dtlpy Documentation

Release 1.48.21

Or Shabtay

Jan 31, 2022

TABLE OF CONTENTS

CHAPTER

ONE

COMMAND LINE INTERAFCE

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 chuck
-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]
```

CHAPTER

TWO

REPOSITORIES

2.1 Organizations

class Organizations(*client_api: dtlpy.services.api_client.ApiClient*)

Bases: `object`

organizations repository

add_member(*email*, *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 member to the Organization :param email: :param role: MemberOrgRole.ADMIN ,MemberOrgRole.OWNER ,MemberOrgRole.MEMBER :param organization_id: :param organization_name: :param organization: :return: True

create(*organization_json*) → *dtlpy.entities.organization.Organization*

Create a new pipeline :param organization_json: json contain the Organization fields :return: Pipeline object

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 :param user_id: :param organization_id: :param organization_name: :param organization: :param sure: are you sure you want to delete? :param really: really really? :return: True

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

Get a Organization object :param organization_id: optional - search by id :param organization_name: optional - search by name :param fetch: optional - fetch entity from platform, default taken from cookie :return: Organization object

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

Get Organization's list. :return: List of Organization objects

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

list all organization groups :param organization: :param organization_id: :param organization_name: :return groups 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 :param organization: :param organization_id: :param organization_name: :param only_available: bool - if True list only the available integrations :return groups 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 :param organization: :param organization_id: :param organization_name: :param role: MemberOrgRole.ADMIN ,MemberOrgRole.OWNER ,MemberOrgRole.MEMBER :return projects 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 a organization :param plan: OrganizationsPlans.FREEMIUM, OrganizationsPlans.PREMIUM :param organization: :param organization_id: :param organization_name: :return: organization object

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 the member role :param email: :param role: MemberOrgRole.ADMIN ,MemberOrgRole.OWNER ,MemberOrgRole.MEMBER :param organization_id: :param organization_name: :param organization:

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`

Datasets repository

create(*integrations_type: dtlpy.entities.driver.ExternalStorage, name, options*)

Add integrations to the Organization :param integrations_type: dl.ExternalStorage :param name: integrations name :param options: s3 - {key: “”, secret: “”},

gcs - {key: “”, secret: “”, content: “”}, azureblob - {key: “”, secret: “”, clientId: “”, tenantId: “”} key_value - {key: “”, value: “”}

Returns True

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

Delete integrations from the Organization :param integrations_id: :param sure: are you sure you want to delete? :param really: really really? :return: True

get(*integrations_id: str*)

get organization integrations :param integrations_id: :return organization integrations:

list(*only_available=False*)

list all organization integrations :param only_available: bool - if True list only the available integrations :return groups list:

update(*new_name: str, integrations_id*)

Update the integrations name :param new_name: :param integrations_id:

2.2 Projects

class Projects(*client_api: dtlpy.services.api_client.ApiClient*, *org=None*)

Bases: `object`

Projects repository

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

Parameters

- **email** –
- **project_id** –
- **role** – "owner" ,"engineer" ,"annotator" ,"annotationManager"

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

Check-out a project :param identifier: project name or partial id :param project_name: :param project_id: :param project: :return:

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

Create a new project :param project_name: :param checkout: :return: Project object

delete(*project_name: Optional[str] = None*, *project_id: Optional[str] = None*, *sure: bool = False*, *really: bool = False*) → bool

Delete a project forever! :param project_name: optional - search by name :param project_id: optional - search by id :param sure: are you sure you want to delete? :param really: really really?

