
dtlpy Documentation

Release 1.51.18

Or Shabtay

Mar 10, 2022

TABLE OF CONTENTS

1	Command Line Interface	1
1.1	Positional Arguments	1
1.2	Named Arguments	1
1.3	Sub-commands:	1
1.3.1	shell	1
1.3.2	upgrade	2
1.3.2.1	optional named arguments	2
1.3.3	logout	2
1.3.4	login	2
1.3.5	login-token	2
1.3.5.1	required named arguments	2
1.3.6	login-secret	2
1.3.6.1	required named arguments	3
1.3.7	login-m2m	3
1.3.7.1	required named arguments	3
1.3.8	init	3
1.3.9	checkout-state	3
1.3.10	help	3
1.3.11	version	4
1.3.12	api	4
1.3.12.1	Positional Arguments	4
1.3.12.2	Sub-commands:	4
1.3.12.2.1	info	4
1.3.12.2.2	setenv	4
1.3.12.2.2.1	required named arguments	4
1.3.13	projects	4
1.3.13.1	Positional Arguments	5
1.3.13.2	Sub-commands:	5
1.3.13.2.1	ls	5
1.3.13.2.2	create	5
1.3.13.2.2.1	required named arguments	5
1.3.13.2.3	checkout	5
1.3.13.2.3.1	required named arguments	5
1.3.13.2.4	web	5
1.3.13.2.4.1	optional named arguments	6
1.3.14	datasets	6
1.3.14.1	Positional Arguments	6
1.3.14.2	Sub-commands:	6
1.3.14.2.1	web	6
1.3.14.2.1.1	optional named arguments	6

1.3.14.2.2	ls	6
1.3.14.2.2.1	optional named arguments	6
1.3.14.2.3	create	7
1.3.14.2.3.1	required named arguments	7
1.3.14.2.3.2	optional named arguments	7
1.3.14.2.4	checkout	7
1.3.14.2.4.1	required named arguments	7
1.3.14.2.4.2	optional named arguments	7
1.3.15	items	7
1.3.15.1	Positional Arguments	8
1.3.15.2	Sub-commands:	8
1.3.15.2.1	web	8
1.3.15.2.1.1	required named arguments	8
1.3.15.2.1.2	optional named arguments	8
1.3.15.2.2	ls	8
1.3.15.2.2.1	optional named arguments	8
1.3.15.2.3	upload	9
1.3.15.2.3.1	required named arguments	9
1.3.15.2.3.2	optional named arguments	9
1.3.15.2.4	download	9
1.3.15.2.4.1	optional named arguments	9
1.3.16	videos	10
1.3.16.1	Positional Arguments	10
1.3.16.2	Sub-commands:	10
1.3.16.2.1	play	10
1.3.16.2.1.1	optional named arguments	10
1.3.16.2.2	upload	10
1.3.16.2.2.1	required named arguments	11
1.3.16.2.2.2	optional named arguments	11
1.3.17	services	11
1.3.17.1	Positional Arguments	11
1.3.17.2	Sub-commands:	11
1.3.17.2.1	execute	11
1.3.17.2.1.1	optional named arguments	12
1.3.17.2.2	tear-down	12
1.3.17.2.2.1	optional named arguments	12
1.3.17.2.3	ls	12
1.3.17.2.3.1	optional named arguments	12
1.3.17.2.4	log	13
1.3.17.2.4.1	required named arguments	13
1.3.17.2.5	delete	13
1.3.17.2.5.1	optional named arguments	13
1.3.18	triggers	13
1.3.18.1	Positional Arguments	13
1.3.18.2	Sub-commands:	14
1.3.18.2.1	create	14
1.3.18.2.1.1	required named arguments	14
1.3.18.2.1.2	optional named arguments	14
1.3.18.2.2	delete	14
1.3.18.2.2.1	required named arguments	14
1.3.18.2.2.2	optional named arguments	15
1.3.18.2.3	ls	15
1.3.18.2.3.1	optional named arguments	15
1.3.19	deploy	15

1.3.19.1	required named arguments	15
1.3.20	generate	15
1.3.20.1	optional named arguments	15
1.3.21	packages	16
1.3.21.1	Positional Arguments	16
1.3.21.2	Sub-commands:	16
1.3.21.2.1	ls	16
1.3.21.2.1.1	optional named arguments	16
1.3.21.2.2	push	16
1.3.21.2.2.1	optional named arguments	16
1.3.21.2.3	test	16
1.3.21.2.3.1	optional named arguments	17
1.3.21.2.4	checkout	17
1.3.21.2.4.1	required named arguments	17
1.3.21.2.5	delete	17
1.3.21.2.5.1	optional named arguments	17
1.3.22	ls	17
1.3.23	pwd	17
1.3.24	cd	18
1.3.24.1	Positional Arguments	18
1.3.25	mkdir	18
1.3.25.1	Positional Arguments	18
1.3.26	clear	18
1.3.27	exit	18
2	Repositories	19
2.1	Organizations	19
2.1.1	Integrations	22
2.2	Projects	23
2.3	Datasets	26
2.3.1	Drivers	31
2.4	Items	32
2.5	Annotations	37
2.6	Recipes	40
2.6.1	Ontologies	42
2.7	Tasks	43
2.7.1	Assignments	48
2.8	Packages	51
2.8.1	Codebases	57
2.9	Services	59
2.9.1	Bots	65
2.10	Triggers	66
2.11	Executions	68
2.12	Pipelines	71
2.12.1	Pipeline Executions	73
2.13	General Commands	74
2.13.1	Download Commands	75
2.13.2	Upload Commands	75
3	Entities	77
3.1	Organization	77
3.1.1	Integration	79
3.2	Project	79
3.2.1	User	81

3.3	Dataset	82
3.3.1	Driver	87
3.4	Item	88
3.4.1	Item Link	90
3.5	Annotation	90
3.5.1	Collection of Annotation entities	94
3.5.2	Annotation Definition	96
3.5.2.1	Box Annotation Definition	96
3.5.2.2	Classification Annotation Definition	97
3.5.2.3	Cuboid Annotation Definition	97
3.5.2.4	Item Description Definition	97
3.5.2.5	Ellipse Annotation Definition	97
3.5.2.6	Note Annotation Definition	98
3.5.2.7	Point Annotation Definition	98
3.5.2.8	Polygon Annotation Definition	98
3.5.2.9	Polyline Annotation Definition	99
3.5.2.10	Pose Annotation Definition	99
3.5.2.11	Segmentation Annotation Definition	99
3.5.2.12	Audio Annotation Definition	100
3.5.2.13	Undefined Annotation Definition	100
3.5.3	Similarity	100
3.6	Filter	101
3.7	Recipe	103
3.7.1	Ontology	104
3.7.1.1	Label	106
3.8	Task	106
3.8.1	Assignment	108
3.9	Package	110
3.9.1	Package Function	112
3.9.2	Package Module	112
3.9.3	Slot	112
3.9.4	Codebase	113
3.10	Service	113
3.10.1	Bot	116
3.11	Trigger	116
3.12	Execution	118
3.13	Pipeline	119
3.13.1	Pipeline Execution	121
3.14	Other	121
3.14.1	Pages	121
3.14.2	Base Entity	122
3.14.3	Command	122
3.14.4	Directory Tree	123
4	Utilities	125
4.1	converter	125
5	Tutorials	131
5.1	Data Management Tutorial	131
5.1.1	Connect Cloud Storage	131
5.1.1.1	Connect Cloud Storage	131
5.1.1.1.1	Cloud Storage Integration	131
5.1.1.1.2	Create Integration With GCS	131

5.1.1.1.2.1	Creating an integration GCS requires having JSON file with GCS configuration.	131
5.1.1.1.2.2	Create Integration With S3	132
5.1.1.1.2.3	Create Integration With Azure	132
5.1.1.1.3	Storage Driver	132
5.1.1.1.4	Create Drivers in the Platform (browser)	132
5.1.2	Manage Datasets	133
5.1.2.1	Manage Datasets	133
5.1.2.1.1	Create Dataset	133
5.1.2.1.2	Create Dataset With Cloud Storage Driver	133
5.1.2.1.3	Retrieve Datasets	133
5.1.2.1.4	Create Directory	134
5.1.2.1.5	Hard-copy a Folder to Another Dataset	134
5.1.3	Data Versioning	135
5.1.3.1	Data Versioning	135
5.1.3.1.1	Clone Datasets	135
5.1.3.1.2	Merge Datasets	135
5.1.4	Upload and Manage Data and Metadata	136
5.1.4.1	Upload & Manage Data & Metadata	136
5.1.4.1.1	Upload specific files	136
5.1.4.1.2	Upload all files in a folder	136
5.1.4.1.3	Upload items from URL link	136
5.1.4.1.3.1	Additional upload options	137
5.1.4.1.4	Upload Items and Annotations Metadata	137
5.1.5	Upload and Manage Annotations	137
5.1.5.1	Upload & Manage Annotations	137
5.1.5.1.1	Convert Annotations To COCO Format	138
5.1.5.1.2	Upload Entire Directory and their Corresponding Dataloop JSON Annotations	138
5.1.5.1.3	Upload Annotations To Video Item	138
5.1.5.2	Show Annotations Over Image	139
5.1.5.3	Download Data, Annotations & Metadata	139
5.1.5.3.1	Download Items and Annotations	139
5.1.5.3.2	Multiple Annotation Options	140
5.1.5.3.3	Filter by Item and/or Annotation	140
5.1.5.3.4	Filter by Annotations	140
5.1.5.3.5	Download Annotations in COCO Format	141
5.2	FaaS Tutorial	141
5.2.1	FaaS Interactive Tutorial – Using Python & Dataloop SDK	141
5.2.1.1	FaaS Interactive Tutorial – Using Python & Dataloop SDK	141
5.2.1.1.1	Concept	141
5.2.1.2	Use Cases	142
5.2.2	Introduction	142
5.2.2.1	Introduction	142
5.2.3	Run Your First Function	143
5.2.3.1	Basic Use Case: Single Function	143
5.2.3.1.1	Create and Deploy a Sample Function	143
5.2.3.1.2	Execute the function	144
5.2.4	Multiple Function	144
5.2.4.1	Advanced Use Case: Multiple Functions	144
5.2.4.1.1	Create and Deploy a Package of Several Functions	144
5.2.4.1.2	Write your code	144
5.2.4.1.3	Define the module	146
5.2.4.1.4	Push the package	146

5.2.4.1.5	Deploy a service	146
5.2.4.1.6	Trigger the service	147
5.2.4.1.7	Execute the function	148
5.2.4.1.8	Review the function's logs	148
5.2.4.1.9	Pause the service:	148
5.3	Model Management	148
5.3.1	Introduction	148
5.3.1.1	Model Management	148
5.3.1.1.1	Introduction	148
5.3.1.1.1.1	Model and Snapshot entities	149
5.3.1.1.1.2	Model	149
5.3.1.1.1.3	Snapshot	149
5.3.1.1.1.4	Buckets and Codebase	149
5.3.1.1.1.5	The Model Adapter	149
5.3.2	Create a Model and Snapshot	150
5.3.2.1	Create Your own Model and Snapshot	150
5.3.3	Using Dataloop's Dataset Generator	151
5.3.3.1	Dataloop Dataloader	151
5.3.3.1.1	Object Detection Examples	152
5.3.3.1.2	Segmentation Examples	153
6	Indices and tables	155
	Python Module Index	157
	Index	159

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

operation

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

-v, --version

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 ]
```

1.3.2.1 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
```

1.3.5.1 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 ]
```

1.3.6.1 required named arguments

-e, --email	user email
-p, --password	user password
-i, --client-id	client id
-s, --client-secret	client secret

1.3.7 login-m2m

Login client id and secret

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

1.3.7.1 required named arguments

-e, --email	user email
-p, --password	user password
-i, --client-id	client id
-s, --client-secret	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} ...
```

1.3.12.1 Positional Arguments

api	Possible choices: info, setenv
	gate operations

1.3.12.2 Sub-commands:

1.3.12.2.1 info

Print api information

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

1.3.12.2.2 setenv

Set platform environment

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

1.3.12.2.2.1 required named arguments

-e, --env	working environment
------------------	---------------------

1.3.13 projects

Operations with projects

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

1.3.13.1 Positional Arguments

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

1.3.13.2 Sub-commands:

1.3.13.2.1 ls

List all projects

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

1.3.13.2.2 create

Create a new project

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

1.3.13.2.2.1 required named arguments

-p, --project-name project name

1.3.13.2.3 checkout

checkout a project

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

1.3.13.2.3.1 required named arguments

-p, --project-name project name

1.3.13.2.4 web

Open in web browser

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

1.3.13.2.4.1 optional named arguments

-p, --project-name project name

1.3.14 datasets

Operations with datasets

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

1.3.14.1 Positional Arguments

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

1.3.14.2 Sub-commands:

1.3.14.2.1 web

Open in web browser

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

1.3.14.2.1.1 optional named arguments

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

1.3.14.2.2 ls

List of datasets in project

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

1.3.14.2.2.1 optional named arguments

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

1.3.14.2.3 create

Create a new dataset

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

1.3.14.2.3.1 required named arguments

-d, --dataset-name dataset name

1.3.14.2.3.2 optional named arguments

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

-c, --checkout checkout the new dataset

Default: False

1.3.14.2.4 checkout

checkout a dataset

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

1.3.14.2.4.1 required named arguments

-d, --dataset-name dataset name

1.3.14.2.4.2 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} ...
```

1.3.15.1 Positional Arguments

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

1.3.15.2 Sub-commands:

1.3.15.2.1 web

Open in web browser

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

1.3.15.2.1.1 required named arguments

-r, --remote-path remote path

1.3.15.2.1.2 optional named arguments

-p, --project-name project name

-d, --dataset-name dataset name

1.3.15.2.2 ls

List of items in dataset

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

1.3.15.2.2.1 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

1.3.15.2.3 upload

Upload directory to dataset

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

1.3.15.2.3.1 required named arguments

-l, --local-path local path

1.3.15.2.3.2 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

1.3.15.2.4 download

Download dataset to a local directory

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

1.3.15.2.4.1 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} ...
```

1.3.16.1 Positional Arguments

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

1.3.16.2 Sub-commands:

1.3.16.2.1 play

Play video

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

1.3.16.2.1.1 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)

1.3.16.2.2 upload

Upload a single video

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

1.3.16.2.2.1 required named arguments

-f, --filename local filename to upload
-p, --project-name project name
-d, --dataset-name dataset name

1.3.16.2.2.2 optional named arguments

-r, --remote-path remote path
 Default: “/”
-sc, --split-chunks Video splitting parameter: Number of chunks to split
-ss, --split-seconds Video splitting parameter: Seconds of each chunk
-st, --split-times Video splitting parameter: List of seconds to split at. e.g 600,1800,2000
-e, --encode 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} ...
```

1.3.17.1 Positional Arguments

services Possible choices: execute, tear-down, ls, log, delete
 services operations

1.3.17.2 Sub-commands:

1.3.17.2.1 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]
```

