Cuboid

Bounding Box and Cuboid annotations types

#### Create Box Annotation

```
# Get item from the platform
item = dataset.items.get(filepath='/your-image-file-path.jpg')
# Create a builder instance
builder = item.annotations.builder()
# Create box annotation with label
builder.add(annotation_definition=dl.Box(top=10,
                                         left=10,
                                         bottom=100,
                                         right=100,
                                         label='my-label'))

# Upload box to the item
item.annotations.upload(builder)
```

#### Create a Rotated Bounding Box Annotation

A rotated box is created by setting its top-left and bottom-right coordinates, and providing its rotation angle.

```
# Get item from the platform
item = dataset.items.get(filepath='/your-image-file-path.jpg')
# Create a builder instance
builder = item.annotations.builder()
# Create box annotation with label
builder.add(annotation_definition=dl.Box(top=10,
                                         left=10,
                                         bottom=100,
                                         right=100,
                                         angle=80,
                                         label='my-label'))

# Upload box to the item
item.annotations.upload(builder)
```

#### Convert Semantic Segmentation to Bounding Box

Convert all semantic segmentation annotations in an item into box annotation

```
annotations = item.annotations.list()
builder = item.annotations.builder()
# run over all annotation in item
for annotation in annotations:
    if annotation.type == dl.AnnotationType.SEGMENTATION:
        print("Found binary annotation - id:", annotation.id)
        builder.add(annotation_definition=annotation.annotation_definition.to_box())
item.annotations.upload(annotations=builder)
```

## Create Cuboid (3D Box) Annotation

Create cuboid annotation in one of two ways :

```
# A.Bring front and back rectangles and the angel of the cuboid
builder.add(annotation_definition=dl.Cube.from_boxes_and_angle(label="label",
                                                                front_top=100,
                                                                front_left=100,
                                                                front_right=300,
                                                                front_bottom=300,
                                                                back_top=200,
                                                                back_left=200,
                                                                back_right=400,
                                                                back_bottom=400,
                                                                angle=0
                                                                ))

# B.Bring all 8 points of the Cuboid
builder.add(annotation_definition=dl.Cube(label="label",
                                          # front top left point coordinates
                                          front_tl=[200, 200],
                                          # front top right point coordinates
                                          front_tr=[500, 250],
                                          # front bottom right point coordinates
                                          front_br=[500, 550],
                                          # front bottom left point coordinates
                                          front_bl=[200, 500],
                                          # back top left point coordinates
                                          back_tl=[300, 300],
                                          # back top right point coordinates
                                          back_tr=[600, 350],
                                          # back bottom right point coordinates
                                          back_br=[600, 650],
                                          # back bottom left point coordinates
                                          back_bl=[300, 600]
                                          ))

item.annotations.upload(builder)
```

## 5.4.4 Polygon and Polyline

Polygon and Polyline annotations types

### Create Single Polygon/Polyline Annotation

```
# Get item from the platform
item = dataset.items.get(filepath='/your-image-file-path.jpg')
# Create a builder instance
builder = item.annotations.builder()
# Create polygon annotation with label
# with array of points: [[x1, y1], [x2, y2], ..., [xn, yn]]
builder.add(annotation_definition=dl.Polygon(geo=[[100, 50],
                                                  [80, 120],
```

(continues on next page)

(continued from previous page)

```

                                [110, 130]],
                                label='my-label'))
# create Polyline annotation with label
builder.add(annotation_definition=dl.Polyline(geo=[[100, 50],
                                                [80, 120],
                                                [110, 130]],
                                                label='my-label'))
# Upload polygon to the item
item.annotations.upload(builder)

```

### Create Multiple Polygons from Mask

```

annotations = item.annotations.list()
mask_annotation = annotations[0]
builder = item.annotations.builder()
builder.add(dl.Polygon.from_segmentation(mask_annotation.geo,
                                       max_instances=2,
                                       label=mask_annotation.label))
item.annotations.upload(builder)

```

### Convert Mask Annotations to Polygon

More about `from_segmentation()` function on [dtlpy](#).

```

annotations = item.annotations.list()
builder = item.annotations.builder()
# run over all annotation in item
for annotation in annotations:
    if annotation.type == dl.AnnotationType.SEGMENTATION:
        print("Found binary annotation - id:", annotation.id)
        builder.add(dl.Polygon.from_segmentation(mask=annotation.annotation_definition.
        ↪ geo,
                                                # binary mask of the annotation
                                                label=annotation.label,
                                                max_instances=None))
        annotation.delete()
item.annotations.upload(annotations=builder)

```

### Convert Polygon Annotation to Mask

More about `from_polygon()` function on [dtlpy](#). This script uses module CV2, please use [page](#) to install it.

```

if annotation.type == dl.AnnotationType.POLYGON:
    print("Found polygon annotation - id:", annotation.id)
    builder.add(dl.Segmentation.from_polygon(geo=annotation.annotation_definition.geo,
                                            # binary mask of the annotation
                                            label=annotation.label,
                                            shape=img.size[::-1] # (h,w)

```

(continues on next page)

(continued from previous page)

```
                                ))
annotation.delete()
item.annotations.upload(annotations=builder)
```

## 5.4.5 Ellipse and Item Description

Ellipse and Item Description annotations types

### Create Ellipse Annotation

```
# Get item from the platform
item = dataset.items.get(filepath='/your-image-file-path.jpg')
# Create a builder instance
builder = item.annotations.builder()
# Create ellipse annotation with label - With params for an ellipse; x and y for the
↪center, rx, and ry for the radius and rotation angle:
builder.add(annotations_definition=dl.Ellipse(x=x,
                                              y=y,
                                              rx=rx,
                                              ry=ry,
                                              angle=angle,
                                              label=label))

# Upload the ellipse to the item
item.annotations.upload(builder)
```

### Item Description

Item description is added as a “system annotation”, and serves as a way to save information about the item, that can be seen by anyone accessing it.

```
# Get item from the platform
item = dataset.items.get(filepath='/your-image-file-path.jpg')
# Add description (update if already exists)- if text is empty it will remove the
↪description from the item
item.set_description(text="this is item description")
```

## 5.4.6 Advance Tutorials

Copy, count, show and annotation parenting.

## Copy Annotations Between Items

By setting annotations entity from one item, and uploading it into another, we can copy annotations between items. Running through all items in a filter allows us to copy from one item into multiple items, for example video snapshots with the same object.

```
# Set the source item with the annotations we want to copy
project = dl.projects.get(project_name='second-project_name')
dataset = project.datasets.get(dataset_name='second-dataset_name')
item = dataset.items.get(item_id='first-id-number')
annotations = item.annotations.list()
# Set the target item where we want to copy to. If located on a different Project or
↳Dataset, set these accordingly
item = dataset.items.get(item_id='second-id-number')
item.annotations.upload(annotations=annotations)
# Copy the annotation into multiple items, based on a filter entity. In this example,
↳the filter is based on directory
filters = dl.Filters()
filters.add(field='filename', values='/fighting/**') # take files from the directory
↳only (recursive)
filters.add(field='type', values='file') # only files
pages = dataset.items.list(filters=filters)
for page in pages:
    for item in page:
        # upload annotations
        item.annotations.upload(annotations=annotations)
```

## Show Images & Annotations

This script uses module CV2, please use this page to install it.

```
from PIL import Image
# Get item
item = dataset.items.get(item_id='write-your-id-number')
# download item as a buffer
buffer = item.download(save_locally=False)
# open image
image = Image.open(buffer)
# download annotations
annotations = item.annotations.show(width=image.size[0],
                                   height=image.size[1],
                                   thickness=3)
annotations = Image.fromarray(annotations.astype(np.uint8))
# show the annotations and the image separately
annotations.show()
image.show()
# Show the annotations with the image
image.paste(annotations, (0, 0), annotations)
image.show()
```

## Show Annotations from JSON file (Dataloop format)

Please notice that directory paths look different in OS and Linux and does not require “r” at the beginning

```
from PIL import Image
import json
with open(r'C:/home/project/images/annotation.json', 'r') as f:
    data = json.load(f)
for annotation in data['annotations']:
    annotations = dl.Annotation.from_json(annotation)
    mask = annotations.show(width=640,
                           height=480,
                           thickness=3,
                           color=(255, 0, 0))
    mask = Image.fromarray(mask.astype(np.uint8))
    mask.show()
```

## Count total number of annotations

The following script counts the number of annotations in a filter. The filter can be set to any context - Dataset, folder or any specific criteria. In the following example, it is set to a dataset.

```
# Create annotations filters instance
filters = dl.Filters(resource=dl.FiltersResource.ANNOTATION)
filters.page_size = 0
# Count the annotations
annotations_count = dataset.annotations.list(filters=filters).items_count
```