Returns True

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 :param project_name: optional - search by name :param project_id: optional - search by id :param checkout: :param fetch: optional - fetch entity from platform, default taken from cookie :return: Project object

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

Get users project's list. :return: List of Project objects

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

Parameters

- **project** –
- **role** – "owner" ,"engineer" ,"annotator" ,"annotationManager"

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

Parameters

- **project_name** –
- **project_id** –
- **project** –

remove_member(*email: str*, *project_id: str*)

Parameters

- **email** –
- **project_id** –

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

Update a project :param project: :param system_metadata: True, if you want to change metadata system :return: Project object

update_member(*email: str*, *project_id: str*, *role:* dtlpy.entities.project.MemberRole = *MemberRole.DEVELOPER*)

Parameters

- **email** –
- **project_id** –
- **role** – "owner" ,"engineer" ,"annotator" ,"annotationManager"

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

checkout(*identifier=None*, *dataset_name=None*, *dataset_id=None*, *dataset=None*)

Check-out a project :param identifier: project name or partial id :param dataset_name: :param dataset_id: :param dataset: :return:

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

Clone a dataset

Parameters

- **dataset_id** – to clone dataset
- **clone_name** – new dataset name
- **filters** – Filters entity or a query dict
- **with_items_annotations** –
- **with_metadata** –
- **with_task_annotations_status** –

Returns

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

Create a new dataset

Parameters

- **dataset_name** – name
- **labels** – dictionary of {tag: color} or list of label entities
- **attributes** – dataset's ontology's attributes
- **ontology_ids** – optional - dataset ontology
- **driver** – optional - storage driver Driver object or driver name
- **driver_id** – optional - driver id
- **checkout** – bool. cache the dataset to work locally
- **expiration_options** – dl.ExpirationOptions object that contain definitions for dataset like MaxItemDays

Returns Dataset object

delete(*dataset_name=None*, *dataset_id=None*, *sure=False*, *really=False*)

Delete a dataset forever! :param dataset_name: optional - search by name :param dataset_id: optional - search by id :param sure: are you sure you want to delete? :param really: really really? :return: True

directory_tree(*dataset: Optional[*dtlpy.entities.dataset.Dataset*] = None*, *dataset_name=None*, *dataset_id=None*)

Get dataset's directory tree :param dataset: :param dataset_name: :param dataset_id: :return:

static download_annotations(*dataset*, *local_path=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=False*, *thickness=1*, *with_text=False*, *remote_path=None*, *include_annotations_in_output=True*, *export_png_files=False*, *filter_output_annotations=False*, *alpha=None*)

Download dataset's annotations by filters. Filtering the dataset both for items and for annotations and download annotations Optional - also download annotations as: mask, instance, image mask of the item

Parameters

- **dataset** – dataset to download from
- **local_path** – local folder or filename to save to.
- **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
- **include_annotations_in_output** – default - False , if export should contain annotations
- **export_png_files** – default - True, if semantic annotations should 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

Returns *List* of local_path per each downloaded item

get(*dataset_name=None*, *dataset_id=None*, *checkout=False*, *fetch=None*) → *dtlpy.entities.dataset.Dataset*

Get dataset by name or id

Parameters

- **dataset_name** – optional - search by name
- **dataset_id** – optional - search by id
- **checkout** –
- **fetch** – optional - fetch entity from platform, default taken from cookie

Returns Dataset object

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

List all datasets. :param name: :param creator: :return: List of datasets

merge(*merge_name*, *dataset_ids*, *project_ids*, *with_items_annotations=True*, *with_metadata=True*, *with_task_annotations_status=True*, *wait=True*)

merge a dataset

Parameters

- **merge_name** – to clone dataset
- **dataset_ids** – new dataset name
- **project_ids** – Filters entity or a query dict
- **with_items_annotations** –
- **with_metadata** –
- **with_task_annotations_status** –
- **wait** – wait the command to finish

Returns

open_in_web(*dataset_name=None*, *dataset_id=None*, *dataset=None*)

Parameters

- **dataset_name** –
- **dataset_id** –
- **dataset** –

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

Set dataset readonly mode :param state: :param dataset: :return:

sync(*dataset_id*, *wait=True*)

Sync dataset with external storage

Parameters

- **dataset_id** – to sync dataset
- **wait** – wait the command to finish

Returns

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

Update dataset field :param dataset: Dataset entity :param system_metadata: bool - True, if you want to change metadata system :param patch: Specific patch request :return: Dataset object

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

Upload annotations to dataset. :param dataset: dataset to upload to it :param local_path: str - local folder where the annotations files is. :param filters: Filters entity or a dictionary containing filters parameters :param clean: bool - if True it remove the old annotations :param remote_root_path: str - the remote root path to match remote and local items 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 :return:

2.3.1 Drivers

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

Bases: `object`

Drivers repository