1.3.17.2.1.1 optional named arguments

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

1.3.17.2.2 tear-down

tear-down service of service.json file

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

1.3.17.2.2.1 optional named arguments

-l, --local-path	path to service.json file
-pr, --project-name	Project name

1.3.17.2.3 ls

List project's services

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

1.3.17.2.3.1 optional named arguments

-pr, --project-name	Project name
-pkg, --package-name	Package name

1.3.17.2.4 log

Get services log

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

1.3.17.2.4.1 required named arguments

-pr, --project-name	Project name
-f, --service-name	Project name
-t, --start	Log start time

1.3.17.2.5 delete

Delete Service

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

1.3.17.2.5.1 optional named arguments

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

1.3.18 triggers

Operations with triggers

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

1.3.18.1 Positional Arguments

triggers	Possible choices: create, delete, ls triggers operations
-----------------	---

1.3.18.2 Sub-commands:

1.3.18.2.1 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]
```

1.3.18.2.1.1 required named arguments

-r, --resource	Resource name
-a, --actions	Actions

1.3.18.2.1.2 optional named arguments

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

1.3.18.2.2 delete

Delete Trigger

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

1.3.18.2.2.1 required named arguments

-t, --trigger-name	Trigger name
---------------------------	--------------

1.3.18.2.2.2 optional named arguments

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

1.3.18.2.3 ls

List triggers

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

1.3.18.2.3.1 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]
```

1.3.19.1 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]
```

1.3.20.1 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} ...
```

1.3.21.1 Positional Arguments

packages	Possible choices: ls, push, test, checkout, delete package operations
-----------------	--

1.3.21.2 Sub-commands:

1.3.21.2.1 ls

List packages

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

1.3.21.2.1.1 optional named arguments

-p, --project-name	Project name
---------------------------	--------------

1.3.21.2.2 push

Create package in platform

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

1.3.21.2.2.1 optional named arguments

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

1.3.21.2.3 test

Tests that Package locally using mock.json

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


1.3.21.2.3.1 optional named arguments

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

1.3.21.2.4 checkout

checkout a package

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

1.3.21.2.4.1 required named arguments

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

1.3.21.2.5 delete

Delete Package

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

1.3.21.2.5.1 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
```

1.3.24.1 Positional Arguments

dir

1.3.25 mkdir

Make directory

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

1.3.25.1 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*: `dtlpy.services.api_client.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*: `dtlpy.entities.organization.MemberOrgRole` = `MemberOrgRole.MEMBER`,
organization_id: `Optional[str]` = `None`, *organization_name*: `Optional[str]` = `None`,
organization: `Optional[dtlpy.entities.organization.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`

delete_member(*user_id*: `str`, *organization_id*: `Optional[str]` = `None`, *organization_name*: `Optional[str]` = `None`, *organization*: `Optional[dtlpy.entities.organization.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*

get(*organization_id: Optional[str] = None, organization_name: Optional[str] = None, fetch: Optional[bool] = None*) → *dtlpy.entities.organization.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*

list() → *dtlpy.miscellaneous.list_print.List[dtlpy.entities.organization.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*

list_groups(*organization: Optional[dtlpy.entities.organization.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*

list_integrations(*organization: Optional[dtlpy.entities.organization.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*

list_members(*organization: Optional[dtlpy.entities.organization.Organization] = None, organization_id: Optional[str] = None, organization_name: Optional[str] = None, role: Optional[dtlpy.entities.organization.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*

update(*plan: str, organization: Optional[dtlpy.entities.organization.Organization] = None, organization_id: Optional[str] = None, organization_name: Optional[str] = None*) → *dtlpy.entities.organization.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*

```
update_member(email: str, role: dtlpy.entities.organization.MemberOrgRole = MemberOrgRole.MEMBER,
               organization_id: Optional[str] = None, organization_name: Optional[str] = None,
               organization: Optional[dtlpy.entities.organization.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*

2.1.1 Integrations

Integrations Repository

```
class Integrations(client_api: dtlpy.services.api_client.ApiClient, org:
                    Optional[dtlpy.entities.organization.Organization] = None, project:
                    Optional[dtlpy.entities.project.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: dtlpy.entities.driver.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: ""}

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*

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*

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*

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*

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*

2.2 Projects

class Projects(*client_api*: *dtlpy.services.api_client.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*: `dtlpy.entities.project.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** – “owner”, “engineer”, “annotator”, “annotationManager”

Returns dict that represent the user

Return type `dict`

checkout(*identifier*: *Optional*[*str*] = *None*, *project_name*: *Optional*[*str*] = *None*, *project_id*: *Optional*[*str*] = *None*, *project*: *Optional*[`dtlpy.entities.project.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

create(*project_name*: *str*, *checkout*: *bool* = *False*) → `dtlpy.entities.project.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`

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`

get(*project_name*: *Optional*[*str*] = *None*, *project_id*: *Optional*[*str*] = *None*, *checkout*: *bool* = *False*, *fetch*: *Optional*[*bool*] = *None*) → *dtlpy.entities.project.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

Returns Project object

Return type *dtlpy.entities.project.Project*

list() → *dtlpy.miscellaneous.list_print.List*[*dtlpy.entities.project.Project*]

Get users' project list.

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

Returns List of Project objects

list_members(*project*: *dtlpy.entities.project.Project*, *role*: *Optional*[*dtlpy.entities.project.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** – “owner”, “engineer”, “annotator”, “annotationManager”

Returns list of the project members

Return type `list`

open_in_web(*project_name*: *Optional*[*str*] = *None*, *project_id*: *Optional*[*str*] = *None*, *project*: *Optional*[*dtlpy.entities.project.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

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*

update(*project*: *dtlpy.entities.project.Project*, *system_metadata*: *bool* = *False*) → *dtlpy.entities.project.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*

update_member(*email*: *str*, *project_id*: *str*, *role*: *dtlpy.entities.project.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** – “owner”, “engineer”, “annotator”, “annotationManager”

Returns dict that represent the user

Return type *dict*

2.3 Datasets

Datasets Repository

class Datasets(*client_api*: *dtlpy.services.api_client.ApiClient*, *project*: *Optional*[*dtlpy.entities.project.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*[*dtlpy.entities.dataset.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

clone(*dataset_id: str, clone_name: str, filters: Optional[dtlpy.entities.filters.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`

create(*dataset_name: str, labels=None, attributes=None, ontology_ids=None, driver: Optional[dtlpy.entities.driver.Driver] = None, driver_id: Optional[str] = None, checkout: bool = False, expiration_options: Optional[dtlpy.entities.dataset.ExpirationOptions] = None*) → `dtlpy.entities.dataset.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

Returns Dataset object

Return type `dtlpy.entities.dataset.Dataset`

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.

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[dtlpy.entities.dataset.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*

static download_annotations(*dataset: dtlpy.entities.dataset.Dataset, local_path: Optional[str] = None, filters: Optional[dtlpy.entities.filters.Filters] = None, annotation_options: Optional[dtlpy.entities.annotation.ViewAnnotationOptions] = None, annotation_filters: Optional[dtlpy.entities.filters.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, *VI* - no original extension in filenames

Returns local_path of the directory where all the downloaded item

Return type *str*

get(dataset_name: *Optional[str]* = None, dataset_id: *Optional[str]* = None, checkout: *bool* = False, fetch: *Optional[bool]* = None) → *dtlpy.entities.dataset.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*

list(name=None, creator=None) → *dtlpy.miscellaneous.list_print.List[dtlpy.entities.dataset.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*

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

open_in_web(*dataset_name: Optional[str] = None, dataset_id: Optional[str] = None, dataset: Optional[dtlpy.entities.dataset.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

set_readonly(*state: bool, dataset: dtlpy.entities.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

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

update(*dataset: dtlpy.entities.dataset.Dataset, system_metadata: bool = False, patch: Optional[dict] = None*) → *dtlpy.entities.dataset.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`

upload_annotations(*dataset*, *local_path*, *filters*: *Optional*[`dtlpy.entities.filters.Filters`] = *None*, *clean*=*False*, *remote_root_path*='/', *export_version*=*ExportVersion.VI*)

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, *VI* - no original extension in filenames

2.3.1 Drivers

class Drivers(*client_api*: `dtlpy.services.api_client.ApiClient`, *project*: *Optional*[`dtlpy.entities.project.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*: `dtlpy.entities.driver.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*

get(*driver_name: Optional[str] = None, driver_id: Optional[str] = None*) → *dtlpy.entities.driver.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*

list() → *dtlpy.miscellaneous.list_print.List[dtlpy.entities.driver.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*

2.4 Items

class Items(*client_api: dtlpy.services.api_client.ApiClient, datasets: Optional[dtlpy.repositories.datasets.Datasets] = None, dataset: Optional[dtlpy.entities.dataset.Dataset] = None, dataset_id=None, items_entity=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`

delete(*filename: Optional[str] = None, item_id: Optional[str] = None, filters: Optional[dtlpy.entities.filters.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*

download(*filters: Optional[dtlpy.entities.filters.Filters] = None, items=None, local_path: Optional[str] = None, file_types: Optional[dtlpy.repositories.items.Items.list] = None, save_locally: bool = True, to_array: bool = False, annotation_options: Optional[dtlpy.entities.annotation.ViewAnnotationOptions] = None, annotation_filters: Optional[dtlpy.entities.filters.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: Optional[float] = None, 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 Nddarray 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

get(filepath: *Optional[str]* = None, item_id: *Optional[str]* = None, fetch: *Optional[bool]* = None, is_dir: *bool* = False) → `dtlpy.entities.item.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`

get_all_items()
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`

list(*filters*: *Optional*[`dtlpy.entities.filters.Filters`] = *None*, *page_offset*: *Optional*[*int*] = *None*, *page_size*: *Optional*[*int*] = *None*) → *dtlpy.entities.paged_entities.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*

make_dir(*directory*, *dataset*: *Optional*[`dtlpy.entities.dataset.Dataset`] = *None*) → *dtlpy.entities.item.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*

move_items(*destination*: *str*, *filters*: *Optional*[`dtlpy.entities.filters.Filters`] = *None*, *items*=*None*, *dataset*: *Optional*[`dtlpy.entities.dataset.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`

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

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*[`dtlpy.entities.item.Item`] = *None*, *filters*: *Optional*[`dtlpy.entities.filters.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`

update_status(*status*: `dtlpy.entities.item.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

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.V1`)

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

2.5 Annotations

class Annotations(*client_api: dtlpy.services.api_client.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`

delete(*annotation: Optional[dtlpy.entities.annotation.Annotation] = None, annotation_id: Optional[str] = None, filters: Optional[dtlpy.entities.filters.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`

download(*filepath*: `str`, *annotation_format*: `dtlpy.entities.annotation.ViewAnnotationOptions` = `ViewAnnotationOptions.MASK`, *img_filepath*: `Optional[str]` = `None`, *height*: `Optional[float]` = `None`, *width*: `Optional[float]` = `None`, *thickness*: `int` = 1, *with_text*: `bool` = `False`, *alpha*: `Optional[float]` = `None`)

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(`dtlpy.entities.annotation.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`

get(*annotation_id*: `str`) → `dtlpy.entities.annotation.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`

list(*filters*: `Optional[dtlpy.entities.filters.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`

show(*image=None, thickness: int = 1, with_text: bool = False, height: Optional[float] = None, width: Optional[float] = None, annotation_format: dtlpy.entities.annotation.ViewAnnotationOptions = ViewAnnotationOptions.MASK, alpha: Optional[float] = None*)

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

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

update_status(*annotation: Optional[dtlpy.entities.annotation.Annotation] = None, annotation_id: Optional[str] = None, status: dtlpy.entities.annotation.AnnotationStatus = AnnotationStatus.ISSUE*) → `dtlpy.entities.annotation.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`

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`

2.6 Recipes

```
class Recipes(client_api: dtlpy.services.api_client.ApiClient, dataset: Optional[dtlpy.entities.dataset.Dataset] =  
None, project: Optional[dtlpy.entities.project.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[dtlpy.entities.recipe.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`

```
create(project_ids=None, ontology_ids=None, labels=None, recipe_name=None, attributes=None) →  
dtlpy.entities.recipe.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`

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`

get(*recipe_id*: `str`) → `dtlpy.entities.recipe.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`

list(*filters*: `Optional[dtlpy.entities.filters.Filters]` = `None`) →

`dtlpy.miscellaneous.list_print.List[dtlpy.entities.recipe.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

open_in_web(*recipe*: `Optional[dtlpy.entities.recipe.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

update(*recipe*: `dtlpy.entities.recipe.Recipe`, *system_metadata*=`False`) → `dtlpy.entities.recipe.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*

2.6.1 Ontologies

```
class Ontologies(client_api: dtlpy.services.api_client.ApiClient, recipe: Optional[dtlpy.entities.recipe.Recipe]
                 = None, project: Optional[dtlpy.entities.project.Project] = None, dataset:
                 Optional[dtlpy.entities.dataset.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) → *dtlpy.entities.ontology.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*

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`

get(ontology_id: *str*) → *dtlpy.entities.ontology.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*

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*) → dtlpy.miscellaneous.list_print.List[*dtlpy.entities.ontology.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

update(*ontology: dtlpy.entities.ontology.Ontology, system_metadata=False*) →
dtlpy.entities.ontology.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*

2.7 Tasks

class Tasks(*client_api: dtlpy.services.api_client.ApiClient, project: Optional[dtlpy.entities.project.Project] = None, dataset: Optional[dtlpy.entities.dataset.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[dtlpy.entities.task.Task] = None, task_id=None, filters: Optional[dtlpy.entities.filters.Filters] = None, items=None, assignee_ids=None, query=None, workload=None, limit=None, wait=True*) → *dtlpy.entities.task.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 per assignee and work load
- **limit** – task limit

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

Returns task entity

Return type *dtlpy.entities.task.Task*

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*: *dtlpy.entities.filters.Filters* = False) → *dtlpy.entities.task.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

Returns Annotation Task object

Return type *dtlpy.entities.task.Task*

create_qa_task(*task*: *dtlpy.entities.task.Task*, *assignee_ids*, *due_date*=None, *filters*=None, *items*=None, *query*=None, *workload*=None, *metadata*=None, *available_actions*=None, *wait*=True) → *dtlpy.entities.task.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

Returns task object

Return type `dtlpy.entities.task.Task`

delete(`task: Optional[dtlpy.entities.task.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`

get(`task_name=None, task_id=None`) → `dtlpy.entities.task.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`

get_items(`task_id: Optional[str] = None, task_name: Optional[str] = None, dataset: Optional[dtlpy.entities.dataset.Dataset] = None, filters: Optional[dtlpy.entities.filters.Filters] = None`) → `dtlpy.entities.paged_entities.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`