## Parenting Annotations

Parenting establishes a relation between 2 annotations, executed by setting the parent\_id parameter. The Dataloop system will reject an attempt to set circular parenting. The following script demonstrate setting parenting relation while uploading/creating annotations

```
builder = item.annotations.builder()
builder.add(annotation_definition=dl.Box(top=10, left=10, bottom=100, right=100,
                                         label='my-parent-label'))

# upload parent annotation
annotations = item.annotations.upload(annotations=builder)

# create the child annotation
builder = item.annotations.builder()
builder.add(annotation_definition=dl.Box(top=10, left=10, bottom=100, right=100,
                                         label='my-child-label'),
            parent_id=annotations[0].id)

# upload annotations to item
item.annotations.upload(annotations=builder)
```

The following script demonstrate setting parenting relation on existing annotations:

```
# create and upload parent annotation
builder = item.annotations.builder()
builder.add(annotation_definition=dl.Box(top=10, left=10, bottom=100, right=100,
```

(continues on next page)

(continued from previous page)

```

                                label='my-parent-label'))
parent_annotation = item.annotations.upload(annotations=builder)[0]
# create and upload child annotation
builder = item.annotations.builder()
builder.add(annotation_definition=dl.Box(top=10, left=10, bottom=100, right=100,
                                label='my-child-label'))
child_annotation = item.annotations.upload(annotations=builder)[0]
# set the child parent ID to the parent
child_annotation.parent_id = parent_annotation.id
# update the annotation
child_annotation.update(system_metadata=True)

```

## Change Annotations' Label

The following example creates a new label in the recipe (an optional step, you can also use an existing label), then applies it to all annotations in a certain filter.

```

# Create a new label
dataset.add_label(label_name='newLabel', color=(2, 43, 123))
# Filter annotations with the "oldLabel" label.
filters = dl.Filters()
filters.resource = dl.FiltersResource.ANNOTATION
filters.add(field='label', values='oldLabel')
pages = dataset.annotations.list(filters=filters)
# Change the Label of the Annotations - For every annotation we filtered out, Change it's
↳ Label to the "newLabel".
for annotation in pages.all():
    annotation.label = 'newLabel'
    annotation.update()

```

## 5.5 Video Annotations

Tutorials for annotating videos

### 5.5.1 Video Annotations

Upload and work with video annotations

In this tutorial we create and upload annotations into a video item. Video annotations differ from image annotations since they span over frames, and need to be set with their scope. This script uses module CV2, please use to install it.

## Setup

```
import dtlpy as dl
if dl.token_expired():
    dl.login()
project = dl.projects.get(project_name='project_name')
dataset = project.datasets.get(dataset_name='dataset_name')
item = dataset.items.get(filepath='/my_item.mp4')
```

## Create A Single annotation

Create a single annotations for a video item and upload it

```
annotation = dl.Annotation.new(item=item)
# Span the annotation over 100 frames. Change this or use a different approach based on
↳ your context
for i_frame in range(100):
    # go over 100 frame
    annotation.add_frame(annotation_definition=dl.Box(top=2 * i_frame,
                                                    left=2 * (i_frame + 10),
                                                    bottom=2 * (i_frame + 50),
                                                    right=2 * (i_frame + 100),
                                                    label="my-label"),
                        frame_num=i_frame, # set the frame for the annotation
                        )
# upload to platform
annotation.upload()
```

## Adding Multiple Annotations Using Annotation Builder

The following scripts demonstrate adding 10 annotations into each frame

```
# create annotation builder
builder = item.annotations.builder()
for i_frame in range(100):
    # go over 100 frames
    for i_detection in range(10):
        # for each frame we have 10 different detections (location is just for the
        ↳ example)
        builder.add(annotation_definition=dl.Box(top=2 * i_frame,
                                                left=2 * i_detection,
                                                bottom=2 * i_frame + 10,
                                                right=2 * i_detection + 100,
                                                label="my-label"),
                    # set the frame for the annotation
                    frame_num=i_frame,
                    # need to input the element id to create the connection between
                    ↳ frames
                    object_id=i_detection + 1,
                    )
```

(continues on next page)



(continued from previous page)

```
# Upload the annotations to platform
item.annotations.upload(builder)
```

## Read Frames of an Annotation

The following example reads all the frames an annotation exist in, e.g. the frame range an annotation spans over.

```
for annotation in item.annotations.list():
    print(annotation.object_id)
    for key in annotation.frames:
        frame = annotation.frames[key]
        print(frame.left, frame.right, frame.top, frame.bottom)
```

## Create Frame Snapshots from Video

One of Dataloop video utilities enables creating a frame snapshot from a video item every X frames (frame\_interval). You will need FFmpeg needs to be installed on your system using .

```
dl.utilities.Videos.video_snapshots_generator(item=item, frame_interval=30)
```

## Play An Item In Video Player

Play a video item with its annotations and labels with a video player

```
from dtlpy.utilities.videos.video_player import VideoPlayer
VideoPlayer(project_name=project_name,
            dataset_name=dataset_name,
            item_filepath=item_filepath)
```

## Show Annotations in a Specified Frame

```
import matplotlib.pyplot as plt
# Get from platform
annotations = item.annotations.list()
# Plot the annotations in frame 55 of the created annotations
frame_annotation = annotations.get_frame(frame_num=55)
plt.figure()
plt.imshow(frame_annotation.show())
plt.title(frame_annotation.label)
# Play video with the Dataloop video player
annotations.video_player()
```

## 5.6 Recipe and Ontology

Tutorials for managing ontologies, labels, and recipes

### 5.6.1 Concepts

What are Recipe and Ontology

#### Recipe and Ontology Concepts

The Dataloop Recipe & Ontology concepts are detailed in our documentation. In short:

- Ontology - an entity that contains labels and attributes. An attribute is linked to a label
- Recipe - An entity that ties an ontology with labeling instructions
  - Linked with an ontology
  - Labeling tools (e.g. box, polygon etc)
  - Optional PDF instructions
  - And more...

### 5.6.2 Ontology

Create and manage Ontology, Labels and Attributes

In this chapter we will create an ontology and populate it with labels

#### Preparing - Entities setup

```
import dtlpy as dl
if dl.token_expired():
    dl.login()
project = dl.projects.get(project_name='project_name')
dataset = project.datasets.get(dataset_name='dataset_name')
# Get recipe from list
recipe = dataset.recipes.list()[0]
# Or get specific recipe:
recipe = dataset.recipes.get(recipe_id='id')
# Get ontology from list or create it using the "Create Ontology" script
ontology = recipe.ontologies.list()[0]
# Or get specific ontology:
ontology = recipe.ontologies.get(ontology_id='id')
# Print entities:
recipe.print()
ontology.print()
```

## Create an Ontology

```
project = dl.projects.get(project_name='project_name')
ontology = project.ontologies.create(title="your_created_ontology_title",
                                     labels=[dl.Label(tag="Chameleon", color=(255, 0, 255))])
```

## Labels

Ontology uses the ‘Labels’ entity, which is a python list object, and as such you can use python list methods such as `sort()`. Be sure to use `ontology.update()` after each python list action.

```
ontology.add_labels(label_list=['Shark', 'Whale', 'Animal.Donkey'], update_ontology=True)
```

Labels can be added with branched hierarchy to facilitate sub-labels at up-to 5 levels. Labels hierarchy is created by adding ‘.’ between parent and child labels. In the above example, this script will get the Donkey Label:

```
child_label = ontology.labels[-1].children[0]
print(child_label.tag, child_label.rgb)
```

## Attributes

An attribute describes a label, without having to add more labels. For example “Car” is a label, but its color is an attribute. You can add multiple attributes to the ontology, and map it to labels. For example create the “color” attribute once, but have multiple labels use it. Attributes can be multiple-selection (e.g checkbox), single selection (radio button), value over slider, a yes/no question and free-text. An attribute can be set as a mandatory one, so annotators have to answer it before they can complete the item.

## Add attributes to the ontology

The following example adds 1 attribute of every type, all as a mandatory attribute:

- Multiple-choice attribute
- Single-choice attributes
- Slider attribute
- Yes/no question attribute
- Free text attribute

```
# This option is not available yet
...
```

## Read Ontology Attributes

Read & print the all the ontology attributes:

```
print(ontology.metadata['attributes'])
keys = [att['key'] for att in ontology.metadata['attributes']]
```

## Getting all labels is (including children):

```
print(ontology.labels_flat_dict)
```

## 5.6.3 Recipe

Create and manage Recipe and Annotations Instructions

Since a recipe is linked with an ontology, it allows for making changes with labels and attributes. When the recipe is set as the default one for a dataset, the same applies for the dataset entity - it can be used for making changes with the labels and attributes which are ultimately linked to it through the recipe and its ontology.