create(*name*, *driver_type*, *integration_id*, *bucket_name*, *project_id=None*, *allow_external_delete=True*, *region=None*, *storage_class=''*, *path=''*)

Parameters

- **name** – the driver name
- **driver_type** – ExternalStorage.S3, ExternalStorage.GCS , ExternalStorage.AZUREBLOB
- **integration_id** – the integration id
- **bucket_name** – the external bucket name
- **project_id** –
- **allow_external_delete** –
- **region** – rilevante only for s3 - the bucket region
- **storage_class** – rilevante only for s3
- **path** – Optional. By default path is the root folder. Path is case sensitive integration

Returns driver object

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

Get a Driver object :param driver_name: optional - search by name :param driver_id: optional - search by id :return: Driver object

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

Get project's drivers list. :return: List of Drivers objects

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

clone(*item_id*, *dst_dataset_id*, *remote_filepath=None*, *metadata=None*, *with_annotations=True*, *with_metadata=True*, *with_task_annotations_status=False*, *allow_many=False*, *wait=True*)

Clone item :param item_id: item to clone :param dst_dataset_id: destination dataset id :param remote_filepath: complete filepath :param metadata: new metadata to add :param with_annotations: clone annotations :param with_metadata: clone metadata :param with_task_annotations_status: clone task annotations status :param allow_many: *bool* if True use multiple clones in single dataset is allowed, (default=False) :param wait: wait the command to finish :return: Item

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

Delete item from platform

Parameters

- **filename** – optional - search item by remote path
- **item_id** – optional - search item by id
- **filters** – optional - delete items by filter

Returns True

download(*filters: Optional[*dtlpy.entities.filters.Filters*] = None*, *items=None*, *local_path=None*, *file_types=None*, *save_locally=True*, *to_array=False*, *annotation_options: Optional[*dtlpy.entities.annotation.ViewAnnotationOptions*] = None*, *annotation_filters: Optional[*dtlpy.entities.filters.Filters*] = None*, *overwrite=False*, *to_items_folder=True*, *thickness=1*, *with_text=False*, *without_relative_path=None*, *avoid_unnecessary_annotation_download=False*, *include_annotations_in_output=True*, *export_png_files=False*, *filter_output_annotations=False*, *alpha=None*)

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** – Filters entity or a dictionary containing filters parameters
- **items** – download Item entity or item_id (or a list of item)
- **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']
- **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)
- **annotation_filters** – Filters entity to filter annotations for download
- **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** – bool - download items without the relative path from platform
- **avoid_unnecessary_annotation_download** – default - False
- **include_annotations_in_output** – default - False , if export should contain annotations
- **export_png_files** – default - True, if semantic annotations should 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

Returns *List* of local_path per each downloaded item

get(*filepath=None*, *item_id=None*, *fetch=None*, *is_dir=False*) → *dtlpy.entities.item.Item*

Get Item object

Parameters

- **filepath** – optional - search by remote path
- **item_id** – optional - search by id
- **fetch** – optional - fetch entity from platform, default taken from cookie
- **is_dir** – True if you want to get an item from dir type

Returns Item object

get_all_items()

Get all items in dataset :param filters: dl.Filters entity to filters items :return: list of all items

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

List items

Parameters

- **filters** – Filters entity or a dictionary containing filters parameters
- **page_offset** – start page
- **page_size** – page size

Returns Pages object

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

Create a directory in a dataset

Parameters

- **directory** – name of directory
- **dataset** – optional

Returns

move_items(*destination*, *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

Parameters

- **destination** – destination directory
- **filters** – optional - either this or items. Query of items to move
- **items** – optional - either this or filters. A list of items to move
- **dataset** – optional

Returns True if success

open_in_web(*filepath=None*, *item_id=None*, *item=None*)

Parameters

- **filepath** – item file path
- **item_id** – item id
- **item** – item entity

update(*item: Optional[*dtlpy.entities.item.Item*] = None*, *filters: Optional[*dtlpy.entities.filters.Filters*] = None*, *update_values=None*, *system_update_values=None*, *system_metadata=False*)

Update items metadata :param item: Item object :param filters: optional update filtered items by given filter :param update_values: optional field to be updated and new values :param system_update_values: values in system metadata to be updated :param system_metadata: bool - True, if you want to change metadata system :return: Item object

update_status(*status:* dtlpy.entities.item.ItemStatus, *items=None*, *item_ids=None*, *filters=None*, *dataset=None*, *clear=False*)

Parameters

- **status** – ItemStatus.COMPLETED, ItemStatus.APPROVED, ItemStatus.DISCARDED
- **items** –
- **item_ids** –
- **filters** –
- **dataset** –
- **clear** –

upload(*local_path*, *local_annotations_path=None*, *remote_path='/'*, *remote_name=None*, *file_types=None*, *overwrite=False*, *item_metadata=None*, *output_entity=<class 'dtlpy.entities.item.Item'>*, *no_output=False*)

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

Parameters

- **local_path** – list of local file, local folder, BufferIO, numpy.ndarray or url to upload
- **local_annotations_path** – path to dataloop format annotations json files.