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[dtlpy.entities.filters.Filters] = None*) →
`Union[dtlpy.miscellaneous.list_print.List[dtlpy.entities.task.Task], dtlpy.entities.paged_entities.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

open_in_web(*task_name: Optional[str] = None, task_id: Optional[str] = None, task: Optional[dtlpy.entities.task.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

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`

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`

update(*task: Optional[dtlpy.entities.task.Task] = None, system_metadata=False*) → `dtlpy.entities.task.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`

2.7.1 Assignments

```
class Assignments(client_api: dtlpy.services.api_client.ApiClient, project:
    Optional[dtlpy.entities.project.Project] = None, task: Optional[dtlpy.entities.task.Task] =
    None, dataset: Optional[dtlpy.entities.dataset.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[dtlpy.entities.task.Task] = None, filters:
    Optional[dtlpy.entities.filters.Filters] = None, items:
    Optional[dtlpy.repositories.assignments.Assignments.list] = None) →
    dtlpy.entities.assignment.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`

```
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`

```
get_items(assignment: Optional[dtlpy.entities.assignment.Assignment] = None, assignment_id=None,
    assignment_name=None, dataset=None, filters=None) →
    dtlpy.entities.paged_entities.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`

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`) → `dtlpy.miscellaneous.list_print.List[dtlpy.entities.assignment.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]`

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

reassign(*assignee_id*: `str`, *assignment*: `Optional[dtlpy.entities.assignment.Assignment] = None`, *assignment_id*: `Optional[str] = None`, *task*: `Optional[dtlpy.entities.task.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*

redistribute(*workload*: *dtlpy.entities.assignment.Workload*, *assignment*: *Optional*[*dtlpy.entities.assignment.Assignment*] = *None*, *assignment_id*: *Optional*[*str*] = *None*, *task*: *Optional*[*dtlpy.entities.task.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.

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

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*

update(*assignment*: *Optional*[*dtlpy.entities.assignment.Assignment*] = *None*, *system_metadata*: *bool* = *False*) → *dtlpy.entities.assignment.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

2.8 Packages

```
class LocalServiceRunner(client_api: dtlpy.services.api_client.ApiClient, packages, cwd=None,
                           multithreading=False, concurrency=10, package:
                           Optional[dtlpy.entities.package.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: dtlpy.services.api_client.ApiClient, project: Optional[dtlpy.entities.project.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) → dtlpy.repositories.packages.Packages.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: dtlpy.entities.trigger.TriggerExecutionMode = 'Once',
                          type_t: dtlpy.entities.trigger.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`

```
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[dtlpy.entities.package.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

```
delete(package: Optional[dtlpy.entities.package.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

deploy(*package_id*: *Optional[str]* = None, *package_name*: *Optional[str]* = None, *package*: *Optional[dtlpy.entities.package.Package]* = None, *service_name*: *Optional[str]* = None, *project_id*: *Optional[str]* = None, *revision*: *Optional[str]* = None, *init_input*: *Optional[Union[List[dtlpy.entities.package_function.FunctionIO], dtlpy.entities.package_function.FunctionIO, dict]]* = None, *runtime*: *Optional[Union[dtlpy.entities.service.KubernetesRuntime, dict]]* = None, *sdk_version*: *Optional[str]* = None, *agent_versions*: *Optional[dict]* = None, *bot*: *Optional[Union[dtlpy.entities.bot.Bot, str]]* = None, *pod_type*: *Optional[dtlpy.entities.service.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, ***kwargs*) → *dtlpy.entities.service.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

Returns Service object

Return type `dtlpy.entities.service.Service`

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

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

get(*package_name: Optional[str] = None*, *package_id: Optional[str] = None*, *checkout: bool = False*, *fetch=None*) → `dtlpy.entities.package.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`

list(*filters: Optional[dtlpy.entities.filters.Filters] = None*, *project_id: Optional[str] = None*) → `dtlpy.entities.paged_entities.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`

open_in_web(*package*: `Optional[dtlpy.entities.package.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

pull(*package*: `dtlpy.entities.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`

push(*project*: `Optional[dtlpy.entities.project.Project]` = `None`, *project_id*: `Optional[str]` = `None`, *package_name*: `Optional[str]` = `None`, *src_path*: `Optional[str]` = `None`, *codebase*: `Optional[Union[dtlpy.entities.codebase.GitCodebase, dtlpy.entities.codebase.ItemCodebase, dtlpy.entities.codebase.FilesystemCodebase]]` = `None`, *modules*: `Optional[List[dtlpy.entities.package_module.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[dtlpy.entities.package_slot.PackageSlot]]` = `None`, *requirements*: `Optional[List[dtlpy.entities.package.PackageRequirement]]` = `None`) → `dtlpy.entities.package.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 of 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*

revisions(*package*: *Optional*[*dtlpy.entities.package.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

test_local_package(*cwd*: *Optional*[*str*] = None, *concurrency*: *Optional*[*int*] = None, *package*: *Optional*[*dtlpy.entities.package.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*

update(*package*: *dtlpy.entities.package.Package*, *revision_increment*: *Optional*[*str*] = None) → *dtlpy.entities.package.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`

2.8.1 Codebases

```
class Codebases(client_api: dtlpy.services.api_client.ApiClient, project: Optional[dtlpy.entities.project.Project]
                = None, dataset: Optional[dtlpy.entities.dataset.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: dtlpy.entities.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`

```
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.

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`

list() → `dtlpy.entities.paged_entities.PagedEntities`

List all codebases.

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

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.

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`

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`

unpack(codebase: Optional[dtlpy.entities.codebase.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`) – *dl.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*

2.9 Services

```
class ServiceLog(_json: dict, service: dtlpy.entities.service.Service, services:
    dtlpy.repositories.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: dtlpy.services.api_client.ApiClient, project: Optional[dtlpy.entities.project.Project] =
    None, package: Optional[dtlpy.entities.package.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: dtlpy.entities.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[dtlpy.entities.package_slot.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*

checkout (*service: Optional[dtlpy.entities.service.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

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*

deploy (*service_name: Optional[str] = None, package: Optional[dtlpy.entities.package.Package] = None, bot: Optional[Union[dtlpy.entities.bot.Bot, str]] = None, revision: Optional[str] = None, init_input: Optional[Union[List[dtlpy.entities.package_function.FunctionIO], dtlpy.entities.package_function.FunctionIO, dict]] = None, runtime: Optional[Union[dtlpy.entities.service.KubernetesRuntime, dict]] = None, pod_type: Optional[dtlpy.entities.service.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[dtlpy.repositories.services.Services.list] = None, **kwargs*) → *dtlpy.entities.service.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*

deploy_from_local_folder(*cwd=None, service_file=None, bot=None, checkout=False, force=False*) → *dtlpy.entities.service.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*

deploy_pipeline(*service_json_path*: *Optional*[*str*] = *None*, *project*: *Optional*[*dtlpy.entities.project.Project*] = *None*, *bot*: *Optional*[*str*] = *None*, *force*: *bool* = *False*)

Deploy pipeline.

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

Parameters

- **service_json_path** (*str*) – path to service file
- **project** (*dtlpy.entities.project.Project*) – project entity
- **bot** (*str*) – user bot to run the service
- **force** (*bool*) – optional - force to deploy

Returns True if success

Return type *bool*

execute(*service*: *Optional*[*dtlpy.entities.service.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*) → *dtlpy.entities.execution.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*

get(*service_name*=*None*, *service_id*=*None*, *checkout*=*False*, *fetch*=*None*) → *dtlpy.entities.service.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*

list(*filters: Optional[dtlpy.entities.filters.Filters] = None*) → *dtlpy.entities.paged_entities.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*

log(*service, 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.

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*

name_validation(*name: str*)
Validation service name.

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

Parameters **name** (*str*) – service name

open_in_web(*service*: *Optional*[*dtlpy.entities.service.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*) –

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*

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*

revisions(*service*: *Optional*[*dtlpy.entities.service.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

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

tear_down(*service_json_path: Optional[str] = None, project: Optional[dtlpy.entities.project.Project] = None*)

Delete a pipeline.

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

Parameters

- **service_json_path** (*str*) – path to the service file
- **project** (`dtlpy.entities.project.Project`) – project entity

Returns True if success

Return type bool

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

2.9.1 Bots

class Bots(*client_api: dtlpy.services.api_client.ApiClient, project: dtlpy.entities.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*

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*

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*

list() → *dtlpy.miscellaneous.list_print.List[dtlpy.entities.bot.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*

2.10 Triggers

class Triggers(*client_api*: *dtlpy.services.api_client.ApiClient*, *project*: *Optional[dtlpy.entities.project.Project]* = None, *service*: *Optional[dtlpy.entities.service.Service]* = None, *project_id*: *Optional[str]* = None, *pipeline*: *Optional[dtlpy.entities.pipeline.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: dtlpy.entities.trigger.TriggerType =
    TriggerType.EVENT, name: Optional[str] = None, webhook_id=None, function_name='run',
    project_id=None, active=True, filters=None, resource: dtlpy.entities.trigger.TriggerResource =
    TriggerResource.ITEM, actions: Optional[dtlpy.entities.trigger.TriggerAction] = None,
    execution_mode: dtlpy.entities.trigger.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) →
    dtlpy.entities.trigger.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 root_node_namespace: namespace of pipeline root node to be triggered

Returns Trigger entity

Return type *dtlpy.entities.trigger.Trigger*

```
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*

```
get(trigger_id=None, trigger_name=None) → dtlpy.entities.trigger.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*

list(*filters*: *Optional*[*dtlpy.entities.filters.Filters*] = *None*) → *dtlpy.entities.paged_entities.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*

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.

update(*trigger*: *dtlpy.entities.trigger.BaseTrigger*) → *dtlpy.entities.trigger.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*

2.11 Executions

```
class Executions(client_api: dtlpy.services.api_client.ApiClient, service:  
                  Optional[dtlpy.entities.service.Service] = None, project:  
                  Optional[dtlpy.entities.project.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*[*dtlpy.entities.package_function.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*) → *dtlpy.entities.execution.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*

get(*execution_id*: *Optional*[*str*] = *None*, *sync*: *bool* = *False*) → *dtlpy.entities.execution.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*

increment(*execution*: *dtlpy.entities.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`

list(*filters*: *Optional*[`dtlpy.entities.filters.Filters`] = *None*) → `dtlpy.entities.paged_entities.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`

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

progress_update(*execution_id*: *str*, *status*: *Optional*[`dtlpy.entities.execution.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`

rerun(*execution*: `dtlpy.entities.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`

terminate(*execution*: `dtlpy.entities.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`

update(*execution*: `dtlpy.entities.execution.Execution`) → `dtlpy.entities.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`

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`

2.12 Pipelines

class Pipelines(*client_api*: `dtlpy.services.api_client.ApiClient`, *project*: `Optional[dtlpy.entities.project.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`) → `dtlpy.entities.pipeline.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`

delete(*pipeline*: `Optional[dtlpy.entities.pipeline.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`

execute(*pipeline*: `Optional[dtlpy.entities.pipeline.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`

get(*pipeline_name*=None, *pipeline_id*=None, *fetch*=None) → `dtlpy.entities.pipeline.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`

install(*pipeline*: `Optional[dtlpy.entities.pipeline.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

list(*filters: Optional[dtlpy.entities.filters.Filters] = None, project_id: Optional[str] = None*) → *dtlpy.entities.paged_entities.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*

open_in_web(*pipeline: Optional[dtlpy.entities.pipeline.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

pause(*pipeline: Optional[dtlpy.entities.pipeline.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

update(*pipeline: Optional[dtlpy.entities.pipeline.Pipeline] = None*) → *dtlpy.entities.pipeline.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*

2.12.1 Pipeline Executions

class PipelineExecutions(*client_api: dtlpy.services.api_client.ApiClient, project: Optional[dtlpy.entities.project.Project] = None, pipeline: Optional[dtlpy.entities.pipeline.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*

get(*pipeline_execution_id*: *str*, *pipeline_id*: *Optional[str]* = None) →

dtlpy.entities.pipeline_execution.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*

list(*filters*: *Optional[dtlpy.entities.filters.Filters]* = None) → *dtlpy.entities.paged_entities.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*

2.13 General Commands

class **Commands**(*client_api*: *dtlpy.services.api_client.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) → *dtlpy.entities.command.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=5*, *url=None*)

Wait for command to finish

Parameters

- **command_id** – Command id to wait to
- **timeout** – int, seconds to wait until TimeoutError is raised. if 0 - wait until done
- **step** – int, seconds between polling
- **url** – url to the command

Returns Command object

2.13.1 Download Commands

2.13.2 Upload Commands

ENTITIES

3.1 Organization

class `MemberOrgRole`(*value*)

Bases: `str`, `enum.Enum`

An enumeration.

class `Organization`(*members: list, groups: list, accounts: list, created_at, updated_at, id, name, logo_url, plan, owner, created_by, client_api: dtlpy.services.api_client.ApiClient, repositories=NOTHING*)

Bases: `dtlpy.entities.base_entity.BaseEntity`

Organization entity

add_member(*email, role: dtlpy.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`

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** – is Entity fetched from Platform
- **_json** – _json response from host
- **client_api** – ApiClient entity

Returns Project object

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[dtlpy.entities.organization.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: dtlpy.entities.organization.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.Enum`

An enumeration.

3.1.1 Integration

```
class Integration(id, name, type, org, created_at, created_by, update_at, client_api:
                  dtlpy.services.api_client.ApiClient, project=None)
```

Bases: `dtlpy.entities.base_entity.BaseEntity`

Integration object

```
delete(sure: bool = False, really: bool = False) → 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: dtlpy.services.api_client.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.Enum`

An enumeration.

```
class Project(contributors, created_at, creator, id, name, org, updated_at, role, account, is_blocked,
              feature_constraints, client_api: dtlpy.services.api_client.ApiClient, repositories=NOTHING)
```

Bases: `dtlpy.entities.base_entity.BaseEntity`

Project entity

add_member(*email*, *role*: `dtlpy.entities.project.MemberRole` = `MemberRole.DEVELOPER`)

Add a member to the project.

Parameters

- **email** (*str*) – member email
- **role** – “owner”, “engineer”, “annotator”, “annotationManager”

Returns dict that represent the user

Return type `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** – is Entity fetched from Platform
- **_json** – _json response from host
- **client_api** – ApiClient entity

Returns Project object

list_members(*role*: *Optional*[`dtlpy.entities.project.MemberRole`] = *None*)

List the project members.

Parameters **role** – “owner”, “engineer”, “annotator”, “annotationManager”

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: dtlpy.entities.project.MemberRole = MemberRole.DEVELOPER*)

Update member's information/details from the project.

Parameters

- **email** (*str*) – member email
- **role** – “owner”, “engineer”, “annotator”, “annotationManager”

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: `dtlpy.entities.base_entity.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, created_at, items_url, readable_type, access_level, driver, readonly,  
               client_api: dtlpy.services.api_client.ApiClient, instance_map=None, project=None,  
               datasets=None, repositories=NOTHING, ontology_ids=None, labels=None,  
               directory_tree=None)
```

Bases: `dtlpy.entities.base_entity.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

Parameters

- **label_name** – str - label name
- **color** – color
- **children** – children (sub labels)
- **attributes** – attributes
- **display_label** – display_label
- **label** – label
- **recipe_id** – optional recipe id
- **ontology_id** – optional ontology id
- **icon_path** – path to image to be display on label

Returns label entity

```
add_labels(label_list, ontology_id=None, recipe_id=None)
```

Add labels to dataset

Parameters

- **label_list** – label list
- **ontology_id** – optional ontology id
- **recipe_id** – optional recipe id

Returns label entities

```
checkout()
```

Checkout the dataset

```
clone(clone_name, filters=None, with_items_annotations=True, with_metadata=True,  
       with_task_annotations_status=True)
```

Clone dataset

Parameters

- **clone_name** – new dataset name
- **filters** (`dtlpy.entities.filters.Filters`) – Filters entity or a query dict
- **with_items_annotations** – clone all item's annotations
- **with_metadata** – clone metadata

- **with_task_annotations_status** – clone task annotations status

Returns dataset object

Return type *dtlpy.entities.dataset.Dataset*

delete(*sure=False, really=False*)

Delete a dataset forever!