## Working With Recipes

```
# Get recipe from a list
recipe = dataset.recipes.list()[0]
# Get recipe by ID - ID can be retrieved from the page URL when opening the recipe in
↳ the platform
recipe = dataset.recipes.get(recipe_id='your-recipe-id')
# Delete recipe - applies only for deleted datasets
dataset.recipes.get(recipe_id='your-recipe-id').delete()
```

## Cloning Recipes

When you want to create a new recipe that's only slightly different from an existing recipe, it can be easier to start by cloning the original recipe and then making changes on its clone. shallow: If True, link to existing ontology,

If false clone all ontologies that are links to the recipe as well.

```
dataset = project.datasets.get(dataset_name="myDataSet")
recipe = dataset.recipes.get(recipe_id="recipe_id")
recipe2 = recipe.clone(shallow=False)
```

## View Dataset Labels

```
# as objects
labels = dataset.labels
# as instance map
labels = dataset.instance_map
```

## Add Labels by Dataset

Working with dataset labels can be done one-by-one or as a list. The Dataset entity documentation details all label options - read .

```
# Add multiple labels
dataset.add_labels(label_list=['person', 'animal', 'object'])
# Add single label with specific color and attributes
dataset.add_label(label_name='person', color=(34, 6, 231))
# Add single label with a thumbnail/icon
dataset.add_label(label_name='person', icon_path='/home/project/images/icon.jpg')
```

## Add Labels Using Label Object

```
# Create Labels list using Label object
labels = [
    dl.Label(tag='Donkey', color=(255, 100, 0)),
    dl.Label(tag='Mammoth', color=(34, 56, 7)),
    dl.Label(tag='Bird', color=(100, 14, 150))
]
# Add Labels to Dataset
dataset.add_labels(label_list=labels)
# or you can also create a recipe from the label list
recipe = dataset.recipes.create(recipe_name='My-Recipe-name', labels=labels)
```

## Add a Label and Sub-Labels

```
label = dl.Label(tag='Fish',
                 color=(34, 6, 231),
                 children=[dl.Label(tag='Shark',
                                    color=(34, 6, 231)),
                          dl.Label(tag='Salmon',
                                    color=(34, 6, 231))]
                )
dataset.add_labels(label_list=label)
# or you can also create a recipe from the label list
recipe = dataset.recipes.create(recipe_name='My-Recipe-name', labels=labels)
```

## Add Hierarchy Labels with Nested

Different options for hierarchy label creation.

```
# Option A
# add father label
labels = dataset.add_label(label_name="animal", color=(123, 134, 64))
# add child label
labels = dataset.add_label(label_name="animal.Dog", color=(45, 34, 164))
# add grandchild label
labels = dataset.add_label(label_name="animal.Dog.poodle")
# Option B: only if you dont have attributes
# parent and grandparent (animal and dog) will be generated automatically
labels = dataset.add_label(label_name="animal.Dog.poodle")
# Option C: with the Big Dict
nested_labels = [
    {'label_name': 'animal.Dog',
     'color': '#220605',
     'children': [{'label_name': 'poodle',
                     'color': '#298345'},
                  {'label_name': 'labrador',
                     'color': '#298651'}]},
    {'label_name': 'animal.cat',
     'color': '#287605',
     'children': [{'label_name': 'Persian',
                     'color': '#298345'},
                  {'label_name': 'Balinese',
                     'color': '#298651'}]}
]
# Add Labels to the dataset:
labels = dataset.add_labels(label_list=nested_labels)
```

## Delete Labels by Dataset

```
dataset.delete_labels(label_names=['Cat', 'Dog'])
```

## Update Label Features

```
# update existing label , if not exist fails
dataset.update_label(label_name='Cat', color="#000080")
# update label, if not exist add it
dataset.update_label(label_name='Cat', color="#fcb03", upsert=True)
```

## 5.7 Model Management

Tutorials for creating and managing model and snapshots

### 5.7.1 Introduction

Getting started with Model.

#### Model Management

##### Introduction

Dataloop's Model Management is here to provide Machine Learning engineers the ability to manage their research and production process.

We want to introduce Dataloop entities to create, manage, view, compare, restore, and deploy training sessions.

Our Model Management gives a separation between Model code, weights and configuration, and the data.

in Offline mode, there is no need to do any code integration with Dataloop - just create a model and snapshots entities and you can start managing your work on the platform create reproducible training:

- same configurations and dataset to reproduce the training
- view project/org models and snapshots in the platform
- view training metrics and results
- compare experiments NOTE: all functions from the codebase can be used in FaaS and pipelines only with custom functions! User must create a FaaS and expose those functions any way he'd like

Online Mode: In the online mode, you can train and deploy your models easily anywhere on the platform. All you need to do is create a Model Adapter class and expose some functions to build an API between Dataloop and your model. After that, you can easily add model blocks to pipelines, add UI slots in the studio, one-button-training etc

#### Model and Snapshot entities

##### Model

The model entity is basically the algorithm, the architecture of the model, e.g Yolov5, Inception, SVM, etc.

- In online it should contain the Model Adapter to create a Dataloop API

## Snapshot

Using the Model (architecture), Dataset and Ontology (data and labels) and configuration (a dictionary) we can create a Snapshot of a training process. The Snapshot contains the weights and any other artifact needed to load the trained model

a snapshot can be used as a parent to another snapshot - to start for that point (fine-tune and transfer learning)

## Buckets and Codebase

1. local
2. item
3. git
4. GCS

## The Model Adapter

The Model Adapter is a python class to create a single API between Dataloop's platform and your Model

1. Train
2. Predict
3. load/save model weights
4. annotation conversion if needed

We enable two modes of work: in Offline mode, everything is local, you don't have to upload any model code or any weights to platform, which causes the platform integration to be minimal. For example, you cannot use the Model Management components in a pipeline, can easily create a button interface with your model's inference and more. In Online mode - once you build an Adapter, our platform can interact with your model and trained snapshots and you can connect buttons and slots inside the platform to create, train, inference etc and connect the model and any train snapshot to the UI or to add to a pipeline

## 5.7.2 Create a Model and Snapshot

Create a Model with a Dataloop Model Adapter

### Create Your own Model and Snapshot

We will create a dummy model adapter in order to build our model and snapshot entities NOTE: This is an example for a torch model adapter. This example will NOT run as-is. For working examples please refer to our models on github

The following class inherits from the `dl.BaseModelAdapter`, which have all the Dataloop methods for interacting with the Model and Snapshot There are four methods that are model-related that the creator must implement for the adapter to have the API with Dataloop

```
import dtlpy as dl
import torch
import os
class SimpleModelAdapter(dl.BaseModelAdapter):
```

(continues on next page)



(continued from previous page)

```

def load(self, local_path, **kwargs):
    print('loading a model')
    self.model = torch.load(os.path.join(local_path, 'model.pth'))
def save(self, local_path, **kwargs):
    print('saving a model to {}'.format(local_path))
    torch.save(self.model, os.path.join(local_path, 'model.pth'))
def train(self, data_path, output_path, **kwargs):
    print('running a training session')
def predict(self, batch, **kwargs):
    print('predicting batch of size: {}'.format(len(batch)))
    preds = self.model(batch)
    return preds

```

Now we can create our Model entity with an Item codebase.

```

project = dl.projects.get('MyProject')
codebase: dl.ItemCodebase = project.codebases.pack(directory='/path/to/codebase')
model = project.models.create(model_name='first-git-model',
                               description='Example from model creation tutorial',
                               output_type=dl.AnnotationType.CLASSIFICATION,
                               tags=['torch', 'inception', 'classification'],
                               codebase=codebase,
                               entry_point='dataloop_adapter.py',
                               )

```

For creating a Model with a Git code, simply change the codebase to be a Git one:

```

project = dl.projects.get('MyProject')
codebase: dl.GitCodebase = dl.GitCodebase(git_url='github.com/mygit', git_tag='v25.6.93')
model = project.models.create(model_name='first-model',
                               description='Example from model creation tutorial',
                               output_type=dl.AnnotationType.CLASSIFICATION,
                               tags=['torch', 'inception', 'classification'],
                               codebase=codebase,
                               entry_point='dataloop_adapter.py',
                               )

```

Creating a local snapshot:

```

bucket = dl.buckets.create(dl.BucketType.ITEM)
bucket.upload('/path/to/weights')
snapshot = model.snapshots.create(snapshot_name='tutorial-snapshot',
                                   description='first snapshot we uploaded',
                                   tags=['pretrained', 'tutorial'],
                                   dataset_id=None,
                                   configuration={'weights_filename': 'model.pth'},
                                   project_id=model.project.id,
                                   bucket=bucket,
                                   labels=['car', 'fish', 'pizza']
                                   )

```

Building to model adapter and calling one of the adapter's methods:

```
adapter = model.build()
adapter.load_from_snapshot(snapshot=snapshot)
adapter.train()
```

### 5.7.3 Using Dataloop's Dataset Generator

Use the SDK and the Dataset Tools to iterate, augment and serve the data to your model

#### Dataloop Dataloader

A `dl.Dataset` image and annotation generator for training and for items visualization

We can visualize the data with augmentation for debugging and exploration. After that, we will use the Data Generator as an input to the training functions.

```
from dtlpy.utilities import DatasetGenerator
import dtlpy as dl
dataset = dl.datasets.get(dataset_id='611b86e647fe2f865323007a')
datagen = DatasetGenerator(data_path='train',
                           dataset_entity=dataset,
                           annotation_type=dl.AnnotationType.BOX)
```