- **remote_path** – remote path to save.
- **remote_name** – remote base name to save. when upload numpy.ndarray as local path, remote_name with .jpg or .png ext is mandatory
- **file_types** – list of file type to upload. e.g ['.jpg', '.png']. default is all

- **item_metadata** –
- **overwrite** – optional - default = False
- **output_entity** – output type
- **no_output** – do not return the items after upload

Returns Output (list/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

delete(*annotation=None*, *annotation_id=None*, *filters: Optional[*dtlpy.entities.filters.Filters*] = None*)

Remove an annotation from item

Parameters

- **annotation** – Annotation object
- **annotation_id** – annotation id
- **filters** – Filters entity or a dictionary containing filters parameters

Returns True/False

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

Save annotation format to file

Parameters

- **filepath** – Target download directory
- **annotation_format** – optional - list(dl.ViewAnnotationOptions)
- **img_filepath** – img file path - needed for img_mask
- **height** – optional - image height
- **width** – optional - image width
- **thickness** – optional - annotation format, default =1
- **with_text** – optional - draw annotation with text, default = False
- **alpha** – opacity value [0 1], default 1

Returns

get(*annotation_id*)

Get a single annotation

Parameters **annotation_id** –

Returns Annotation object or None

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

List Annotation

Parameters

- **filters** – Filters entity or a dictionary containing filters parameters
- **page_offset** – starting page
- **page_size** – size of page

Returns Pages object

`show`(*image=None, thickness=1, with_text=False, height=None, width=None, annotation_format: dtlpy.entities.annotation.ViewAnnotationOptions = ViewAnnotationOptions.MASK, alpha=None*)

Show annotations

Parameters

- **image** – empty or image to draw on
- **thickness** – line thickness
- **with_text** – add label to annotation
- **height** – height
- **width** – width
- **annotation_format** – options: list(dl.ViewAnnotationOptions)
- **alpha** – opacity value [0 1], default 1

Returns ndarray of the annotations

`update`(*annotations, system_metadata=False*)

Update an existing annotation.