Parameters

- **sure** (*bool*) – are you sure you want to delete?
- **really** (*bool*) – really really?

Returns True is success

Return type *bool*

delete_labels(*label_names*)

Delete labels from dataset's ontologies

Parameters **label_names** – label object/ label name / list of label objects / list of label names

Returns

download(*filters=None, local_path=None, file_types=None, annotation_options: Optional[dtlpy.entities.annotation.ViewAnnotationOptions] = None, annotation_filters=None, overwrite=False, to_items_folder=True, thickness=1, with_text=False, without_relative_path=None, alpha=None, 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

- **filters** (*dtlpy.entities.filters.Filters*) – Filters entity or a dictionary containing filters parameters
- **local_path** – local folder or filename to save to.
- **file_types** – a list of file type to download. e.g ['video/webm', 'video/mp4', 'image/jpeg', 'image/png']
- **annotation_options** – download annotations options: list(*dtlpy.entities.annotation.ViewAnnotationOptions*) not relevant for JSON option
- **annotation_filters** – Filters entity to filter annotations for download not relevant for JSON option
- **overwrite** – optional - default = False
- **to_items_folder** – Create 'items' folder and download items to it
- **thickness** – optional - line thickness, if -1 annotation will be filled, default = 1
- **with_text** – optional - add text to annotations, default = False
- **without_relative_path** – string - remote path - download items without the relative path from platform
- **alpha** – opacity value [0 1], default 1
- **export_version** (*str*) – exported items will have original extension in filename, *VI* - no original extension in filenames

Returns *List* of local_path per each downloaded item

download_annotations(*local_path=None, filters=None, annotation_options: Optional[dtlpy.entities.annotation.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=None, 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** – local folder or filename to save to.
- **filters** ([dtlpy.entities.filters.Filters](#)) – Filters entity or a dictionary containing filters parameters
- **annotation_options** – download annotations options: list([dl.ViewAnnotationOptions](#))
- **annotation_filters** – Filters entity to filter annotations for download
- **overwrite** – optional - default = False
- **thickness** – optional - line thickness, if -1 annotation will be filled, default =1
- **with_text** – optional - add text to annotations, default = False
- **remote_path** – DEPRECATED and ignored. use filters
- **include_annotations_in_output** – default - False , if export should contain annotations
- **export_png_files** – default - if True, semantic annotations should be exported as png files
- **filter_output_annotations** – default - False, given an export by filter - determine if to filter out annotations
- **alpha** – 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*

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](#) name of the partition
- **local_path** – 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: dtlpy.entities.project.Project, json: dict, client_api: dtlpy.services.api_client.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

open_in_web()

Open the dataset in web platform

static serialize_labels(*labels_dict*)

Convert hex color format to rgb

Parameters **labels_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

Parameters **state** (*bool*) – state

switch_recipe(*recipe_id=None*, *recipe=None*)

Switch the recipe that linked to the dataset with the given one

Parameters

- **recipe_id** – recipe id
- **recipe** – recipe entity

Returns**sync**(*wait=True*)

Sync dataset with external storage

Parameters **wait** – wait for the command to finish**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 dataset field

Parameters **system_metadata** (`bool`) – bool - True, if you want to change metadata system**Returns** Dataset object**Return type** `dtlpy.entities.dataset.Dataset`**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

Parameters

- **label_name** – label name
- **color** – color
- **children** – children (sub labels)
- **attributes** – attributes
- **display_label** – display label
- **label** – label
- **recipe_id** – optional recipe id
- **ontology_id** – optional ontology id
- **upsert** – if True will add in case it does not existing
- **icon_path** – path to image to be display on label

Returns label entity**update_labels**(*label_list, ontology_id=None, recipe_id=None, upsert=False*)

Add labels to dataset

Parameters

- **label_list** – label list
- **ontology_id** – optional ontology id
- **recipe_id** – optional recipe id
- **upsert** – if True will add in case it does not existing

Returns label entities

upload_annotations(*local_path*, *filters=None*, *clean=False*, *remote_root_path='/'*,
export_version=ExportVersion.V1)

Upload annotations to dataset.

Parameters

- **local_path** – str - local folder where the annotations files is.
- **filters** (`dtlpy.entities.filters.Filters`) – Filters entity or a dictionary containing filters parameters
- **clean** – bool - if True it 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

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

class ExpirationOptions(*item_max_days: Optional[int] = None*)

Bases: `object`

ExpirationOptions object

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: dtlpy.services.api_client.ApiClient*)

Bases: `dtlpy.entities.base_entity.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.Enum`

An enumeration.

3.4 Item

class `ExportMetadata(value)`

Bases: `enum.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: dtlpy.services.api_client.ApiClient, platform_dict, dataset, project, repositories=NOTHING)`

Bases: `dtlpy.entities.base_entity.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** – destination dataset id
- **remote_filepath** – complete filepath
- **metadata** – new metadata to add
- **with_annotations** – clone annotations
- **with_metadata** – clone metadata
- **with_task_annotations_status** – clone task annotations status
- **allow_many** – *bool* if True use multiple clones in single dataset is allowed, (default=False)
- **wait** – wait for the command to finish

Returns

Item

delete()

Delete item from platform

Returns

True

download(*local_path=None, file_types=None, save_locally=True, to_array=False, annotation_options: Optional[dtlpy.entities.annotation.ViewAnnotationOptions] = None, overwrite=False, to_items_folder=True, thickness=1, with_text=False, annotation_filters=None, alpha=None, export_version=ExportVersion.VI*)

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** – local folder or filename to save to disk or returns `BytelsIO`
- **file_types** – a list of file type to download. e.g ['video/webm', 'video/mp4', 'image/jpeg', 'image/png']
- **save_locally** – *bool*. save to disk or return a buffer
- **to_array** – returns `Ndarray` when True and `local_path = False`
- **annotation_options** – download annotations options: `list(dl.ViewAnnotationOptions)`
- **overwrite** – optional - default = False
- **to_items_folder** – Create 'items' folder and download items to it

- **thickness** – optional - line thickness, if -1 annotation will be filled, default =1
- **with_text** – optional - add text to annotations, default = False
- **annotation_filters** – Filters entity to filter annotations for download
- **alpha** – opacity value [0 1], default 1
- **export_version** (*str*) – exported items will have original extension in filename, *VI* - no original extension in filenames

Returns Output (list)

classmethod from_json(*_json*, *client_api*, *dataset=None*, *project=None*, *is_fetched=True*)

Build an item entity object from a json

Parameters

- **project** – project entity
- **_json** – _json response from host
- **dataset** – dataset in which the annotation's item is located
- **client_api** – ApiClient entity
- **is_fetched** – is Entity fetched from Platform

Returns Item object

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** – new full path to move item to.

Returns True if update successfully

open_in_web()

Open the items in web platform

Returns

set_description(*text: str*)

Update Item description

Parameters **text** – 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 - True, if you want to change metadata system

Returns Item object

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

class ItemStatus(*value*)
Bases: *str*, *enum.Enum*

An enumeration.

class ModalityRefTypeEnum(*value*)
Bases: *str*, *enum.Enum*

State enum

class ModalityTypeEnum(*value*)
Bases: *str*, *enum.Enum*

State enum

3.4.1 Item Link

class LinkTypeEnum(*value*)
Bases: *str*, *enum.Enum*

State enum

3.5 Annotation

class Annotation(*annotation_definition*:
dtlpy.entities.annotation_definitions.base_annotation_definition.BaseAnnotationDefinition, *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, *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: *dtlpy.entities.base_entity.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`

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

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

delete()

Remove an annotation from item

Returns True if success

Return type `bool`

download(*filepath*: `str`, *annotation_format*: `dtlpy.entities.annotation.ViewAnnotationOptions` = `ViewAnnotationOptions.MASK`, *height*: `Optional[float]` = None, *width*: `Optional[float]` = None, *thickness*: `int` = 1, *with_text*: `bool` = False, *alpha*: `Optional[float]` = None)

Save annotation to file

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`

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

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*

set_frame(*frame*)

Set annotation to frame state

Parameters **frame** (*int*) – frame number

Returns True if success

Return type *bool*

show(*image=None, thickness=None, with_text=False, height=None, width=None, annotation_format: dtlpy.entities.annotation.ViewAnnotationOptions = ViewAnnotationOptions.MASK, color=None, label_instance_dict=None, alpha=None*)

Show annotations mark the annotation of the image array and return it

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** – list(dl.ViewAnnotationOptions)
- **color** (*tuple*) – optional - color tuple
- **label_instance_dict** – the instance labels
- **alpha** (*float*) – opacity value [0 1], default 1

Returns list or single ndarray of the annotations

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.

Parameters **system_metadata** – True, if you want to change metadata system

Returns Annotation object

Return type *dtlpy.entities.annotation.Annotation*

update_status(*status: dtlpy.entities.annotation.AnnotationStatus = AnnotationStatus.ISSUE*)

Set status on annotation

Parameters **status** (*str*) – can be AnnotationStatus.ISSUE, AnnotationStatus.APPROVED, AnnotationStatus.REVIEW, AnnotationStatus.CLEAR

Returns Annotation object

Return type *dtlpy.entities.annotation.Annotation*

upload()

Create a new annotation in host

Returns Annotation entity

Return type *dtlpy.entities.annotation.Annotation*

class **AnnotationStatus**(*value*)

Bases: *str, enum.Enum*

An enumeration.

class **AnnotationType**(*value*)

Bases: *str, enum.Enum*

An enumeration.

class **ExportVersion**(*value*)

Bases: *str, enum.Enum*

An enumeration.

class **FrameAnnotation**(*annotation, annotation_definition, frame_num, fixed, object_visible, recipe_2_attributes=None, interpolation=False*)

Bases: *dtlpy.entities.base_entity.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.Enum`

An enumeration.

3.5.1 Collection of Annotation entities

class AnnotationCollection(*item=None, annotations=NOTHING, dataset=None, colors=None*)

Bases: `dtlpy.entities.base_entity.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

download(*filepath*, *img_filepath=None*, *annotation_format*: [dtlpy.entities.annotation.ViewAnnotationOptions](#) = [ViewAnnotationOptions.MASK](#), *height=None*, *width=None*, *thickness=1*, *with_text=False*, *orientation=0*, *alpha=None*)

Save annotations to file

Parameters

- **filepath** – path to save annotation
- **img_filepath** – img file path - needed for img_mask
- **annotation_format** – how to show thw annotations. options: list([dtlpy.entities.annotation.ViewAnnotationOptions](#))
- **height** – height
- **width** – width
- **thickness** – thickness
- **with_text** – add a text to the image
- **orientation** – the image orientation
- **alpha** – opacity value [0 1], default 1

Returns

from_instance_mask(*mask*, *instance_map=None*)
convert annotation from instance mask format :param mask: the mask annotation :param instance_map: labels

from_vtt_file(*filepath*)
convert annotation from vtt format :param filepath: path to the file

get_frame(*frame_num*)
Get frame

Parameters **frame_num** – 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*: [dtlpy.entities.annotation.ViewAnnotationOptions](#) = [ViewAnnotationOptions.MASK](#), *label_instance_dict=None*, *color=None*, *alpha=None*)
Show annotations according to annotation_format

Parameters

- **image** – empty or image to draw on
- **height** – height
- **width** – width
- **thickness** – line thickness
- **with_text** – add label to annotation
- **annotation_format** – how to show the annotations. options:
list(dl.ViewAnnotationOptions)
- **label_instance_dict** – instance label map {'Label': 1, 'More': 2}
- **color** – optional - color tuple
- **alpha** – opacity value [0 1], default 1

Returns ndarray of the annotations

to_json()

Convert annotation object to a platform json representation

Returns platform json

Return type dict

3.5.2 Annotation Definition

3.5.2.1 Box Annotation Definition

class Box(*left=None, top=None, right=None, bottom=None, label=None, attributes=None, description=None, angle=None*)

Bases: dtlpy.entities.annotation_definitions.base_annotation_definition.
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 “angle”

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

3.5.2.2 Classification Annotation Definition

class Classification(*label, attributes=None, description=None*)

Bases: `dtlpy.entities.annotation_definitions.base_annotation_definition.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

3.5.2.3 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: `dtlpy.entities.annotation_definitions.base_annotation_definition.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

3.5.2.4 Item Description Definition

class Description(*text, description=None*)

Bases: `dtlpy.entities.annotation_definitions.base_annotation_definition.BaseAnnotationDefinition`

Subtitle annotation object

3.5.2.5 Ellipse Annotation Definition

class Ellipse(*x, y, rx, ry, angle, label, attributes=None, description=None*)

Bases: `dtlpy.entities.annotation_definitions.base_annotation_definition.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

3.5.2.6 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: `dtlpy.entities.annotation_definitions.box.Box`

Note annotation object

3.5.2.7 Point Annotation Definition

```
class Point(x, y, label, attributes=None, description=None)
```

Bases: `dtlpy.entities.annotation_definitions.base_annotation_definition.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

3.5.2.8 Polygon Annotation Definition

```
class Polygon(geo, label, attributes=None, description=None)
```

Bases: `dtlpy.entities.annotation_definitions.base_annotation_definition.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

3.5.2.9 Polyline Annotation Definition

class Polyline(*geo, label, attributes=None, description=None*)

Bases: dtlpy.entities.annotation_definitions.base_annotation_definition.
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

3.5.2.10 Pose Annotation Definition

class Pose(*label, template_id, instance_id=None, attributes=None, points=None, description=None*)

Bases: dtlpy.entities.annotation_definitions.base_annotation_definition.
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

3.5.2.11 Segmentation Annotation Definition

class Segmentation(*geo, label, attributes=None, description=None*)

Bases: dtlpy.entities.annotation_definitions.base_annotation_definition.
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