#### Object Detection Examples

We can visualize a random item from the dataset:

```
for i in range(5):
    datagen.visualize()
```

Or get the same item using its index:

```
for i in range(5):
    datagen.visualize(10)
```

Adding augmentations using `imgaug` repository:

```
from imgaug import augmenters as iaa
import numpy as np
augmentation = iaa.Sequential([
    iaa.Resize({"height": 256, "width": 256}),
    # iaa.Superpixels(p_replace=(0, 0.5), n_segments=(10, 50)),
    iaa.flip.Fliplr(p=0.5),
    iaa.flip.Flipud(p=0.5),
    iaa.GaussianBlur(sigma=(0.0, 0.8)),
])
tfs = [
    augmentation,
    np.copy,
    # transforms.ToTensor()
]
```

(continues on next page)

(continued from previous page)

```

datagen = DatasetGenerator(data_path='train',
                           dataset_entity=dataset,
                           annotation_type=dl.AnnotationType.BOX,
                           transforms=tfs)

datagen.visualize()
datagen.visualize(10)

```

All of the Data Generator options (from the function docstring):

**param dataset\_entity**

dl.Dataset entity

**param annotation\_type**

dl.AnnotationType - type of annotation to load from the annotated dataset

**param filters**

dl.Filters - filtering entity to filter the dataset items

**param data\_path**

Path to Dataloop annotations (root to “item” and “json”).

:param overwrite: :param label\_to\_id\_map: dict - {label\_string: id} dictionary :param transforms: Optional transform to be applied on a sample. list or torchvision.Transform :param num\_workers: :param shuffle: Whether to shuffle the data (default: True) If set to False, sorts the data in alphanumeric order. :param seed: Optional random seed for shuffling and transformations. :param to\_categorical: convert label id to categorical format :param class\_balancing: if True - performing random over-sample with class ids as the target to balance training data :param return\_originals: bool - If True, return ALSO images and annotations before transformations (for debug) :param ignore\_empty: bool - If True, generator will NOT collect items without annotations

The output of a single element is a dictionary holding all the relevant information. the keys for the DataGen above are: ['image\_filepath', 'item\_id', 'box', 'class', 'labels', 'annotation\_filepath', 'image', 'annotations', 'orig\_image', 'orig\_annotations']

```
print(list(datagen[0].keys()))
```

We'll add the flag to return the origin items to understand better how the augmentations look like. Let's set the flag and we can plot:

```

import matplotlib.pyplot as plt
datagen = DatasetGenerator(data_path='train',
                           dataset_entity=dataset,
                           annotation_type=dl.AnnotationType.BOX,
                           return_originals=True,
                           shuffle=False,
                           transforms=tfs)

fig, ax = plt.subplots(2, 2)
for i in range(2):
    item_element = datagen[np.random.randint(len(datagen))]
    ax[i, 0].imshow(item_element['image'])
    ax[i, 0].set_title('After Augmentations')
    ax[i, 1].imshow(item_element['orig_image'])
    ax[i, 1].set_title('Before Augmentations')

```

## Segmentation Examples

First we'll load a semantic dataset and view some images and the output structure

```
dataset = dl.datasets.get(dataset_id='6197985a104eb81cb728e4ac')
datagen = DatasetGenerator(data_path='semantic',
                           dataset_entity=dataset,
                           transforms=tfs,
                           return_originals=True,
                           annotation_type=dl.AnnotationType.SEGMENTATION)

for i in range(5):
    datagen.visualize()
```

Visualize original vs augmented image and annotations mask:

```
fig, ax = plt.subplots(2, 4)
for i in range(2):
    item_element = datagen[np.random.randint(len(datagen))]
    ax[i, 0].imshow(item_element['orig_image'])
    ax[i, 0].set_title('Original Image')
    ax[i, 1].imshow(item_element['orig_annotations'])
    ax[i, 1].set_title('Original Annotations')
    ax[i, 2].imshow(item_element['image'])
    ax[i, 2].set_title('Augmented Image')
    ax[i, 3].imshow(item_element['annotations'])
    ax[i, 3].set_title('Augmented Annotations')
```

Converting to 3d one-hot encoding to visualize the binary mask per label. We will plot only 8 labels (there might be more on the item):

```
item_element = datagen[np.random.randint(len(datagen))]
annotations = item_element['annotations']
unique_labels = np.unique(annotations)
one_hot_annotations = np.arange(len(datagen.id_to_label_map)) == annotations[..., None]
print('unique label indices in the item: {}'.format(unique_labels))
print('unique labels in the item: {}'.format([datagen.id_to_label_map[i] for i in unique_labels]))
plt.figure()
plt.imshow(item_element['image'])
fig = plt.figure()
for i_label_ind, label_ind in enumerate(unique_labels[:8]):
    ax = fig.add_subplot(2, 4, i_label_ind + 1)
    ax.imshow(one_hot_annotations[:, :, label_ind])
    ax.set_title(datagen.id_to_label_map[label_ind])
```

## Setting a Label Map

One of the inputs to the DatasetGenerator is ‘label\_to\_id\_map’. This variable can be used to change the label mapping for the annotations and allow using the dataset ontology in a greater variety of cases. For example, you can map multiple labels so a single id or add a default value for all the unlabeled pixels in segmentation annotations. This is what the annotation looks like without any mapping:

```
# project = dl.projects.get(project_name='Semantic')
# dataset = project.datasets.get(dataset_name='Hamster')
# dataset.items.upload(local_path='assets/images/hamster.jpg',
#                      local_annotations_path='assets/images/hamster.json')
dataset = dl.datasets.get(dataset_id='621ddc855c2a3d151451ec58')
datagen = DatasetGenerator(data_path='semantic',
                           dataset_entity=dataset,
                           return_originals=True,
                           overwrite=True,
                           annotation_type=dl.AnnotationType.SEGMENTATION)

datagen.visualize()
data_item = datagen[0]
plt.imshow(data_item['annotations'])
print('BG value: {}'.format(data_item['annotations'][0, 0]))
```

Now, we’ll map both the ‘eye’ label and the background to 2 and the ‘fur’ to 1:

```
dataset = dl.datasets.get(dataset_id='6197985a104eb81cb728e4ac')
label_to_id_map = {'cat': 1,
                   'dog': 1,
                   '$default': 0}
dataloader = DatasetGenerator(data_path='semantic',
                              dataset_entity=dataset,
                              transforms=tfs,
                              return_originals=True,
                              label_to_id_map=label_to_id_map,
                              annotation_type=dl.AnnotationType.SEGMENTATION)

for i in range(5):
    dataloader.visualize()
```

## Batch size and batch\_size and collate\_fn

If batch\_size is not None, the returned structure will be a list with batch\_size data items. Setting a collate function will convert the returned structure to a tensor of any kind. The default collate will convert everything to ndarrays. We also have tensorflow and torch collate to convert to the corresponding tensors.

```
dataset = dl.datasets.get(dataset_id='611b86e647fe2f865323007a')
datagen = DatasetGenerator(data_path='train',
                           dataset_entity=dataset,
                           batch_size=10,
                           annotation_type=dl.AnnotationType.BOX)

batch = datagen[0]
print('type: {}, len: {}'.format(type(batch), len(batch)))
print('single element in the list: {}'.format(batch[0]['image']))
# with collate
```

(continues on next page)

(continued from previous page)

```
from dtlpy.utilities.dataset_generators import collate_default
datagen = DatasetGenerator(data_path='train',
                           dataset_entity=dataset,
                           collate_fn=collate_default,
                           batch_size=10,
                           annotation_type=dl.AnnotationType.BOX)
batch = datagen[0]
print('type: {}, len: {}, shape: {}'.format(type(batch['images']), len(batch['images']),
↪batch['images'].shape))
```