Parameters

- **annotations** – annotations object
- **system_metadata** – bool - True, if you want to change metadata system

Returns True

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

Set status on annotation

Parameters

- **annotation** – optional - Annotation entity
- **annotation_id** – optional - annotation id to set status
- **status** – can be AnnotationStatus.ISSUE, AnnotationStatus.APPROVED, AnnotationStatus.REVIEW, AnnotationStatus.CLEAR

Returns Annotation object

`upload`(*annotations*)

Create a new annotation

Parameters **annotations** – list or single annotation of type Annotation

Returns list of annotation objects

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`

Items repository

clone(*recipe: Optional[*dtlpy.entities.recipe.Recipe*] = None*, *recipe_id=None*, *shallow=False*)

Clone Recipe

Parameters

- **recipe** – Recipe object
- **recipe_id** – Recipe id
- **shallow** – If True, link ot existing ontology, clones all ontology that are link to the recipe as well

Returns Cloned ontology object

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

Create New Recipe

if ontology_ids is None an ontology will be created first :param project_ids: :param ontology_ids: :param labels: :param recipe_name: :param attributes:

delete(*recipe_id*)

Delete recipe from platform

Parameters **recipe_id** – recipe id

Returns True

get(*recipe_id*) → *dtlpy.entities.recipe.Recipe*

Get Recipe object

Parameters **recipe_id** – recipe id

Returns Recipe object

list(*filters: Optional[*dtlpy.entities.filters.Filters*] = None*) → dtlpy.miscellaneous.list_print.List[*dtlpy.entities.recipe.Recipe*]

List recipes for dataset :param filters:

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

Parameters

- **recipe** –
- **recipe_id** –

update(*recipe:* dtlpy.entities.recipe.Recipe, *system_metadata=False*) → *dtlpy.entities.recipe.Recipe*
Update items metadata

Parameters

- **recipe** – Recipe object
- **system_metadata** – bool - True, if you want to change metadata system

Returns Recipe object

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

create(*labels*, *title=None*, *project_ids=None*, *attributes=None*) → *dtlpy.entities.ontology.Ontology*
Create a new ontology

Parameters

- **labels** – recipe tags
- **title** – ontology title, name
- **project_ids** – recipe project/s
- **attributes** – recipe attributes

Returns Ontology object

delete(*ontology_id*)
Delete Ontology from platform

Parameters **ontology_id** – ontology_id id

Returns True

get(*ontology_id*) → *dtlpy.entities.ontology.Ontology*
Get Ontology object

Parameters **ontology_id** – ontology id

Returns Ontology object

static labels_to_roots(*labels*)
Converts labels dict to a list of platform representation of labels

Parameters **labels** – labels dict

Returns platform representation of labels

list(*project_ids=None*) → dtlpy.miscellaneous.list_print.List[*dtlpy.entities.ontology.Ontology*]
List ontologies for recipe :param project_ids: :return:

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

Update Ontology metadata

Parameters

- **ontology** – Ontology object
- **system_metadata** – bool - True, if you want to change metadata system

Returns Ontology object

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

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 Task

:param task :param task_id: :param filters: :param items: :param assignee_ids: :param query: :param workload: :param limit: :param wait: wait the command to finish :return:

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

Parameters

- **task_name** –
- **due_date** –
- **assignee_ids** –
- **workload** –
- **dataset** –
- **task_owner** –
- **task_type** – "annotation" or "qa"
- **task_parent_id** – optional if type is qa - parent task id
- **project_id** –
- **recipe_id** –
- **assignments_ids** –
- **metadata** –
- **filters** –
- **items** –
- **query** –
- **available_actions** –

- **wait** – wait the command to finish
- **check_if_exist** – dl.Filters check if task exist according to filter

Returns Annotation Task object

create_qa_task(*task*, *assignee_ids*, *due_date=None*, *filters=None*, *items=None*, *query=None*) → *dtlpy.entities.task.Task*

Parameters

- **task** –
- **assignee_ids** –
- **due_date** –
- **filters** –
- **items** –
- **query** –

delete(*task: Optional[*dtlpy.entities.task.Task*] = None*, *task_name=None*, *task_id=None*, *wait=True*)

Delete an Annotation Task :param task: :param task_name: :param task_id: :param wait: wait the command to finish :return: True

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

Get an Annotation Task object :param task_name: optional - search by name :param task_id: optional - search by id :return: task_id object

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*

Parameters

- **task_id** –
- **task_name** –
- **dataset** –
- **filters** –

Returns

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*]

Get Annotation Task list :param project_ids: list of project ids :param status: :param task_name: task name :param pages_size: :param page_offset: :param recipe: :param creator: :param assignments: assignments :param min_date:double :param max_date: double :param filters: dl.Filters entity to filters items :return: List of Annotation Task objects