3.5.2.12 Audio Annotation Definition

```
class Subtitle(text, label, attributes=None, description=None)
    Bases: dtlpy.entities.annotation_definitions.base_annotation_definition.
            BaseAnnotationDefinition
    Subtitle annotation object
```

3.5.2.13 Undefined Annotation Definition

```
class UndefinedAnnotationType(type, label, coordinates, attributes=None, description=None)
    Bases: dtlpy.entities.annotation_definitions.base_annotation_definition.
            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: dtlpy.entities.similarity.CollectionTypes, name, items=None)
    Bases: object
    Base Collection Entity

    add(ref, type: dtlpy.entities.similarity.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: dtlpy.entities.similarity.SimilarityTypeEnum, ref)
    Bases: object
    Base CollectionItem
```

```

class CollectionTypes(value)
    Bases: str, enum.Enum

    An enumeration.

class MultiView(name, items=None)
    Bases: dtlpy.entities.similarity.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: dtlpy.entities.similarity.CollectionItem

    Single multi view item

class Similarity(ref, name=None, items=None)
    Bases: dtlpy.entities.similarity.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: dtlpy.entities.similarity.CollectionItem

    Single similarity item

class SimilarityTypeEnum(value)
    Bases: str, enum.Enum

    State enum

```

3.6 Filter

```

class Filters(field=None, values=None, operator: Optional[dtlpy.entities.filters.FiltersOperations] = None,
              method: Optional[dtlpy.entities.filters.FiltersMethod] = None, custom_filter=None, resource:
              dtlpy.entities.filters.FiltersResource = FiltersResource.ITEM, use_defaults=True, context=None)
    Bases: object

    Filters entity to filter items from pages in platform

```

add(*field*, *values*, *operator*: *Optional*[dtlpy.entities.filters.FiltersOperations] = None, *method*: *Optional*[dtlpy.entities.filters.FiltersMethod] = None)
Add filter

Parameters

- **field** – Metadata field / attribute
- **values** – field values
- **operator** – optional - in, gt, lt, eq, ne
- **method** – Optional - or/and

Returns

add_join(*field*, *values*, *operator*: *Optional*[dtlpy.entities.filters.FiltersOperations] = None, *method*: *dtlpy.entities.filters.FiltersMethod* = *FiltersMethod.AND*)
join a query to the filter

Parameters

- **field** – field to add
- **values** – values
- **operator** – optional - in, gt, lt, eq, ne
- **method** – optional - str - *FiltersMethod.AND*, *FiltersMethod.OR*

generate_url_query_params(*url*)
generate url query params

Parameters **url** –

has_field(*field*)
is filter has field

Parameters **field** – field to check

Returns Ture is have it

Return type *bool*

pop(*field*)
Pop filed

Parameters **field** – field to pop

pop_join(*field*)
Pop join

Parameters **field** – field to pop

prepare(*operation*=None, *update*=None, *query_only*=False, *system_update*=None, *system_metadata*=False)
To dictionary for platform call

Parameters

- **operation** – operation
- **update** – update
- **query_only** – 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*: dtlpy.entities.filters.FiltersOrderByDirection = *FiltersOrderByDirection.ASCENDING*)
sort the filter

Parameters

- **field** – field to sort by it
- **value** – FiltersOrderByDirection.ASCENDING, FiltersOrderByDirection.DECENDING

class FiltersKnownFields(*value*)

Bases: str, enum.Enum

An enumeration.

class FiltersMethod(*value*)

Bases: str, enum.Enum

An enumeration.

class FiltersOperations(*value*)

Bases: str, enum.Enum

An enumeration.

class FiltersOrderByDirection(*value*)

Bases: str, enum.Enum

An enumeration.

class FiltersResource(*value*)

Bases: str, enum.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*: dtlpy.services.api_client.ApiClient, *dataset=None*, *project=None*, *repositories=NOTHING*)

Bases: dtlpy.entities.base_entity.BaseEntity

Recipe object

clone(*shallow=False*)

Clone Recipe

Parameters **shallow** – If True, link ot existing ontology, clones all ontology that are link to the recipe as well

Returns Cloned ontology object

delete(*force*: *bool = False*)

Delete recipe from platform

Parameters **force** (*bool*) – force delete recipe

Returns True

classmethod `from_json(_json, client_api, dataset=None, project=None, is_fetched=True)`

Build a Recipe entity object from a json

Parameters

- **_json** – _json response from host
- **dataset** – recipe’s dataset
- **project** – recipe’s project
- **client_api** – ApiClient entity
- **is_fetched** – is Entity fetched from Platform

Returns Recipe object

get_annotation_template_id(template_name)

Get annotation template id by template name

Parameters **template_name** –

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 - True, if you want to change metadata system

Returns Recipe object

3.7.1 Ontology

class `Ontology(client_api: dtlpy.services.api_client.ApiClient, id, creator, url, title, labels, metadata, attributes, recipe=None, dataset=None, project=None, repositories=NOTHING, instance_map=None)`

Bases: `dtlpy.entities.base_entity.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** – label name
- **color** – optional - if not given a random color will be selected

- **children** – optional - children
- **attributes** – optional - attributes
- **display_label** – optional - display_label
- **label** – label
- **add** – to add or not
- **icon_path** – path to image to be display on label
- **update_ontology** – update the ontology, default = False for backward compatible

Returns Label entity

add_labels(*label_list*, *update_ontology=False*)

Adds a list of labels to ontology

Parameters

- **label_list** – list of labels [{"value": {"tag": "tag", "displayLabel": "displayLabel", "color": "#color", "attributes": [attributes]}, "children": [children]}]
- **update_ontology** – update the ontology, default = False for backward compatible

Returns List of label entities added

delete()

Delete recipe from platform

Returns True

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** – is Entity fetched from Platform
- **project** – project entity
- **dataset** – dataset entity
- **_json** – _json response from host
- **recipe** – ontology's recipe
- **client_api** – ApiClient entity

Returns Ontology object

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 - True, if you want to change metadata system

Returns Ontology object

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** – label name
- **color** – optional - if not given a random color will be selected
- **children** – optional - children
- **attributes** – optional - attributes
- **display_label** – optional - display_label
- **label** – label
- **add** – to add or not
- **icon_path** – path to image to be display on label
- **upsert** – if True will add in case it does not existing
- **update_ontology** – update the ontology, default = False for backward compatible

Returns Label entity

update_labels(*label_list, upsert=False, update_ontology=False*)

Update a list of labels to ontology

Parameters

- **label_list** – list of labels [{"value": {"tag": "tag", "displayLabel": "displayLabel", "color": "#color", "attributes": [attributes]}, "children": [children]}]
- **upsert** – if True will add in case it does not existing
- **update_ontology** – update the ontology, default = False for backward compatible

Returns List of label entities added

3.7.1.1 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=0, wait=True*)

Add items to Task

Parameters

- **filters** (`dtlpy.entities.filters.Filters`) – Filters entity or a dictionary containing filters parameters
- **items** – items list for the assignment
- **assignee_ids** – list of assignee for the assignment
- **workload** – the load of work
- **limit** – limit
- **wait** – wait for the command to finish

Returns

create_assignment(*assignment_name*, *assignee_id*, *items=None*, *filters=None*)

Create a new assignment

Parameters

- **assignment_name** – assignment name
- **assignee_id** – list of assignee for the assignment
- **items** – items list for the assignment
- **filters** (`dtlpy.entities.filters.Filters`) – Filters entity or a dictionary containing filters parameters

Returns

create_qa_task(*due_date*, *assignee_ids*, *filters=None*, *items=None*, *query=None*, *workload=None*, *metadata=None*, *available_actions=None*, *wait=True*)

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

Returns task object

Return type `dtlpy.entities.task.Task`

delete(*wait=True*)

Delete task from platform

Parameters **wait** – wait for the command to finish

Returns True

get_items(*filters=None*)

Get the task items

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

Returns

open_in_web()

Open the task in web platform

Returns

set_status(*status: str, operation: str, item_ids: List[str]*)

Update item status within task

Parameters

- **status** – str - string the describes the status
- **operation** – str - 'create' or 'delete'
- **item_ids** – List[str]

:return : Boolean

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** – 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: `dtlpy.entities.base_entity.BaseEntity`

Assignment object

get_items(*dataset=None, filters=None*)

Get all the items in the assignment

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`

open_in_web()

Open the assignment in web platform

Returns

reassign(*assignee_id, wait=True*)

Reassign an assignment

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***redistribute**(*workload*, *wait=True*)

Redistribute an assignment

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**set_status**(*status: str*, *operation: str*, *item_id: str*)

Set item status within assignment

Parameters

- **status** (*str*) – status
- **operation** (*str*) – created/deleted
- **item_id** (*str*) – item id

Returns True id 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 assignment

Parameters **system_metadata** (*bool*) – True, if you want to change metadata system**Returns** Assignment object**Return type** *dtlpy.entities.assignment.Assignment* assignment**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: dtlpy.services.api_client.ApiClient,
              revisions=None, repositories=NOTHING, artifacts=None, codebases=None, requirements=None)
```

Bases: `dtlpy.entities.base_entity.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, **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

Returns Service object

classmethod **from_json**(*_json, client_api, project, is_fetched=True*)

Turn platform representation of package into a package entity

Parameters

- **_json** – platform representation of package
- **client_api** – ApiClient entity
- **project** – project entity
- **is_fetched** – is Entity fetched from Platform

Returns Package entity

open_in_web()

Open the package in web platform

Returns

pull(*version=None, local_path=None*)

Push local package

Parameters

- **version** – version
- **local_path** – local path

Returns

push(*codebase: Optional[Union[dtlpy.entities.codebase.GitCodebase, dtlpy.entities.codebase.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** – save package to local checkout
- **src_path** – location of package codebase folder to zip
- **package_name** – name of package
- **modules** – list of PackageModule
- **revision_increment** – optional - str - version bumping method - major/minor/patch - default = None
- **service_update** – optional - bool - update the service
- **service_config** – optional - json of service - a service that have config from the main service if wanted

Returns

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.Enum

An enumeration.

3.9.1 Package Function

class PackageFunction(outputs=NOTHING, name=NOTHING, description="", inputs=NOTHING, display_name=None, display_icon=None)

Bases: dtlpy.entities.base_entity.BaseEntity

Webhook object

class PackageInputType(value)

Bases: str, enum.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: dtlpy.entities.base_entity.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: dtlpy.entities.package_slot.SlotPostAction = NOTHING, default_inputs: Optional[list] = None, input_options: Optional[list] = None)

Bases: dtlpy.entities.base_entity.BaseEntity

Webhook object

class SlotDisplayScopeResource(value)

Bases: str, enum.Enum

An enumeration.


```
class SlotPostActionType(value)
```

Bases: `str`, `enum.Enum`

An enumeration.

```
class UiBindingPanel(value)
```

Bases: `str`, `enum.Enum`

An enumeration.

3.9.4 Codebase

3.10 Service

```
class InstanceCatalog(value)
```

Bases: `str`, `enum.Enum`

An enumeration.

```
class KubernetesAutoscalerType(value)
```

Bases: `str`, `enum.Enum`

An enumeration.

```
class OnResetAction(value)
```

Bases: `str`, `enum.Enum`

An enumeration.

```
class RuntimeType(value)
```

Bases: `str`, `enum.Enum`

An enumeration.

```
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:
               dtlpy.entities.service.OnResetAction, project_id, is_global, max_attempts, package, client_api:
               dtlpy.services.api_client.ApiClient, revisions=None, project=None, repositories=NOTHING)
```

Bases: `dtlpy.entities.base_entity.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*) – prevent override
- **visible** (*bool*) – visible
- **icon** (*str*) – icon
- **kwargs** –

Returns List of user setting for activated slots

checkout()

Checkout

Returns

delete()

Delete Service object

Returns True

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** – input dictionary or list of FunctionIO entities
- **function_name** – str - function name to run

:param resource:dl.PackageInputType - input type. :param item_id:str - optional - input to function :param dataset_id:str - optional - input to function :param annotation_id:str - optional - input to function :param project_id:str - resource's project :param sync: bool - wait for function to end :param stream_logs: bool - prints logs of the new execution. only works with sync=True :param return_output: bool - if True and sync is True - will return the output directly :return:

classmethod from_json(*_json: dict, client_api: dtlpy.services.api_client.ApiClient, package=None, project=None, is_fetched=True*)

Build a service entity object from a json

Parameters

- **_json** – platform json
- **client_api** – ApiClient entity
- **package** – package entity
- **project** – project entity
- **is_fetched** – is Entity fetched from Platform

Returns

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** –

- **start** – iso format time
- **end** – iso format time
- **follow** – keep stream future logs
- **text** – text
- **execution_id** (*str*) – execution id
- **function_name** (*str*) – function name
- **replica_id** (*str*) – replica id
- **system** – system
- **view** – view
- **until_completed** (*bool*) – wait until completed

Returns ServiceLog entity

open_in_web()

Open the service in web platform

Returns

pause()

Returns

resume()

Returns

status()

Get Service status

Returns True

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** – force update

Returns Service entity

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: *dtlpy.entities.user.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: dtlpy.services.api_client.ApiClient, op_type='service', repositories=NOTHING*)

Bases: *dtlpy.entities.base_entity.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** – 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`

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: dtlpy.services.api_client.ApiClient, op_type='service',
                  repositories=NOTHING, start_at=None, end_at=None, cron=None)
```

Bases: `dtlpy.entities.trigger.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:
              dtlpy.services.api_client.ApiClient, op_type='service', repositories=NOTHING, filters=None,
              execution_mode=TriggerExecutionMode.ONCE, actions=TriggerAction.CREATED,
              resource=TriggerResource.ITEM)
```

Bases: `dtlpy.entities.trigger.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** – project entity
- **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.Enum

An enumeration.

class TriggerExecutionMode(value)

Bases: str, enum.Enum

An enumeration.

class TriggerResource(value)

Bases: str, enum.Enum

An enumeration.

class TriggerType(value)

Bases: str, enum.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: dtlpy.services.api_client.ApiClient, service, project=None, repositories=NOTHING, pipeline: Optional[dict] = None)

Bases: dtlpy.entities.base_entity.BaseEntity

Service execution entity

classmethod from_json(_json, client_api, project=None, service=None, is_fetched=True)

Parameters

- **_json** – platform json
- **client_api** – ApiClient entity
- **project** – project entity
- **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[dtlpy.entities.execution.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.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, info, project, client_api: dtlpy.services.api_client.ApiClient, repositories=NOTHING*)

Bases: `dtlpy.entities.base_entity.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** – platform representation of package
- **client_api** – ApiClient entity
- **project** – project entity
- **is_fetched** – is Entity fetched from Platform

Returns Package entity

install()

install pipeline

Returns Composition entity

open_in_web()

Open the pipeline in web platform

Returns

pause()

pause pipeline

Returns Composition entity

set_start_node(*node: dtlpy.entities.node.PipelineNode*)

Set the start node of the pipeline

Parameters **node** (*PipelineNode*) – node to be the start node

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, created_at, updated_at, pipeline_id, pipeline_execution_id, pipeline, client_api: dtlpy.services.api_client.ApiClient, repositories=NOTHING*)

Bases: `dtlpy.entities.base_entity.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** – platform representation of package
- **client_api** – ApiClient entity
- **pipeline** – Pipeline entity
- **is_fetched** – is Entity fetched from Platform

Returns Package entity

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: dtlpy.services.api_client.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:
               dtlpy.services.api_client.ApiClient, repositories=NOTHING)
```

Bases: `dtlpy.entities.base_entity.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=5*)

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

Returns Command object

class **CommandsStatus**(*value*)

Bases: `str`, `enum.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: dtlpy.utilities.base_package_runner.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*: `dtlpy.utilities.converter.AnnotationFormat`, *from_format*: `dtlpy.utilities.converter.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*: `dtlpy.utilities.converter.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 Connect Cloud Storage

Setup integration with GCS/S3/Azure

5.1.1.1 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.