## INDICES AND TABLES

- `genindex`
- `modindex`
- `search`





## PYTHON MODULE INDEX

### d

`dtlpy.entities.annotation`, 121  
`dtlpy.entities.annotation_collection`, 127  
`dtlpy.entities.annotation_definitions.base_annotation_definition`, 131  
`dtlpy.entities.annotation_definitions.box`, 131  
`dtlpy.entities.annotation_definitions.classification`, 131  
`dtlpy.entities.annotation_definitions.cube`, 131  
`dtlpy.entities.annotation_definitions.description`, 132  
`dtlpy.entities.annotation_definitions.ellipse`, 132  
`dtlpy.entities.annotation_definitions.note`, 132  
`dtlpy.entities.annotation_definitions.point`, 132  
`dtlpy.entities.annotation_definitions.polygon`, 133  
`dtlpy.entities.annotation_definitions.polyline`, 133  
`dtlpy.entities.annotation_definitions.pose`, 133  
`dtlpy.entities.annotation_definitions.segmentation`, 134  
`dtlpy.entities.annotation_definitions.subtitle`, 134  
`dtlpy.entities.annotation_definitions.undefined_annotation`, 134  
`dtlpy.entities.assignment`, 147  
`dtlpy.entities.base_entity`, 167  
`dtlpy.entities.bot`, 159  
`dtlpy.entities.codebase`, 154  
`dtlpy.entities.command`, 167  
`dtlpy.entities.dataset`, 107  
`dtlpy.entities.directory_tree`, 168  
`dtlpy.entities.driver`, 116  
`dtlpy.entities.execution`, 162  
`dtlpy.entities.filters`, 136  
`dtlpy.entities.integration`, 104  
`dtlpy.entities.item`, 117  
`dtlpy.entities.label`, 144  
`dtlpy.entities.links`, 121  
`dtlpy.entities.ontology`, 140  
`dtlpy.entities.organization`, 101  
`dtlpy.entities.package`, 150  
`dtlpy.entities.package_function`, 153  
`dtlpy.entities.package_module`, 153  
`dtlpy.entities.package_slot`, 154  
`dtlpy.entities.paged_entities`, 166  
`dtlpy.entities.pipeline`, 163  
`dtlpy.entities.pipeline_execution`, 165  
`dtlpy.entities.project`, 104  
`dtlpy.entities.recipe`, 139  
`dtlpy.entities.service`, 154  
`dtlpy.entities.similarity`, 135  
`dtlpy.entities.task`, 144  
`dtlpy.entities.trigger`, 160  
`dtlpy.entities.user`, 107  
`dtlpy.repositories.annotations`, 47  
`dtlpy.repositories.assignments`, 63  
`dtlpy.repositories.bots`, 86  
`dtlpy.repositories.codebases`, 75  
`dtlpy.repositories.commands`, 99  
`dtlpy.repositories.datasets`, 32  
`dtlpy.repositories.downloader`, 100  
`dtlpy.repositories.drivers`, 39  
`dtlpy.repositories.executions`, 90  
`dtlpy.repositories.integrations`, 26  
`dtlpy.repositories.items`, 40  
`dtlpy.repositories.ontologies`, 53  
`dtlpy.repositories.organizations`, 21  
`dtlpy.repositories.packages`, 67  
`dtlpy.repositories.pipeline_executions`, 98  
`dtlpy.repositories.pipelines`, 94  
`dtlpy.repositories.projects`, 28  
`dtlpy.repositories.recipes`, 51  
`dtlpy.repositories.services`, 78  
`dtlpy.repositories.tasks`, 56  
`dtlpy.repositories.triggers`, 87  
`dtlpy.repositories.uploader`, 100  
`dtlpy.utilities.converter`, 169



## A

abort() (Command method), 167  
 abort() (Commands method), 99  
 activate\_slots() (Service method), 156  
 activate\_slots() (Services method), 78  
 add() (AnnotationCollection method), 127  
 add() (Collection method), 135  
 add() (Filters method), 136  
 add() (Workload method), 149  
 add\_frame() (Annotation method), 121  
 add\_frames() (Annotation method), 121  
 add\_function() (PackageModule method), 153  
 add\_items() (Task method), 144  
 add\_items() (Tasks method), 56  
 add\_join() (Filters method), 136  
 add\_label() (Dataset method), 107  
 add\_label() (Ontology method), 140  
 add\_labels() (Dataset method), 108  
 add\_labels() (Ontology method), 141  
 add\_member() (Organization method), 101  
 add\_member() (Organizations method), 21  
 add\_member() (Project method), 105  
 add\_member() (Projects method), 28  
 Annotation (class in *dtlpy.entities.annotation*), 121  
 AnnotationCollection (class in *dtlpy.entities.annotation\_collection*), 127  
 Annotations (class in *dtlpy.repositories.annotations*), 47  
 AnnotationStatus (class in *dtlpy.entities.annotation*), 126  
 AnnotationType (class in *dtlpy.entities.annotation*), 126  
 Assignment (class in *dtlpy.entities.assignment*), 147  
 Assignments (class in *dtlpy.repositories.assignments*), 63  
 attach\_agent\_progress() (Converter method), 169

## B

BaseTrigger (class in *dtlpy.entities.trigger*), 160  
 Bot (class in *dtlpy.entities.bot*), 159  
 Bots (class in *dtlpy.repositories.bots*), 86  
 Box (class in *dtlpy.entities.annotation\_definitions.box*), 131

build\_requirements() (Packages method), 67  
 build\_trigger\_dict() (Packages static method), 68  
 builder() (Annotations method), 47

## C

cache\_action() (Organization method), 101  
 cache\_action() (Organizations method), 21  
 CacheAction (class in *dtlpy.entities.organization*), 101  
 check\_cls\_arguments() (Packages static method), 68  
 checkout() (Dataset method), 108  
 checkout() (Datasets method), 32  
 checkout() (Package method), 150  
 checkout() (Packages method), 68  
 checkout() (Project method), 105  
 checkout() (Projects method), 28  
 checkout() (Service method), 156  
 checkout() (Services method), 79  
 Classification (class in *dtlpy.entities.annotation\_definitions.classification*), 131  
 clone() (Dataset method), 108  
 clone() (Datasets method), 32  
 clone() (Item method), 117  
 clone() (Items method), 40  
 clone() (Recipe method), 139  
 clone() (Recipes method), 51  
 clone\_git() (Codebases method), 75  
 Codebases (class in *dtlpy.repositories.codebases*), 75  
 Collection (class in *dtlpy.entities.similarity*), 135  
 CollectionItem (class in *dtlpy.entities.similarity*), 135  
 CollectionTypes (class in *dtlpy.entities.similarity*), 135  
 color\_map (Ontology property), 141  
 Command (class in *dtlpy.entities.command*), 167  
 Commands (class in *dtlpy.repositories.commands*), 99  
 CommandsStatus (class in *dtlpy.entities.command*), 167  
 convert() (Converter method), 169  
 convert\_dataset() (Converter method), 169  
 convert\_directory() (Converter method), 170  
 convert\_file() (Converter method), 170  
 Converter (class in *dtlpy.utilities.converter*), 169  
 create() (Assignments method), 63  
 create() (Bots method), 86

`create()` (*Datasets method*), 33  
`create()` (*Drivers method*), 39  
`create()` (*Executions method*), 90  
`create()` (*Integrations method*), 26  
`create()` (*Ontologies method*), 53  
`create()` (*PipelineExecutions method*), 98  
`create()` (*Pipelines method*), 94  
`create()` (*Projects method*), 29  
`create()` (*Recipes method*), 51  
`create()` (*Tasks method*), 57  
`create()` (*Triggers method*), 87  
`create_assignment()` (*Task method*), 144  
`create_qa_task()` (*Task method*), 145  
`create_qa_task()` (*Tasks method*), 58  
`CronTrigger` (*class in dtlpy.entities.trigger*), 160  
`Cube` (*class in dtlpy.entities.annotation\_definitions.cube*), 131  
`custom_format()` (*Converter static method*), 171

## D

`Dataset` (*class in dtlpy.entities.dataset*), 107  
`Datasets` (*class in dtlpy.repositories.datasets*), 32  
`delete()` (*Annotation method*), 122  
`delete()` (*AnnotationCollection method*), 127  
`delete()` (*Annotations method*), 47  
`delete()` (*BaseTrigger method*), 160  
`delete()` (*Bot method*), 159  
`delete()` (*Bots method*), 86  
`delete()` (*Dataset method*), 109  
`delete()` (*Datasets method*), 33  
`delete()` (*Integration method*), 104  
`delete()` (*Integrations method*), 26  
`delete()` (*Item method*), 118  
`delete()` (*Items method*), 41  
`delete()` (*Ontologies method*), 54  
`delete()` (*Ontology method*), 141  
`delete()` (*Package method*), 150  
`delete()` (*Packages method*), 69  
`delete()` (*Pipeline method*), 163  
`delete()` (*Pipelines method*), 94  
`delete()` (*Project method*), 105  
`delete()` (*Projects method*), 29  
`delete()` (*Recipe method*), 139  
`delete()` (*Recipes method*), 52  
`delete()` (*Service method*), 157  
`delete()` (*Services method*), 79  
`delete()` (*Task method*), 145  
`delete()` (*Tasks method*), 59  
`delete()` (*Triggers method*), 88  
`delete_attributes()` (*Dataset method*), 109  
`delete_attributes()` (*Ontologies method*), 54  
`delete_attributes()` (*Ontology method*), 141  
`delete_labels()` (*Dataset method*), 109  
`delete_labels()` (*Ontology method*), 141