open_in_web(*task_name=None*, *task_id=None*, *task=None*)

Parameters

- **task_name** –

- **task_id** –
- **task** –

query(*filters=None*, *project_ids=None*)

Parameters

- **filters** –
- **project_ids** –

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

Update item status within task

Parameters

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

:return : Boolean

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

Update an Annotation Task :param task: task entity :param system_metadata: True, if you want to change metadata system :return: Annotation Task object

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

create(*assignee_id*, *task=None*, *filters=None*, *items=None*) → *dtlpy.entities.assignment.Assignment*

Create a new assignment :param assignee_id: the assignee for the assignment :param task: task entity :param filters: Filters entity or a dictionary containing filters parameters :param items: list of items :return: Assignment object

get(*assignment_name=None*, *assignment_id=None*)

Get a Project object :param assignment_name: optional - search by name :param assignment_id: optional - search by id :return: Project object

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 :param assignment: assignment entity :param assignment_id: assignment id :param assignment_name: assignment name :param dataset: dataset entity :param filters: Filters entity or a dictionary containing filters parameters :return:

list(*project_ids=None*, *status=None*, *assignment_name=None*, *assignee_id=None*, *pages_size=None*, *page_offset=None*, *task_id=None*) → dtlpy.miscellaneous.list_print.List[*dtlpy.entities.assignment.Assignment*]

Get Assignments list

Parameters

- **project_ids** – list of project ids
- **status** –
- **assignment_name** –
- **assignee_id** –
- **pages_size** –
- **page_offset** –
- **task_id** –

Returns List of Assignment objects

open_in_web(*assignment_name=None*, *assignment_id=None*, *assignment=None*)

Parameters

- **assignment_name** –
- **assignment_id** –
- **assignment** –

reassign(*assignee_id*, *assignment=None*, *assignment_id=None*, *task=None*, *task_id=None*, *wait=True*)

Reassign an assignment :param assignee_id: :param assignment: :param assignment_id: :param task: :param task_id: :param wait: wait the command to finish :return: Assignment object

redistribute(*workload*, *assignment=None*, *assignment_id=None*, *task=None*, *task_id=None*, *wait=True*)

Redistribute an assignment :param workload: :param assignment: :param assignment_id: :param task: :param task_id: :param wait: wait the command to finish :return: Assignment object

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

Set item status within assignment @param status: str @param operation: created/deleted @param item_id: str @param assignment_id: str @return: Boolean

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

Update an assignment :param assignment: assignment entity :param system_metadata: bool - True, if you want to change metadata system :return: Assignment object

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 :param field_name: :param field_type: :param mock_json: :param project: :param mock_inputs: :return:

get_mainpy_run_service()
Get mainpy run service :return:

run_local_project(*project=None*)

Parameters **project** –

class Packages(*client_api: dtlpy.services.api_client.ApiClient*, *project: Optional[*dtlpy.entities.project.Project*] = None*)
Bases: object

Packages Repository

build_requirements(*filepath*) → dtlpy.repositories.packages.Packages.list
build a requirements list from file path :param filepath: path of the requirements file :return: a list of dl.PackageRequirement

static build_trigger_dict(*actions*, *name='default_module'*, *filters=None*, *function='run'*, *execution_mode='Once'*, *type_t='Event'*)
build trigger dict :param actions: :param name: :param filters: :param function: :param execution_mode: :param type_t:

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

Parameters

- **cls** –
- **missing** –
- **function_name** –
- **function_inputs** –

checkout(*package=None*, *package_id=None*, *package_name=None*)
Checkout as package :param package: :param package_id: :param package_name: :return:

delete(*package: Optional[*dtlpy.entities.package.Package*] = None*, *package_name=None*, *package_id=None*)
Delete Package object

Parameters

- **package** –
- **package_name** –
- **package_id** –

Returns True

deploy(*package_id=None*, *package_name=None*, *package=None*, *service_name=None*, *project_id=None*, *revision=None*, *init_input=None*, *runtime=None*, *sdk_version=None*, *agent_versions=None*, *bot=None*, *pod_type=None*, *verify=True*, *checkout=False*, *module_name=None*, *run_execution_as_process=None*, *execution_timeout=None*, *drain_time=None*, *on_reset=None*, *max_attempts=None*, *force=False*, ***kwargs*) → *dtlpy.entities.service.Service*