5.1.1.1.1 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.

5.1.1.1.2 Create Integration With GCS

5.1.1.1.2.1 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,
```

(continues on next page)

(continued from previous page)

```
options={'key': '',
        'secret': '',
        'content': gcs_to_string})
```

5.1.1.1.2.2 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"})
```

5.1.1.1.2.3 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'})
```

5.1.1.1.3 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.

5.1.1.1.4 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
driver = dl.drivers.create(name='driver_name', driver_type=dl.ExternalStorage.S3,↳
↳ integration_id='integration_id',
```

(continues on next page)

(continued from previous page)

```
bucket_name='bucket_name', project_id='project_id',
allow_external_delete=True,
region='eu-west-1', storage_class="", path="")
```

5.1.2 Manage Datasets

Create and manage Datasets and connect them with your cloud storage

5.1.2.1 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).

5.1.2.1.1 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_and_shlomi(dataset_name='my-dataset-name')
```

5.1.2.1.2 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
↪")
```

5.1.2.1.3 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')
```

5.1.2.1.4 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")
```

5.1.2.1.5 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:]
        # 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))
```

(continues on next page)

(continued from previous page)

```

if copy_annotations:
    new_item.annotations.upload(item.annotations.list())

```

5.1.3 Data Versioning

How to manage versions

5.1.3.1 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.

5.1.3.1.1 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)

```

5.1.3.1.2 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

5.1.4.1 Upload & Manage Data & Metadata

5.1.4.1.1 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
```

5.1.4.1.2 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
```

5.1.4.1.3 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()
```


5.1.4.1.3.1 Additional upload options

Additional upload options include using buffer, pillow, openCV, and NdArray - see our complete documentation for code examples.

5.1.4.1.4 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

5.1.5.1 Upload & Manage Annotations

```
item = dl.items.get(item_id="")
annotation = item.annotations.get(annotation_id="")
annotation.metadata["user"] = True
annotation.update()
```

5.1.5.1.1 Convert Annotations To COCO Format

```
import dtlpy as dl
dataset = project.datasets.get(dataset_name='dataset_name')
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'
)
```

5.1.5.1.2 Upload Entire Directory and their Corresponding Dataloop JSON Annotations

```
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')
# 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)
```

5.1.5.1.3 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 dtlpy as dl
import pandas as pd
project = dl.projects.get(project_name='my_project')
dataset = project.datasets.get(dataset_id='my_dataset')
# 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'],
```

(continues on next page)

(continued from previous page)

```

                                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)

```

5.1.5.2 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='')

```

5.1.5.3 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.

5.1.5.3.1 Download Items and Annotations

Download dataset items and annotations to your computer folder in two separate folders. See all annotation options [here](#).

```

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.download(local_path=r'C:/home/project/images', # The default value is ".dataloop
↪ folder
                    annotation_options=dl.VIEW_ANNOTATION_OPTIONS_JSON)

```

5.1.5.3.2 Multiple Annotation Options

See all annotation options [here](#).

```
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.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])
```

5.1.5.3.3 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'.

```
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')
# 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')
dataset.download( # The default value is ".dataloop" folder
                  local_path=r'C:/home/project/images',
                  filters = item_filters,
                  annotation_filters=annotation_filters,
                  annotation_options=dl.VIEW_ANNOTATION_OPTIONS_JSON)
```

5.1.5.3.4 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](#).

```
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(item_id="item_id") #Get item from dataset to be able to view
↳ the dataset colors on Mask
# Filter items with dog annotations
```

(continues on next page)

(continued from previous page)

```

annotation_filters = dl.Filters(resource='annotations', field='label', values='dog')
item.download( # the default value is ".dataloop" folder
               local_path=r'C:/home/project/images',
               annotation_filters=annotation_filters,
               annotation_options=dl.VIEW_ANNOTATION_OPTIONS_JSON)

```

5.1.5.3.5 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'.

```

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')
# 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)

```

5.2 FaaS Tutorial

Tutorials for FaaS

5.2.1 FaaS Interactive Tutorial – Using Python & Dataloop SDK

FaaS Interactive Tutorial

5.2.1.1 FaaS Interactive Tutorial – Using Python & Dataloop SDK

5.2.1.1.1 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.

5.2.1.2 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.

5.2.2.1 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_name')
```

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/
↪train2017/0000000184321.jpg'],
    remote_path='/folder_name')
```

5.2.3 Run Your First Function

Create and run your first FaaS in the Dataloop platform

5.2.3.1 Basic Use Case: Single Function

5.2.3.1.1 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.filename)

    # 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:

```
service = project.services.deploy(func=rgb2gray,
                                   service_name='grayscale-item-service')
```

5.2.3.1.2 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 Function

Create a Package with multiple functions and modules

5.2.4.1 Advanced Use Case: Multiple Functions

5.2.4.1.1 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. To 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')
```

5.2.4.1.2 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
```

(continues on next page)

(continued from previous page)

```

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)

    # 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)

```

5.2.4.1.3 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',
                                                  inputs=[dl.FunctionIO(type=dl.
→PackageInputType.ITEM,
                                                  name=
→'item'))])
]]]
```

5.2.4.1.4 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_name)
package = project.packages.push(package_name='image-processing',
                                modules=modules,
                                src_path=src_path)
```

5.2.4.1.5 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')
```

5.2.4.1.6 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`.

5.2.4.1.7 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](#).

5.2.4.1.8 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()
```

5.2.4.1.9 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!

5.3 Model Management

Tutorials for creating and managing model and snapshots

5.3.1 Introduction

Getting started with Model.

5.3.1.1 Model Management

5.3.1.1.1 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

TODO: add more documentation in the Adapter function and maybe some example

5.3.1.1.1.1 Model and Snapshot entities

5.3.1.1.1.2 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

TODO: add the module attributes

5.3.1.1.1.3 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)

5.3.1.1.1.4 Buckets and Codebase

1. local
2. item
3. git
4. GCS

5.3.1.1.1.5 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

5.3.2 Create a Model and Snapshot

Create a Model with a Dataloop Model Adapter

5.3.2.1 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):
    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')
```

(continues on next page)

(continued from previous page)

```

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()

```

5.3.3 Using Dataloop's Dataset Generator

Use the SDK and the Dataset Tools to iterate, augment and serve the data to your model

5.3.3.1 Dataloop Dataloader

A dl.Dataset image and annotation generator for training and for items visualization

We can visualize the data with augmentation for debug 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')
dataloader = DatasetGenerator(data_path='train',
                              dataset_entity=dataset,
                              annotation_type=dl.AnnotationType.BOX)

```

5.3.3.1.1 Object Detection Examples

We can visualize a random item from the dataset:

```
for i in range(5):
    dataloader.visualize()
```

Or get the same item using it's index:

```
for i in range(5):
    dataloader.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()
]
dataloader = DatasetGenerator(data_path='train',
                             dataset_entity=dataset,
                             annotation_type=dl.AnnotationType.BOX,
                             transforms=tfs)

dataloader.visualize()
dataloader.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(dataloader[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
dataloader = 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 = dataloader[np.random.randint(len(dataloader))]
    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')
```

5.3.3.1.2 Segmentation Examples

First we'll load a semantic dataset and view some images and the output structure

```
dataset = dl.datasets.get(dataset_id='6197985a104eb81cb728e4ac')
dataloader = DatasetGenerator(data_path='semantic',
                              dataset_entity=dataset,
                              transforms=tfs,
                              return_originals=True,
                              annotation_type=dl.AnnotationType.SEGMENTATION)

for i in range(5):
    dataloader.visualize()
```

Visualize original vs augmented image and annotations mask:

```
fig, ax = plt.subplots(2, 4)
for i in range(2):
    item_element = dataloader[np.random.randint(len(dataloader))]
    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 to visualize the binary mask per label. We will plot only 8 label (there might be more on the item):

```
item_element = dataloader[np.random.randint(len(dataloader))]
annotations = item_element['annotations']
unique_labels = np.unique(annotations)
one_hot_annotations = np.arange(len(dataloader.id_to_label_map)) == annotations[...],
↪ None]
print('unique label indices in the item: {}'.format(unique_labels))
print('unique labels in the item: {}'.format([dataloader.id_to_label_map[i] for i in
↪ unique_labels]))
```

(continues on next page)

(continued from previous page)

```
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(dataloader.id_to_label_map[label_ind])
```

INDICES AND TABLES

- `genindex`
- `modindex`

PYTHON MODULE INDEX

d

- `dtlpy.entities.annotation`, 90
- `dtlpy.entities.annotation_collection`, 94
- `dtlpy.entities.annotation_definitions.base_annotation_definition`, 96
- `dtlpy.entities.annotation_definitions.box`, 96
- `dtlpy.entities.annotation_definitions.classification`, 97
- `dtlpy.entities.annotation_definitions.cube`, 97
- `dtlpy.entities.annotation_definitions.description`, 97
- `dtlpy.entities.annotation_definitions.ellipse`, 97
- `dtlpy.entities.annotation_definitions.note`, 98
- `dtlpy.entities.annotation_definitions.point`, 98
- `dtlpy.entities.annotation_definitions.polygon`, 98
- `dtlpy.entities.annotation_definitions.polyline`, 99
- `dtlpy.entities.annotation_definitions.pose`, 99
- `dtlpy.entities.annotation_definitions.segmentation`, 99
- `dtlpy.entities.annotation_definitions.subtitle`, 100
- `dtlpy.entities.annotation_definitions.undefined_annotation`, 100
- `dtlpy.entities.assignment`, 108
- `dtlpy.entities.base_entity`, 122
- `dtlpy.entities.bot`, 116
- `dtlpy.entities.codebase`, 113
- `dtlpy.entities.command`, 122
- `dtlpy.entities.dataset`, 82
- `dtlpy.entities.directory_tree`, 123
- `dtlpy.entities.driver`, 87
- `dtlpy.entities.execution`, 118
- `dtlpy.entities.filters`, 101
- `dtlpy.entities.integration`, 79
- `dtlpy.entities.item`, 88
- `dtlpy.entities.label`, 106
- `dtlpy.entities.links`, 90
- `dtlpy.entities.ontology`, 104
- `dtlpy.entities.organization`, 77
- `dtlpy.entities.package`, 110
- `dtlpy.entities.package_function`, 112
- `dtlpy.entities.package_module`, 112
- `dtlpy.entities.package_slot`, 112
- `dtlpy.entities.paged_entities`, 121
- `dtlpy.entities.pipeline`, 119
- `dtlpy.entities.pipeline_execution`, 121
- `dtlpy.entities.project`, 79
- `dtlpy.entities.recipe`, 103
- `dtlpy.entities.service`, 113
- `dtlpy.entities.similarity`, 100
- `dtlpy.entities.task`, 106
- `dtlpy.entities.trigger`, 116
- `dtlpy.entities.user`, 81
- `dtlpy.repositories.annotations`, 37
- `dtlpy.repositories.assignments`, 48
- `dtlpy.repositories.bots`, 65
- `dtlpy.repositories.codebases`, 57
- `dtlpy.repositories.commands`, 74
- `dtlpy.repositories.datasets`, 26
- `dtlpy.repositories.downloader`, 75
- `dtlpy.repositories.drivers`, 31
- `dtlpy.repositories.executions`, 68
- `dtlpy.repositories.integrations`, 22
- `dtlpy.repositories.items`, 32
- `dtlpy.repositories.ontologies`, 42
- `dtlpy.repositories.organizations`, 19
- `dtlpy.repositories.packages`, 51
- `dtlpy.repositories.pipeline_executions`, 73
- `dtlpy.repositories.pipelines`, 71
- `dtlpy.repositories.projects`, 23
- `dtlpy.repositories.recipes`, 40
- `dtlpy.repositories.services`, 59
- `dtlpy.repositories.tasks`, 43
- `dtlpy.repositories.triggers`, 66
- `dtlpy.repositories.uploader`, 75
- `dtlpy.utilities.converter`, 125

A

abort() (Command method), 122
 abort() (Commands method), 74
 activate_slots() (Service method), 113
 activate_slots() (Services method), 59
 add() (AnnotationCollection method), 94
 add() (Collection method), 100
 add() (Filters method), 101
 add() (Workload method), 109
 add_frame() (Annotation method), 90
 add_frames() (Annotation method), 91
 add_function() (PackageModule method), 112
 add_items() (Task method), 106
 add_items() (Tasks method), 43
 add_join() (Filters method), 102
 add_label() (Dataset method), 82
 add_label() (Ontology method), 104
 add_labels() (Dataset method), 82
 add_labels() (Ontology method), 105
 add_member() (Organization method), 77
 add_member() (Organizations method), 19
 add_member() (Project method), 80
 add_member() (Projects method), 23
 Annotation (class in *dtlpy.entities.annotation*), 90
 AnnotationCollection (class in *dtlpy.entities.annotation_collection*), 94
 Annotations (class in *dtlpy.repositories.annotations*), 37
 AnnotationStatus (class in *dtlpy.entities.annotation*), 93
 AnnotationType (class in *dtlpy.entities.annotation*), 93
 Assignment (class in *dtlpy.entities.assignment*), 108
 Assignments (class in *dtlpy.repositories.assignments*), 48
 attach_agent_progress() (Converter method), 125

B

BaseTrigger (class in *dtlpy.entities.trigger*), 116
 Bot (class in *dtlpy.entities.bot*), 116
 Bots (class in *dtlpy.repositories.bots*), 65
 Box (class in *dtlpy.entities.annotation_definitions.box*), 96

build_requirements() (Packages method), 51
 build_trigger_dict() (Packages static method), 52
 builder() (Annotations method), 37