`delete_member()` (*Organization method*), 102  
`delete_member()` (*Organizations method*), 22  
`deploy()` (*Package method*), 150  
`deploy()` (*Packages method*), 69  
`deploy()` (*Services method*), 79  
`deploy_from_file()` (*Packages method*), 70  
`deploy_from_local_folder()` (*Services method*), 81  
`Description` (*class in dtlpy.entities.annotation\_definitions.description*), 132  
`directory_tree()` (*Datasets method*), 34  
`DirectoryTree` (*class in dtlpy.entities.directory\_tree*), 168  
`download()` (*Annotation method*), 122  
`download()` (*AnnotationCollection method*), 128  
`download()` (*Annotations method*), 47  
`download()` (*Dataset method*), 110  
`download()` (*Item method*), 118  
`download()` (*Items method*), 41  
`download_annotations()` (*Dataset method*), 110  
`download_annotations()` (*Datasets static method*), 34  
`download_partition()` (*Dataset method*), 112  
`Driver` (*class in dtlpy.entities.driver*), 116  
`Drivers` (*class in dtlpy.repositories.drivers*), 39  
`dtlpy.entities.annotation`  
    *module*, 121  
`dtlpy.entities.annotation_collection`  
    *module*, 127  
`dtlpy.entities.annotation_definitions.base_annotation_defi`  
    *module*, 131  
`dtlpy.entities.annotation_definitions.box`  
    *module*, 131  
`dtlpy.entities.annotation_definitions.classification`  
    *module*, 131  
`dtlpy.entities.annotation_definitions.cube`  
    *module*, 131  
`dtlpy.entities.annotation_definitions.description`  
    *module*, 132  
`dtlpy.entities.annotation_definitions.ellipse`  
    *module*, 132  
`dtlpy.entities.annotation_definitions.note`  
    *module*, 132  
`dtlpy.entities.annotation_definitions.point`  
    *module*, 132  
`dtlpy.entities.annotation_definitions.polygon`  
    *module*, 133  
`dtlpy.entities.annotation_definitions.polyline`  
    *module*, 133  
`dtlpy.entities.annotation_definitions.pose`  
    *module*, 133  
`dtlpy.entities.annotation_definitions.segmentation`  
    *module*, 134  
`dtlpy.entities.annotation_definitions.subtitle`  
    *module*, 134

---

```

dtlpy.entities.annotation_definitions.undefined_annotation
    module, 134
dtlpy.entities.assignment
    module, 147
dtlpy.entities.base_entity
    module, 167
dtlpy.entities.bot
    module, 159
dtlpy.entities.codebase
    module, 154
dtlpy.entities.command
    module, 167
dtlpy.entities.dataset
    module, 107
dtlpy.entities.directory_tree
    module, 168
dtlpy.entities.driver
    module, 116
dtlpy.entities.execution
    module, 162
dtlpy.entities.filters
    module, 136
dtlpy.entities.integration
    module, 104
dtlpy.entities.item
    module, 117
dtlpy.entities.label
    module, 144
dtlpy.entities.links
    module, 121
dtlpy.entities.ontology
    module, 140
dtlpy.entities.organization
    module, 101
dtlpy.entities.package
    module, 150
dtlpy.entities.package_function
    module, 153
dtlpy.entities.package_module
    module, 153
dtlpy.entities.package_slot
    module, 154
dtlpy.entities.paged_entities
    module, 166
dtlpy.entities.pipeline
    module, 163
dtlpy.entities.pipeline_execution
    module, 165
dtlpy.entities.project
    module, 104
dtlpy.entities.recipe
    module, 139
dtlpy.entities.service
    module, 154
dtlpy.entities.similarity
    module, 135
dtlpy.entities.task
    module, 144
dtlpy.entities.trigger
    module, 160
dtlpy.entities.user
    module, 107
dtlpy.repositories.annotations
    module, 47
dtlpy.repositories.assignments
    module, 63
dtlpy.repositories.bots
    module, 86
dtlpy.repositories.codebases
    module, 75
dtlpy.repositories.commands
    module, 99
dtlpy.repositories.datasets
    module, 32
dtlpy.repositories.downloader
    module, 100
dtlpy.repositories.drivers
    module, 39
dtlpy.repositories.executions
    module, 90
dtlpy.repositories.integrations
    module, 26
dtlpy.repositories.items
    module, 40
dtlpy.repositories.ontologies
    module, 53
dtlpy.repositories.organizations
    module, 21
dtlpy.repositories.packages
    module, 67
dtlpy.repositories.pipeline_executions
    module, 98
dtlpy.repositories.pipelines
    module, 94
dtlpy.repositories.projects
    module, 28
dtlpy.repositories.recipes
    module, 51
dtlpy.repositories.services
    module, 78
dtlpy.repositories.tasks
    module, 56
dtlpy.repositories.triggers
    module, 87
dtlpy.repositories.uploader
    module, 100
dtlpy.utilities.converter
    module, 169

```

## E

`Ellipse` (class in `dtlpy.entities.annotation_definitions.ellipse`), 132

`execute()` (*Pipeline method*), 163

`execute()` (*Pipelines method*), 95

`execute()` (*Service method*), 157

`execute()` (*Services method*), 81

`Execution` (class in `dtlpy.entities.execution`), 162

`Executions` (class in `dtlpy.repositories.executions`), 90

`ExecutionStatus` (class in `dtlpy.entities.execution`), 163

`ExpirationOptions` (class in `dtlpy.entities.dataset`), 116

`ExportMetadata` (class in `dtlpy.entities.item`), 117

`ExportVersion` (class in `dtlpy.entities.annotation`), 126

`ExternalStorage` (class in `dtlpy.entities.driver`), 117

## F

`Filters` (class in `dtlpy.entities.filters`), 136

`FiltersKnownFields` (class in `dtlpy.entities.filters`), 138

`FiltersMethod` (class in `dtlpy.entities.filters`), 138

`FiltersOperations` (class in `dtlpy.entities.filters`), 138

`FiltersOrderByDirection` (class in `dtlpy.entities.filters`), 138

`FiltersResource` (class in `dtlpy.entities.filters`), 138

`FrameAnnotation` (class in `dtlpy.entities.annotation`), 126

`from_boxes_and_angle()` (*Cube class method*), 131

`from_coco()` (*Converter method*), 171

`from_instance_mask()` (*AnnotationCollection method*), 128

`from_json()` (*Annotation class method*), 123

`from_json()` (*AnnotationCollection class method*), 128

`from_json()` (*BaseTrigger class method*), 160

`from_json()` (*Bot class method*), 159

`from_json()` (*Command class method*), 167

`from_json()` (*CronTrigger class method*), 160

`from_json()` (*Dataset class method*), 112

`from_json()` (*Driver class method*), 116

`from_json()` (*Execution class method*), 162

`from_json()` (*Integration class method*), 104

`from_json()` (*Item class method*), 119

`from_json()` (*Ontology class method*), 142

`from_json()` (*Organization class method*), 102

`from_json()` (*Package class method*), 151

`from_json()` (*Pipeline class method*), 164

`from_json()` (*PipelineExecution class method*), 165

`from_json()` (*Project class method*), 105

`from_json()` (*Recipe class method*), 139

`from_json()` (*Service class method*), 157

`from_json()` (*Trigger class method*), 161

`from_json()` (*User class method*), 107

`from_polygon()` (*Segmentation class method*), 134

`from_segmentation()` (*Box class method*), 131

`from_segmentation()` (*Polygon class method*), 133

`from_snapshot()` (*FrameAnnotation class method*), 126

`from_voc()` (*Converter static method*), 171

`from_vtt_file()` (*AnnotationCollection method*), 129

`from_yolo()` (*Converter method*), 171

## G

`generate()` (*Packages static method*), 71

`generate()` (*Workload class method*), 149

`generate_url_query_params()` (*Filters method*), 137

`get()` (*Annotations method*), 48

`get()` (*Assignments method*), 63

`get()` (*Bots method*), 86

`get()` (*Codebases method*), 75

`get()` (*Commands method*), 99

`get()` (*Datasets method*), 35

`get()` (*Drivers method*), 39

`get()` (*Executions method*), 91

`get()` (*Integrations method*), 27

`get()` (*Items method*), 42

`get()` (*Ontologies method*), 54

`get()` (*Organizations method*), 23

`get()` (*Packages method*), 71

`get()` (*PipelineExecutions method*), 98

`get()` (*Pipelines method*), 95

`get()` (*Projects method*), 29

`get()` (*Recipes method*), 52

`get()` (*Services method*), 82

`get()` (*Tasks method*), 59

`get()` (*Triggers method*), 89

`get_all_items()` (*Items method*), 43

`get_annotation_template_id()` (*Recipe method*), 139

`get_current_version()` (*Codebases static method*), 76

`get_field()` (*LocalServiceRunner method*), 67

`get_frame()` (*AnnotationCollection method*), 129

`get_items()` (*Assignment method*), 147

`get_items()` (*Assignments method*), 63

`get_items()` (*Task method*), 146

`get_items()` (*Tasks method*), 59

`get_mainpy_run_service()` (*LocalServiceRunner method*), 67

`get_page()` (*PagedEntities method*), 166

`get_partitions()` (*Dataset method*), 112

`get_recipe_ids()` (*Dataset method*), 112

`go_to_page()` (*PagedEntities method*), 166

## H

`has_field()` (*Filters method*), 137

## I

`in_progress()` (*Command method*), 167

`increment()` (*Execution method*), 162

`increment()` (*Executions method*), 91



IndexDriver (class in *dtlpy.entities.dataset*), 116  
 install() (Pipeline method), 164  
 install() (Pipelines method), 96  
 instance\_map (Ontology property), 142  
 InstanceCatalog (class in *dtlpy.entities.service*), 154  
 Integration (class in *dtlpy.entities.integration*), 104  
 Integrations (class in *dtlpy.repositories.integrations*), 26  
 Item (class in *dtlpy.entities.item*), 117  
 Items (class in *dtlpy.repositories.items*), 40  
 items (MultiView property), 135  
 items (Similarity property), 135  
 ItemStatus (class in *dtlpy.entities.item*), 120