C

check_cls_arguments() (Packages static method), 52
 checkout() (Dataset method), 82
 checkout() (Datasets method), 26
 checkout() (Package method), 110
 checkout() (Packages method), 52
 checkout() (Project method), 80
 checkout() (Projects method), 24
 checkout() (Service method), 114
 checkout() (Services method), 60
 Classification (class in *dtlpy.entities.annotation_definitions.classification*), 97
 clone() (Dataset method), 82
 clone() (Datasets method), 27
 clone() (Item method), 88
 clone() (Items method), 32
 clone() (Recipe method), 103
 clone() (Recipes method), 40
 clone_git() (Codebases method), 57
 Codebases (class in *dtlpy.repositories.codebases*), 57
 Collection (class in *dtlpy.entities.similarity*), 100
 CollectionItem (class in *dtlpy.entities.similarity*), 100
 CollectionTypes (class in *dtlpy.entities.similarity*), 100
 Command (class in *dtlpy.entities.command*), 122
 Commands (class in *dtlpy.repositories.commands*), 74
 CommandsStatus (class in *dtlpy.entities.command*), 123
 convert() (Converter method), 125
 convert_dataset() (Converter method), 125
 convert_directory() (Converter method), 126
 convert_file() (Converter method), 126
 Converter (class in *dtlpy.utilities.converter*), 125
 create() (Assignments method), 48
 create() (Bots method), 65
 create() (Datasets method), 27
 create() (Drivers method), 31
 create() (Executions method), 68
 create() (Integrations method), 22

[create\(\)](#) (*Ontologies method*), 42
[create\(\)](#) (*PipelineExecutions method*), 73
[create\(\)](#) (*Pipelines method*), 71
[create\(\)](#) (*Projects method*), 24
[create\(\)](#) (*Recipes method*), 40
[create\(\)](#) (*Tasks method*), 44
[create\(\)](#) (*Triggers method*), 66
[create_assignment\(\)](#) (*Task method*), 107
[create_qa_task\(\)](#) (*Task method*), 107
[create_qa_task\(\)](#) (*Tasks method*), 44
[CronTrigger](#) (*class in dtlpy.entities.trigger*), 117
[Cube](#) (*class in dtlpy.entities.annotation_definitions.cube*), 97
[custom_format\(\)](#) (*Converter static method*), 126

D

[Dataset](#) (*class in dtlpy.entities.dataset*), 82
[Datasets](#) (*class in dtlpy.repositories.datasets*), 26
[delete\(\)](#) (*Annotation method*), 91
[delete\(\)](#) (*Annotations method*), 37
[delete\(\)](#) (*BaseTrigger method*), 116
[delete\(\)](#) (*Bots method*), 116
[delete\(\)](#) (*Bots method*), 65
[delete\(\)](#) (*Dataset method*), 83
[delete\(\)](#) (*Datasets method*), 27
[delete\(\)](#) (*Integration method*), 79
[delete\(\)](#) (*Integrations method*), 22
[delete\(\)](#) (*Item method*), 88
[delete\(\)](#) (*Items method*), 33
[delete\(\)](#) (*Ontologies method*), 42
[delete\(\)](#) (*Ontology method*), 105
[delete\(\)](#) (*Package method*), 110
[delete\(\)](#) (*Packages method*), 52
[delete\(\)](#) (*Pipeline method*), 119
[delete\(\)](#) (*Pipelines method*), 72
[delete\(\)](#) (*Project method*), 80
[delete\(\)](#) (*Projects method*), 24
[delete\(\)](#) (*Recipe method*), 103
[delete\(\)](#) (*Recipes method*), 41
[delete\(\)](#) (*Service method*), 114
[delete\(\)](#) (*Services method*), 60
[delete\(\)](#) (*Task method*), 107
[delete\(\)](#) (*Tasks method*), 45
[delete\(\)](#) (*Triggers method*), 67
[delete_labels\(\)](#) (*Dataset method*), 83
[delete_labels\(\)](#) (*Ontology method*), 105
[delete_member\(\)](#) (*Organization method*), 77
[delete_member\(\)](#) (*Organizations method*), 19
[deploy\(\)](#) (*Package method*), 110
[deploy\(\)](#) (*Packages method*), 53
[deploy\(\)](#) (*Services method*), 60
[deploy_from_file\(\)](#) (*Packages method*), 54
[deploy_from_local_folder\(\)](#) (*Services method*), 61
[deploy_pipeline\(\)](#) (*Services method*), 61

[Description](#) (*class in dtlpy.entities.annotation_definitions.description*), 97
[directory_tree\(\)](#) (*Datasets method*), 28
[DirectoryTree](#) (*class in dtlpy.entities.directory_tree*), 123
[download\(\)](#) (*Annotation method*), 91
[download\(\)](#) (*AnnotationCollection method*), 95
[download\(\)](#) (*Annotations method*), 38
[download\(\)](#) (*Dataset method*), 83
[download\(\)](#) (*Item method*), 88
[download\(\)](#) (*Items method*), 33
[download_annotations\(\)](#) (*Dataset method*), 83
[download_annotations\(\)](#) (*Datasets static method*), 28
[download_partition\(\)](#) (*Dataset method*), 84
[Driver](#) (*class in dtlpy.entities.driver*), 87
[Drivers](#) (*class in dtlpy.repositories.drivers*), 31
[dtlpy.entities.annotation](#) module, 90
[dtlpy.entities.annotation_collection](#) module, 94
[dtlpy.entities.annotation_definitions.base_annotation_defi](#) module, 96
[dtlpy.entities.annotation_definitions.box](#) module, 96
[dtlpy.entities.annotation_definitions.classification](#) module, 97
[dtlpy.entities.annotation_definitions.cube](#) module, 97
[dtlpy.entities.annotation_definitions.description](#) module, 97
[dtlpy.entities.annotation_definitions.ellipse](#) module, 97
[dtlpy.entities.annotation_definitions.note](#) module, 98
[dtlpy.entities.annotation_definitions.point](#) module, 98
[dtlpy.entities.annotation_definitions.polygon](#) module, 98
[dtlpy.entities.annotation_definitions.polyline](#) module, 99
[dtlpy.entities.annotation_definitions.pose](#) module, 99
[dtlpy.entities.annotation_definitions.segmentation](#) module, 99
[dtlpy.entities.annotation_definitions.subtitle](#) module, 100
[dtlpy.entities.annotation_definitions.undefined_annotation](#) module, 100
[dtlpy.entities.assignment](#) module, 108
[dtlpy.entities.base_entity](#) module, 122
[dtlpy.entities.bot](#)

- module, 116
- dtlpy.entities.codebase
 - module, 113
- dtlpy.entities.command
 - module, 122
- dtlpy.entities.dataset
 - module, 82
- dtlpy.entities.directory_tree
 - module, 123
- dtlpy.entities.driver
 - module, 87
- dtlpy.entities.execution
 - module, 118
- dtlpy.entities.filters
 - module, 101
- dtlpy.entities.integration
 - module, 79
- dtlpy.entities.item
 - module, 88
- dtlpy.entities.label
 - module, 106
- dtlpy.entities.links
 - module, 90
- dtlpy.entities.ontology
 - module, 104
- dtlpy.entities.organization
 - module, 77
- dtlpy.entities.package
 - module, 110
- dtlpy.entities.package_function
 - module, 112
- dtlpy.entities.package_module
 - module, 112
- dtlpy.entities.package_slot
 - module, 112
- dtlpy.entities.paged_entities
 - module, 121
- dtlpy.entities.pipeline
 - module, 119
- dtlpy.entities.pipeline_execution
 - module, 121
- dtlpy.entities.project
 - module, 79
- dtlpy.entities.recipe
 - module, 103
- dtlpy.entities.service
 - module, 113
- dtlpy.entities.similarity
 - module, 100
- dtlpy.entities.task
 - module, 106
- dtlpy.entities.trigger
 - module, 116
- dtlpy.entities.user

- module, 81
- dtlpy.repositories.annotations
 - module, 37
- dtlpy.repositories.assignments
 - module, 48
- dtlpy.repositories.bots
 - module, 65
- dtlpy.repositories.codebases
 - module, 57
- dtlpy.repositories.commands
 - module, 74
- dtlpy.repositories.datasets
 - module, 26
- dtlpy.repositories.downloader
 - module, 75
- dtlpy.repositories.drivers
 - module, 31
- dtlpy.repositories.executions
 - module, 68
- dtlpy.repositories.integrations
 - module, 22
- dtlpy.repositories.items
 - module, 32
- dtlpy.repositories.ontologies
 - module, 42
- dtlpy.repositories.organizations
 - module, 19
- dtlpy.repositories.packages
 - module, 51
- dtlpy.repositories.pipeline_executions
 - module, 73
- dtlpy.repositories.pipelines
 - module, 71
- dtlpy.repositories.projects
 - module, 23
- dtlpy.repositories.recipes
 - module, 40
- dtlpy.repositories.services
 - module, 59
- dtlpy.repositories.tasks
 - module, 43
- dtlpy.repositories.triggers
 - module, 66
- dtlpy.repositories.uploader
 - module, 75
- dtlpy.utilities.converter
 - module, 125

E

- Ellipse (*class in dtlpy.entities.annotation_definitions.ellipse*), 97
- execute() (*Pipeline method*), 119
- execute() (*Pipelines method*), 72
- execute() (*Service method*), 114

`execute()` (*Services method*), 62
`Execution` (*class in dtlpy.entities.execution*), 118
`Executions` (*class in dtlpy.repositories.executions*), 68
`ExecutionStatus` (*class in dtlpy.entities.execution*), 119
`ExpirationOptions` (*class in dtlpy.entities.dataset*), 87
`ExportMetadata` (*class in dtlpy.entities.item*), 88
`ExportVersion` (*class in dtlpy.entities.annotation*), 93
`ExternalStorage` (*class in dtlpy.entities.driver*), 87

F

`Filters` (*class in dtlpy.entities.filters*), 101
`FiltersKnownFields` (*class in dtlpy.entities.filters*), 103
`FiltersMethod` (*class in dtlpy.entities.filters*), 103
`FiltersOperations` (*class in dtlpy.entities.filters*), 103
`FiltersOrderByDirection` (*class in dtlpy.entities.filters*), 103
`FiltersResource` (*class in dtlpy.entities.filters*), 103
`FrameAnnotation` (*class in dtlpy.entities.annotation*), 93
`from_boxes_and_angle()` (*Cube class method*), 97
`from_coco()` (*Converter method*), 127
`from_instance_mask()` (*AnnotationCollection method*), 95
`from_json()` (*Annotation class method*), 91
`from_json()` (*BaseTrigger class method*), 116
`from_json()` (*Bot class method*), 116
`from_json()` (*Command class method*), 122
`from_json()` (*CronTrigger class method*), 117
`from_json()` (*Dataset class method*), 84
`from_json()` (*Driver class method*), 87
`from_json()` (*Execution class method*), 118
`from_json()` (*Integration class method*), 79
`from_json()` (*Item class method*), 89
`from_json()` (*Ontology class method*), 105
`from_json()` (*Organization class method*), 77
`from_json()` (*Package class method*), 111
`from_json()` (*Pipeline class method*), 120
`from_json()` (*PipelineExecution class method*), 121
`from_json()` (*Project class method*), 80
`from_json()` (*Recipe class method*), 104
`from_json()` (*Service class method*), 114
`from_json()` (*Trigger class method*), 117
`from_json()` (*User class method*), 81
`from_polygon()` (*Segmentation class method*), 99
`from_segmentation()` (*Box class method*), 96
`from_segmentation()` (*Polygon class method*), 98
`from_snapshot()` (*FrameAnnotation class method*), 93
`from_voc()` (*Converter static method*), 127
`from_vtt_file()` (*AnnotationCollection method*), 95
`from_yolo()` (*Converter method*), 127

G

`generate()` (*Packages static method*), 54
`generate()` (*Workload class method*), 109
`generate_url_query_params()` (*Filters method*), 102

`get()` (*Annotations method*), 38
`get()` (*Assignments method*), 48
`get()` (*Bots method*), 66
`get()` (*Codebases method*), 57
`get()` (*Commands method*), 74
`get()` (*Datasets method*), 29
`get()` (*Drivers method*), 32
`get()` (*Executions method*), 69
`get()` (*Integrations method*), 23
`get()` (*Items method*), 34
`get()` (*Ontologies method*), 42
`get()` (*Organizations method*), 20
`get()` (*Packages method*), 54
`get()` (*PipelineExecutions method*), 74
`get()` (*Pipelines method*), 72
`get()` (*Projects method*), 25
`get()` (*Recipes method*), 41
`get()` (*Services method*), 62
`get()` (*Tasks method*), 45
`get()` (*Triggers method*), 67
`get_all_items()` (*Items method*), 34
`get_annotation_template_id()` (*Recipe method*), 104
`get_current_version()` (*Codebases static method*), 57
`get_field()` (*LocalServiceRunner method*), 51
`get_frame()` (*AnnotationCollection method*), 95
`get_items()` (*Assignment method*), 108
`get_items()` (*Assignments method*), 48
`get_items()` (*Task method*), 107
`get_items()` (*Tasks method*), 45
`get_mainpy_run_service()` (*LocalServiceRunner method*), 51
`get_page()` (*PagedEntities method*), 121
`get_partitions()` (*Dataset method*), 85
`get_recipe_ids()` (*Dataset method*), 85
`go_to_page()` (*PagedEntities method*), 121

H

`has_field()` (*Filters method*), 102

I

`in_progress()` (*Command method*), 122
`increment()` (*Execution method*), 118
`increment()` (*Executions method*), 69
`install()` (*Pipeline method*), 120
`install()` (*Pipelines method*), 72
`InstanceCatalog` (*class in dtlpy.entities.service*), 113
`Integration` (*class in dtlpy.entities.integration*), 79
`Integrations` (*class in dtlpy.repositories.integrations*), 22
`Item` (*class in dtlpy.entities.item*), 88
`Items` (*class in dtlpy.repositories.items*), 32
`items` (*MultiView property*), 101

items (*Similarity property*), 101

ItemStatus (*class in dtlpy.entities.item*), 90

K

KubernetesAutoscalerType (*class in dtlpy.entities.service*), 113

L

labels_to_roots() (*Ontologies static method*), 42

LinkTypeEnum (*class in dtlpy.entities.links*), 90

list() (*Annotations method*), 38

list() (*Assignments method*), 49

list() (*Bots method*), 66

list() (*Codebases method*), 58

list() (*Commands method*), 75

list() (*Datasets method*), 29

list() (*Drivers method*), 32

list() (*Executions method*), 70

list() (*Integrations method*), 23

list() (*Items method*), 35

list() (*Ontologies method*), 42

list() (*Organizations method*), 20

list() (*Packages method*), 54

list() (*PipelineExecutions method*), 74

list() (*Pipelines method*), 73

list() (*Projects method*), 25

list() (*Recipes method*), 41

list() (*Services method*), 63

list() (*Tasks method*), 46

list() (*Triggers method*), 68

list_groups() (*Organization method*), 78

list_groups() (*Organizations method*), 20

list_integrations() (*Organizations method*), 20

list_members() (*Organization method*), 78

list_members() (*Organizations method*), 21

list_members() (*Project method*), 80

list_members() (*Projects method*), 25

list_versions() (*Codebases method*), 58

LocalServiceRunner (*class in dtlpy.repositories.packages*), 51

log() (*Service method*), 114

log() (*Services method*), 63

logs() (*Execution method*), 118

logs() (*Executions method*), 70

M

make_dir() (*Items method*), 35

MemberOrgRole (*class in dtlpy.entities.organization*), 77

MemberRole (*class in dtlpy.entities.project*), 79

merge() (*Datasets method*), 29

Message (*class in dtlpy.entities.annotation_definitions.note*), 98

ModalityRefTypeEnum (*class in dtlpy.entities.item*), 90

ModalityTypeEnum (*class in dtlpy.entities.item*), 90

module

dtlpy.entities.annotation, 90

dtlpy.entities.annotation_collection, 94

dtlpy.entities.annotation_definitions.base_annotation, 96

dtlpy.entities.annotation_definitions.box, 96

dtlpy.entities.annotation_definitions.classification, 97

dtlpy.entities.annotation_definitions.cube, 97

dtlpy.entities.annotation_definitions.description, 97

dtlpy.entities.annotation_definitions.ellipse, 97

dtlpy.entities.annotation_definitions.note, 98

dtlpy.entities.annotation_definitions.point, 98

dtlpy.entities.annotation_definitions.polygon, 98

dtlpy.entities.annotation_definitions.polyline, 99

dtlpy.entities.annotation_definitions.pose, 99

dtlpy.entities.annotation_definitions.segmentation, 99

dtlpy.entities.annotation_definitions.subtitle, 100

dtlpy.entities.annotation_definitions.undefined_annotation, 100

dtlpy.entities.assignment, 108

dtlpy.entities.base_entity, 122

dtlpy.entities.bot, 116

dtlpy.entities.codebase, 113

dtlpy.entities.command, 122

dtlpy.entities.dataset, 82

dtlpy.entities.directory_tree, 123

dtlpy.entities.driver, 87

dtlpy.entities.execution, 118

dtlpy.entities.filters, 101

dtlpy.entities.integration, 79

dtlpy.entities.item, 88

dtlpy.entities.label, 106

dtlpy.entities.links, 90

dtlpy.entities.ontology, 104

dtlpy.entities.organization, 77

dtlpy.entities.package, 110

dtlpy.entities.package_function, 112

dtlpy.entities.package_module, 112

dtlpy.entities.package_slot, 112

dtlpy.entities.paged_entities, 121

dtlpy.entities.pipeline, 119

dtlpy.entities.pipeline_execution, 121

dtlpy.entities.project, 79
dtlpy.entities.recipe, 103
dtlpy.entities.service, 113
dtlpy.entities.similarity, 100
dtlpy.entities.task, 106
dtlpy.entities.trigger, 116
dtlpy.entities.user, 81
dtlpy.repositories.annotations, 37
dtlpy.repositories.assignments, 48
dtlpy.repositories.bots, 65
dtlpy.repositories.codebases, 57
dtlpy.repositories.commands, 74
dtlpy.repositories.datasets, 26
dtlpy.repositories.downloader, 75
dtlpy.repositories.drivers, 31
dtlpy.repositories.executions, 68
dtlpy.repositories.integrations, 22
dtlpy.repositories.items, 32
dtlpy.repositories.ontologies, 42
dtlpy.repositories.organizations, 19
dtlpy.repositories.packages, 51
dtlpy.repositories.pipeline_executions, 73
dtlpy.repositories.pipelines, 71
dtlpy.repositories.projects, 23
dtlpy.repositories.recipes, 40
dtlpy.repositories.services, 59
dtlpy.repositories.tasks, 43
dtlpy.repositories.triggers, 66
dtlpy.repositories.uploader, 75
dtlpy.utilities.converter, 125
move() (*Item method*), 89
move_items() (*Items method*), 35
MultiView (*class in dtlpy.entities.similarity*), 101
MultiViewItem (*class in dtlpy.entities.similarity*), 101

N

name_validation() (*Services method*), 63
name_validation() (*Triggers method*), 68
new() (*Annotation class method*), 92
new() (*FrameAnnotation class method*), 94
next_page() (*PagedEntities method*), 121
Note (*class in dtlpy.entities.annotation_definitions.note*), 98

O

OnResetAction (*class in dtlpy.entities.service*), 113
Ontologies (*class in dtlpy.repositories.ontologies*), 42
Ontology (*class in dtlpy.entities.ontology*), 104
open_in_web() (*Assignment method*), 108
open_in_web() (*Assignments method*), 49
open_in_web() (*Dataset method*), 85
open_in_web() (*Datasets method*), 30
open_in_web() (*Item method*), 89

open_in_web() (*Items method*), 35
open_in_web() (*Organization method*), 78
open_in_web() (*Package method*), 111
open_in_web() (*Packages method*), 55
open_in_web() (*Pipeline method*), 120
open_in_web() (*Pipelines method*), 73
open_in_web() (*Project method*), 80
open_in_web() (*Projects method*), 25
open_in_web() (*Recipe method*), 104
open_in_web() (*Recipes method*), 41
open_in_web() (*Service method*), 115
open_in_web() (*Services method*), 63
open_in_web() (*Task method*), 108
open_in_web() (*Tasks method*), 46
Organization (*class in dtlpy.entities.organization*), 77
Organizations (*class in dtlpy.repositories.organizations*), 19
OrganizationsPlans (*class in dtlpy.entities.organization*), 78

P

pack() (*Codebases method*), 58
Package (*class in dtlpy.entities.package*), 110
PackageFunction (*class in dtlpy.entities.package_function*), 112
PackageInputType (*class in dtlpy.entities.package_function*), 112
PackageModule (*class in dtlpy.entities.package_module*), 112
Packages (*class in dtlpy.repositories.packages*), 51
PackageSlot (*class in dtlpy.entities.package_slot*), 112
PagedEntities (*class in dtlpy.entities.paged_entities*), 121
pause() (*Pipeline method*), 120
pause() (*Pipelines method*), 73
pause() (*Service method*), 115
pause() (*Services method*), 64
Pipeline (*class in dtlpy.entities.pipeline*), 119
PipelineExecution (*class in dtlpy.entities.pipeline_execution*), 121
PipelineExecutions (*class in dtlpy.repositories.pipeline_executions*), 73
Pipelines (*class in dtlpy.repositories.pipelines*), 71
Point (*class in dtlpy.entities.annotation_definitions.point*), 98
Polygon (*class in dtlpy.entities.annotation_definitions.polygon*), 98
Polyline (*class in dtlpy.entities.annotation_definitions.polyline*), 99
pop() (*Collection method*), 100
pop() (*Filters method*), 102
pop_join() (*Filters method*), 102
Pose (*class in dtlpy.entities.annotation_definitions.pose*), 99

prepare() (*Filters method*), 102
 prev_page() (*PagedEntities method*), 122
 print() (*AnnotationCollection method*), 95
 process_result() (*PagedEntities method*), 122
 progress_update() (*Execution method*), 118
 progress_update() (*Executions method*), 70
 Project (*class in dtlpy.entities.project*), 79
 Projects (*class in dtlpy.repositories.projects*), 23
 pull() (*Package method*), 111
 pull() (*Packages method*), 55
 pull_git() (*Codebases method*), 58
 push() (*Package method*), 111
 push() (*Packages method*), 55

Q

query() (*Tasks method*), 47

R

reassign() (*Assignment method*), 108
 reassign() (*Assignments method*), 49
 Recipe (*class in dtlpy.entities.recipe*), 103
 Recipes (*class in dtlpy.repositories.recipes*), 40
 redistribute() (*Assignment method*), 109
 redistribute() (*Assignments method*), 50
 remove_member() (*Project method*), 80
 remove_member() (*Projects method*), 25
 RequirementOperator (*class in dtlpy.entities.package*), 112
 rerun() (*Execution method*), 119
 rerun() (*Executions method*), 70
 resource_information() (*Triggers method*), 68
 resume() (*Service method*), 115
 resume() (*Services method*), 64
 return_page() (*PagedEntities method*), 122
 revisions() (*Packages method*), 56
 revisions() (*Services method*), 64
 run_local_project() (*LocalServiceRunner method*), 51
 RuntimeType (*class in dtlpy.entities.service*), 113

S

save_to_file() (*Converter method*), 127
 Segmentation (*class in dtlpy.entities.annotation_definitions.segmentation*), 99
 serialize_labels() (*Dataset static method*), 85
 Service (*class in dtlpy.entities.service*), 113
 ServiceLog (*class in dtlpy.repositories.services*), 59
 Services (*class in dtlpy.repositories.services*), 59
 set_description() (*Item method*), 89
 set_frame() (*Annotation method*), 92
 set_items_entity() (*Items method*), 36
 set_partition() (*Dataset method*), 85
 set_readonly() (*Dataset method*), 85

set_readonly() (*Datasets method*), 30
 set_start_node() (*Pipeline method*), 120
 set_status() (*Assignment method*), 109
 set_status() (*Assignments method*), 50
 set_status() (*Task method*), 108
 set_status() (*Tasks method*), 47
 show() (*Annotation method*), 92
 show() (*AnnotationCollection method*), 95
 show() (*Annotations method*), 39
 show() (*Box method*), 96
 show() (*Classification method*), 97
 show() (*Cube method*), 97
 show() (*Ellipse method*), 97
 show() (*FrameAnnotation method*), 94
 show() (*Point method*), 98
 show() (*Polygon method*), 98
 show() (*Polyline method*), 99
 show() (*Pose method*), 99
 show() (*Segmentation method*), 99
 show() (*UndefinedAnnotationType method*), 100
 Similarity (*class in dtlpy.entities.similarity*), 101
 SimilarityItem (*class in dtlpy.entities.similarity*), 101
 SimilarityTypeEnum (*class in dtlpy.entities.similarity*), 101
 SingleDirectory (*class in dtlpy.entities.directory_tree*), 123
 SlotDisplayScopeResource (*class in dtlpy.entities.package_slot*), 112
 SlotPostActionType (*class in dtlpy.entities.package_slot*), 112
 sort_by() (*Filters method*), 103
 status() (*Service method*), 115
 status() (*Services method*), 64
 Subtitle (*class in dtlpy.entities.annotation_definitions.subtitle*), 100
 switch_recipe() (*Dataset method*), 85
 sync() (*Dataset method*), 86
 sync() (*Datasets method*), 30

T

target (*Similarity property*), 101
 Task (*class in dtlpy.entities.task*), 106
 Tasks (*class in dtlpy.repositories.tasks*), 43
 tear_down() (*Services method*), 65
 terminate() (*Execution method*), 119
 terminate() (*Executions method*), 70
 test_local_package() (*Packages method*), 56
 to_box() (*Segmentation method*), 100
 to_coco() (*Converter static method*), 128
 to_json() (*Annotation method*), 93
 to_json() (*AnnotationCollection method*), 96
 to_json() (*Assignment method*), 109
 to_json() (*BaseTrigger method*), 117
 to_json() (*Bot method*), 116

`to_json()` (*Collection method*), 100
`to_json()` (*Command method*), 122
`to_json()` (*CronTrigger method*), 117
`to_json()` (*Dataset method*), 86
`to_json()` (*Driver method*), 87
`to_json()` (*Execution method*), 119
`to_json()` (*Integration method*), 79
`to_json()` (*Item method*), 89
`to_json()` (*MultiView method*), 101
`to_json()` (*Ontology method*), 105
`to_json()` (*Organization method*), 78
`to_json()` (*Package method*), 111
`to_json()` (*Pipeline method*), 120
`to_json()` (*PipelineExecution method*), 121
`to_json()` (*Project method*), 80
`to_json()` (*Recipe method*), 104
`to_json()` (*Service method*), 115
`to_json()` (*Similarity method*), 101
`to_json()` (*Task method*), 108
`to_json()` (*Trigger method*), 117
`to_json()` (*User method*), 81
`to_voc()` (*Converter static method*), 128
`to_yolo()` (*Converter method*), 128
`Trigger` (*class in dtlpy.entities.trigger*), 117
`TriggerAction` (*class in dtlpy.entities.trigger*), 118
`TriggerExecutionMode` (*class in dtlpy.entities.trigger*), 118
`TriggerResource` (*class in dtlpy.entities.trigger*), 118
`Triggers` (*class in dtlpy.repositories.triggers*), 66
`TriggerType` (*class in dtlpy.entities.trigger*), 118

U

`UiBindingPanel` (*class in dtlpy.entities.package_slot*), 113

`UndefinedAnnotationType` (*class in dtlpy.entities.annotation_definitions.undefined_annotation*), 100

`unpack()` (*Codebases method*), 58
`update()` (*Annotation method*), 93
`update()` (*Annotations method*), 39
`update()` (*Assignment method*), 109
`update()` (*Assignments method*), 50
`update()` (*BaseTrigger method*), 117
`update()` (*Dataset method*), 86
`update()` (*Datasets method*), 30
`update()` (*Execution method*), 119
`update()` (*Executions method*), 71
`update()` (*Integration method*), 79
`update()` (*Integrations method*), 23
`update()` (*Item method*), 89
`update()` (*Items method*), 36
`update()` (*Ontologies method*), 43
`update()` (*Ontology method*), 105
`update()` (*Organization method*), 78

`update()` (*Organizations method*), 21
`update()` (*Package method*), 112
`update()` (*Packages method*), 56
`update()` (*Pipeline method*), 120
`update()` (*Pipelines method*), 73
`update()` (*Project method*), 81
`update()` (*Projects method*), 26
`update()` (*Recipe method*), 104
`update()` (*Recipes method*), 41
`update()` (*Service method*), 115
`update()` (*Services method*), 65
`update()` (*Task method*), 108
`update()` (*Tasks method*), 47
`update()` (*Triggers method*), 68
`update_label()` (*Dataset method*), 86
`update_label()` (*Ontology method*), 106
`update_labels()` (*Dataset method*), 86
`update_labels()` (*Ontology method*), 106
`update_member()` (*Organization method*), 78
`update_member()` (*Organizations method*), 21
`update_member()` (*Project method*), 81
`update_member()` (*Projects method*), 26
`update_status()` (*Annotation method*), 93
`update_status()` (*Annotations method*), 39
`update_status()` (*Item method*), 89
`update_status()` (*Items method*), 36
`upload()` (*Annotation method*), 93
`upload()` (*Annotations method*), 40
`upload()` (*Items method*), 36
`upload_annotations()` (*Dataset method*), 86
`upload_annotations()` (*Datasets method*), 31
`upload_local_dataset()` (*Converter method*), 129
`User` (*class in dtlpy.entities.user*), 81

V

`view()` (*ServiceLog method*), 59

`ViewAnnotationOptions` (*class in dtlpy.entities.annotation*), 94

W

`wait()` (*Command method*), 122
`wait()` (*Commands method*), 75
`wait()` (*Execution method*), 119
`wait()` (*Executions method*), 71
`Workload` (*class in dtlpy.entities.assignment*), 109
`WorkloadUnit` (*class in dtlpy.entities.assignment*), 109