## K

KubernetesAutoscalerType (class in *dtlpy.entities.service*), 155

## L

labels\_to\_roots() (Ontologies static method), 55  
 LinkTypeEnum (class in *dtlpy.entities.links*), 121  
 list() (Annotations method), 48  
 list() (Assignments method), 64  
 list() (Bots method), 87  
 list() (Codebases method), 76  
 list() (Commands method), 99  
 list() (Datasets method), 36  
 list() (Drivers method), 40  
 list() (Executions method), 91  
 list() (Integrations method), 27  
 list() (Items method), 43  
 list() (Ontologies method), 55  
 list() (Organizations method), 23  
 list() (Packages method), 71  
 list() (PipelineExecutions method), 99  
 list() (Pipelines method), 96  
 list() (Projects method), 30  
 list() (Recipes method), 52  
 list() (Services method), 82  
 list() (Tasks method), 60  
 list() (Triggers method), 89  
 list\_groups() (Organization method), 102  
 list\_groups() (Organizations method), 23  
 list\_integrations() (Organizations method), 24  
 list\_members() (Organization method), 102  
 list\_members() (Organizations method), 24  
 list\_members() (Project method), 105  
 list\_members() (Projects method), 30  
 list\_versions() (Codebases method), 76  
 LocalServiceRunner (class in *dtlpy.repositories.packages*), 67  
 log() (Service method), 158  
 log() (Services method), 83  
 logs() (Execution method), 162

logs() (Executions method), 92

## M

make\_dir() (Items method), 44  
 MemberOrgRole (class in *dtlpy.entities.organization*), 101  
 MemberRole (class in *dtlpy.entities.project*), 104  
 merge() (Datasets method), 36  
 Message (class in *dtlpy.entities.annotation\_definitions.note*), 132  
 ModalityRefTypeEnum (class in *dtlpy.entities.item*), 120  
 ModalityTypeEnum (class in *dtlpy.entities.item*), 120  
 module  
   dtlpy.entities.annotation, 121  
   dtlpy.entities.annotation\_collection, 127  
   dtlpy.entities.annotation\_definitions.base\_annotation, 131  
   dtlpy.entities.annotation\_definitions.box, 131  
   dtlpy.entities.annotation\_definitions.classification, 131  
   dtlpy.entities.annotation\_definitions.cube, 131  
   dtlpy.entities.annotation\_definitions.description, 132  
   dtlpy.entities.annotation\_definitions.ellipse, 132  
   dtlpy.entities.annotation\_definitions.note, 132  
   dtlpy.entities.annotation\_definitions.point, 132  
   dtlpy.entities.annotation\_definitions.polygon, 133  
   dtlpy.entities.annotation\_definitions.polyline, 133  
   dtlpy.entities.annotation\_definitions.pose, 133  
   dtlpy.entities.annotation\_definitions.segmentation, 134  
   dtlpy.entities.annotation\_definitions.subtitle, 134  
   dtlpy.entities.annotation\_definitions.undefined\_annotation, 134  
   dtlpy.entities.assignment, 147  
   dtlpy.entities.base\_entity, 167  
   dtlpy.entities.bot, 159  
   dtlpy.entities.codebase, 154  
   dtlpy.entities.command, 167  
   dtlpy.entities.dataset, 107  
   dtlpy.entities.directory\_tree, 168  
   dtlpy.entities.driver, 116  
   dtlpy.entities.execution, 162  
   dtlpy.entities.filters, 136  
   dtlpy.entities.integration, 104

dtlpy.entities.item, 117  
dtlpy.entities.label, 144  
dtlpy.entities.links, 121  
dtlpy.entities.ontology, 140  
dtlpy.entities.organization, 101  
dtlpy.entities.package, 150  
dtlpy.entities.package\_function, 153  
dtlpy.entities.package\_module, 153  
dtlpy.entities.package\_slot, 154  
dtlpy.entities.paged\_entities, 166  
dtlpy.entities.pipeline, 163  
dtlpy.entities.pipeline\_execution, 165  
dtlpy.entities.project, 104  
dtlpy.entities.recipe, 139  
dtlpy.entities.service, 154  
dtlpy.entities.similarity, 135  
dtlpy.entities.task, 144  
dtlpy.entities.trigger, 160  
dtlpy.entities.user, 107  
dtlpy.repositories.annotations, 47  
dtlpy.repositories.assignments, 63  
dtlpy.repositories.bots, 86  
dtlpy.repositories.codebases, 75  
dtlpy.repositories.commands, 99  
dtlpy.repositories.datasets, 32  
dtlpy.repositories.downloader, 100  
dtlpy.repositories.drivers, 39  
dtlpy.repositories.executions, 90  
dtlpy.repositories.integrations, 26  
dtlpy.repositories.items, 40  
dtlpy.repositories.ontologies, 53  
dtlpy.repositories.organizations, 21  
dtlpy.repositories.packages, 67  
dtlpy.repositories.pipeline\_executions, 98  
dtlpy.repositories.pipelines, 94  
dtlpy.repositories.projects, 28  
dtlpy.repositories.recipes, 51  
dtlpy.repositories.services, 78  
dtlpy.repositories.tasks, 56  
dtlpy.repositories.triggers, 87  
dtlpy.repositories.uploader, 100  
dtlpy.utilities.converter, 169  
move() (Item method), 119  
move\_items() (Items method), 44  
MultiView (class in dtlpy.entities.similarity), 135  
MultiViewItem (class in dtlpy.entities.similarity), 135

## N

name\_validation() (Services method), 83  
name\_validation() (Triggers method), 89  
new() (Annotation class method), 123  
new() (FrameAnnotation class method), 126  
next\_page() (PagedEntities method), 166

Note (class in dtlpy.entities.annotation\_definitions.note), 132

## O

OnResetAction (class in dtlpy.entities.service), 155  
Ontologies (class in dtlpy.repositories.ontologies), 53  
Ontology (class in dtlpy.entities.ontology), 140  
open\_in\_web() (Assignment method), 147  
open\_in\_web() (Assignments method), 64  
open\_in\_web() (Dataset method), 113  
open\_in\_web() (Datasets method), 37  
open\_in\_web() (Filters method), 137  
open\_in\_web() (Item method), 119  
open\_in\_web() (Items method), 44  
open\_in\_web() (Organization method), 103  
open\_in\_web() (Package method), 151  
open\_in\_web() (Packages method), 72  
open\_in\_web() (Pipeline method), 164  
open\_in\_web() (Pipelines method), 96  
open\_in\_web() (Project method), 105  
open\_in\_web() (Projects method), 30  
open\_in\_web() (Recipe method), 140  
open\_in\_web() (Recipes method), 53  
open\_in\_web() (Service method), 158  
open\_in\_web() (Services method), 84  
open\_in\_web() (Task method), 146  
open\_in\_web() (Tasks method), 61  
Organization (class in dtlpy.entities.organization), 101  
Organizations (class in dtlpy.repositories.organizations), 21  
OrganizationsPlans (class in dtlpy.entities.organization), 103

## P

pack() (Codebases method), 76  
Package (class in dtlpy.entities.package), 150  
PackageFunction (class in dtlpy.entities.package\_function), 153  
PackageInputType (class in dtlpy.entities.package\_function), 153  
PackageModule (class in dtlpy.entities.package\_module), 153  
Packages (class in dtlpy.repositories.packages), 67  
PackageSlot (class in dtlpy.entities.package\_slot), 154  
PagedEntities (class in dtlpy.entities.paged\_entities), 166  
pause() (Pipeline method), 164  
pause() (Pipelines method), 97  
pause() (Service method), 158  
pause() (Services method), 84  
Pipeline (class in dtlpy.entities.pipeline), 163  
PipelineExecution (class in dtlpy.entities.pipeline\_execution), 165



PipelineExecutions (class in *dtlpy.repositories.pipeline\_executions*), 98  
 Pipelines (class in *dtlpy.repositories.pipelines*), 94  
 platform\_url() (Filters method), 137  
 PodType (class in *dtlpy.entities.organization*), 103  
 Point (class in *dtlpy.entities.annotation\_definitions.point*), 132  
 Polygon (class in *dtlpy.entities.annotation\_definitions.polygon*), 133  
 Polyline (class in *dtlpy.entities.annotation\_definitions.polyline*), 133  
 pop() (Collection method), 135  
 pop() (Filters method), 137  
 pop\_join() (Filters method), 137  
 Pose (class in *dtlpy.entities.annotation\_definitions.pose*), 133  
 prepare() (Filters method), 137  
 prev\_page() (PagedEntities method), 166  
 print() (AnnotationCollection method), 129  
 process\_result() (PagedEntities method), 166  
 progress\_update() (Execution method), 162  
 progress\_update() (Executions method), 92  
 Project (class in *dtlpy.entities.project*), 104  
 Projects (class in *dtlpy.repositories.projects*), 28  
 pull() (Package method), 151  
 pull() (Packages method), 72  
 pull\_git() (Codebases method), 77  
 push() (Package method), 151  
 push() (Packages method), 72  
**Q**  
 query() (Tasks method), 61  
**R**  
 reassign() (Assignment method), 147  
 reassign() (Assignments method), 65  
 Recipe (class in *dtlpy.entities.recipe*), 139  
 Recipes (class in *dtlpy.repositories.recipes*), 51  
 redistribute() (Assignment method), 148  
 redistribute() (Assignments method), 65  
 remove\_items() (Task method), 146  
 remove\_items() (Tasks method), 61  
 remove\_member() (Project method), 106  
 remove\_member() (Projects method), 31  
 RequirementOperator (class in *dtlpy.entities.package*), 153  
 rerun() (Execution method), 162  
 rerun() (Executions method), 93  
 reset() (Pipeline method), 164  
 reset() (Pipelines method), 97  
 resource\_information() (Triggers method), 89  
 resume() (Service method), 158  
 resume() (Services method), 84  
 return\_page() (PagedEntities method), 166  
 revisions() (Packages method), 73  
 revisions() (Services method), 85  
 run\_local\_project() (LocalServiceRunner method), 67  
 RuntimeType (class in *dtlpy.entities.service*), 155  
**S**  
 save\_to\_file() (Converter method), 172  
 Segmentation (class in *dtlpy.entities.annotation\_definitions.segmentation*), 134  
 serialize\_labels() (Dataset static method), 113  
 Service (class in *dtlpy.entities.service*), 156  
 ServiceLog (class in *dtlpy.repositories.services*), 78  
 Services (class in *dtlpy.repositories.services*), 78  
 set\_description() (Item method), 119  
 set\_frame() (Annotation method), 124  
 set\_items\_entity() (Items method), 45  
 set\_partition() (Dataset method), 113  
 set\_readonly() (Dataset method), 113  
 set\_readonly() (Datasets method), 37  
 set\_start\_node() (Pipeline method), 164  
 set\_status() (Assignment method), 148  
 set\_status() (Assignments method), 66  
 set\_status() (Task method), 146  
 set\_status() (Tasks method), 62  
 show() (Annotation method), 124  
 show() (AnnotationCollection method), 129  
 show() (Annotations method), 49  
 show() (Box method), 131  
 show() (Classification method), 131  
 show() (Cube method), 131  
 show() (Ellipse method), 132  
 show() (FrameAnnotation method), 126  
 show() (Point method), 132  
 show() (Polygon method), 133  
 show() (Polyline method), 133  
 show() (Pose method), 133  
 show() (Segmentation method), 134  
 show() (UndefinedAnnotationType method), 134  
 Similarity (class in *dtlpy.entities.similarity*), 135  
 SimilarityItem (class in *dtlpy.entities.similarity*), 136  
 SimilarityTypeEnum (class in *dtlpy.entities.similarity*), 136  
 SingleDirectory (class in *dtlpy.entities.directory\_tree*), 168  
 SlotDisplayScopeResource (class in *dtlpy.entities.package\_slot*), 154  
 SlotPostActionType (class in *dtlpy.entities.package\_slot*), 154  
 sort\_by() (Filters method), 138  
 stats() (Pipeline method), 164  
 stats() (Pipelines method), 97  
 status() (Service method), 158

status() (*Services method*), 85

Subtitle (*class in dtlpy.entities.annotation\_definitions.subtitle*), 134

switch\_recipe() (*Dataset method*), 113

sync() (*Dataset method*), 113

sync() (*Datasets method*), 37

## T

target (*Similarity property*), 136

Task (*class in dtlpy.entities.task*), 144

Tasks (*class in dtlpy.repositories.tasks*), 56

terminate() (*Execution method*), 163

terminate() (*Executions method*), 93

test() (*Package method*), 152

test\_local\_package() (*Packages method*), 74

to\_box() (*Segmentation method*), 134

to\_coco() (*Converter static method*), 172

to\_json() (*Annotation method*), 125

to\_json() (*AnnotationCollection method*), 130

to\_json() (*Assignment method*), 149

to\_json() (*BaseTrigger method*), 160

to\_json() (*Bot method*), 159

to\_json() (*Collection method*), 135

to\_json() (*Command method*), 167

to\_json() (*CronTrigger method*), 161

to\_json() (*Dataset method*), 114

to\_json() (*Driver method*), 117

to\_json() (*Execution method*), 163

to\_json() (*Integration method*), 104

to\_json() (*Item method*), 119

to\_json() (*MultiView method*), 135

to\_json() (*Ontology method*), 142

to\_json() (*Organization method*), 103

to\_json() (*Package method*), 153

to\_json() (*Pipeline method*), 164

to\_json() (*PipelineExecution method*), 165

to\_json() (*Project method*), 106

to\_json() (*Recipe method*), 140

to\_json() (*Service method*), 159

to\_json() (*Similarity method*), 136

to\_json() (*Task method*), 146

to\_json() (*Trigger method*), 161

to\_json() (*User method*), 107

to\_voc() (*Converter static method*), 172

to\_yolo() (*Converter method*), 173

Trigger (*class in dtlpy.entities.trigger*), 161

TriggerAction (*class in dtlpy.entities.trigger*), 161

TriggerExecutionMode (*class in dtlpy.entities.trigger*), 161

TriggerResource (*class in dtlpy.entities.trigger*), 161

Triggers (*class in dtlpy.repositories.triggers*), 87

TriggerType (*class in dtlpy.entities.trigger*), 161

## U

UnbindingPanel (*class in dtlpy.entities.package\_slot*), 154

UndefinedAnnotationType (*class in dtlpy.entities.annotation\_definitions.undefined\_annotation*), 134

unpack() (*Codebases method*), 77

update() (*Annotation method*), 125

update() (*AnnotationCollection method*), 130

update() (*Annotations method*), 49

update() (*Assignment method*), 149

update() (*Assignments method*), 66

update() (*BaseTrigger method*), 160

update() (*Dataset method*), 114

update() (*Datasets method*), 38

update() (*Execution method*), 163

update() (*Executions method*), 93

update() (*Integration method*), 104

update() (*Integrations method*), 27

update() (*Item method*), 120

update() (*Items method*), 45

update() (*Ontologies method*), 55

update() (*Ontology method*), 142

update() (*Organization method*), 103

update() (*Organizations method*), 25

update() (*Package method*), 153

update() (*Packages method*), 74

update() (*Pipeline method*), 165

update() (*Pipelines method*), 98

update() (*Project method*), 106

update() (*Projects method*), 31

update() (*Recipe method*), 140

update() (*Recipes method*), 53

update() (*Service method*), 159

update() (*Services method*), 85

update() (*Task method*), 147

update() (*Tasks method*), 62

update() (*Triggers method*), 90

update\_attributes() (*Dataset method*), 114

update\_attributes() (*Ontologies method*), 55

update\_attributes() (*Ontology method*), 142

update\_label() (*Dataset method*), 115

update\_label() (*Ontology method*), 143

update\_labels() (*Dataset method*), 115

update\_labels() (*Ontology method*), 143

update\_member() (*Organization method*), 103

update\_member() (*Organizations method*), 25

update\_member() (*Project method*), 106

update\_member() (*Projects method*), 31

update\_status() (*Annotation method*), 125

update\_status() (*Annotations method*), 50

update\_status() (*Item method*), 120

update\_status() (*Items method*), 45

upload() (*Annotation method*), 125

`upload()` (*AnnotationCollection method*), 130  
`upload()` (*Annotations method*), 50  
`upload()` (*Items method*), 46  
`upload_annotations()` (*Dataset method*), 116  
`upload_annotations()` (*Datasets method*), 38  
`upload_local_dataset()` (*Converter method*), 173  
`User` (*class in dtlpy.entities.user*), 107

## V

`view()` (*ServiceLog method*), 78  
`ViewAnnotationOptions` (*class in dtlpy.entities.annotation*), 126

## W

`wait()` (*Command method*), 167  
`wait()` (*Commands method*), 100  
`wait()` (*Execution method*), 163  
`wait()` (*Executions method*), 93  
`Workload` (*class in dtlpy.entities.assignment*), 149  
`WorkloadUnit` (*class in dtlpy.entities.assignment*), 149