Deploy package :param package_id: :param package_name: :param package: :param service_name: :param project_id: :param revision: :param init_input: :param runtime: :param sdk_version: - optional - string - sdk version :param agent_versions: - dictionary - - optional -versions of sdk, agent runner and agent proxy :param bot: :param pod_type: :param verify: :param checkout: :param module_name: :param run_execution_as_process: :param execution_timeout: :param drain_time: :param on_reset: :param

max_attempts: Maximum execution retries in-case of a service reset :param force: optional - terminate old replicas immediately :return:

deploy_from_file(*project*, *json_filepath*)

Parameters

- **project** –
- **json_filepath** –

static generate(*name=None*, *src_path=None*, *service_name=None*, *package_type='default_package_type'*)

Generate new package environment :param name: :param src_path: :param service_name: :param package_type: :return:

get(*package_name=None*, *package_id=None*, *checkout=False*, *fetch=None*) → *dtlpy.entities.package.Package*

Get Package object :param package_name: :param package_id: :param checkout: bool :param fetch: optional - fetch entity from platform, default taken from cookie :return: Package object

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

List project packages :param filters: :param project_id: :return:

open_in_web(*package=None*, *package_id=None*, *package_name=None*)

Parameters

- **package** –
- **package_id** –
- **package_name** –

pull(*package:* dtlpy.entities.package.Package, *version=None*, *local_path=None*, *project_id=None*)

Parameters

- **package** –
- **version** –
- **local_path** –
- **project_id** –

Returns

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

Parameters

- **project** – optional - project entity to deploy to. default from context or checked-out
- **project_id** – optional - project id to deploy to. default from context or checked-out
- **package_name** – package name
- **src_path** – path to package codebase
- **codebase** –
- **modules** – list of modules PackageModules of the package
- **is_global** –
- **checkout** – checkout package to local dir
- **revision_increment** – optional - str - version bumping method - major/minor/patch - default = None
- **version** – semver version f the package
- **ignore_sanity_check** – NOT RECOMMENDED - skip code sanity check before pushing
- **service_update** – optional - bool - update the service
- **service_config** – json of service - a service that have config from the main service if wanted
- **slots** – optional - list of slots PackageSlot of the package
- **requirements** – requirements - list of package requirements

Returns

revisions(*package: Optional[*dtlpy.entities.package.Package*] = None*, *package_id=None*)

Get package revisions history

Parameters

- **package** – Package entity
- **package_id** – package id

test_local_package(*cwd=None*, *concurrency=None*, *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*)

Test local package :param cwd: str - path to the file :param concurrency: int -the concurrency of the test :param package: entities.package :param module_name: str - module name :param function_name: str - function name :param class_name: str - class name :param entry_point: str - the file to run like main.py :param mock_file_path: str - the mock file that have the inputs :return:

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

Update Package changes to platform :param package: :param revision_increment: optional - str - version bumping method - major/minor/patch - default = None :return: Package entity

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

clone_git(*codebase*, *local_path*)

Parameters

- **codebase** –
- **local_path** –

get(*codebase_name=None*, *codebase_id=None*, *version=None*)

Get a Codebase object :param codebase_name: optional - search by name :param codebase_id: optional - search by id :param version: codebase version. default is latest. options: "all", "latest" or ver number - "10" :return: Codebase object

static get_current_version(*all_versions_pages*, *zip_md*)

Parameters

- **all_versions_pages** –
- **zip_md** –

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

List all code bases :return: Paged entity

list_versions(*codebase_name*)

List all codebase versions

Parameters **codebase_name** – code base name

Returns list of versions

pack(*directory*, *name=None*, *description=''*)

Zip a local code directory and post to codebases :param directory: local directory to pack :param name: codebase name :param description: codebase description :return: Codebase object

pull_git(*codebase*, *local_path*)

Parameters

- **codebase** –
- **local_path** –

unpack(*codebase: Optional[dtlpy.entities.codebase.Codebase] = None*, *codebase_name=None*, *codebase_id=None*, *local_path=None*, *version=None*)

Unpack codebase locally. Download source code and unzip :param codebase: *dl.Codebase* object :param codebase_name: search by name :param codebase_id: search by id :param local_path: local path to save codebase :param version: codebase version to unpack. default - latest :return: String (dirpath)

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*)

Parameters **until_completed** –

2.9.1 Bots

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

Bases: `object`

Bots repository

create(*name*, *return_credentials: bool = False*)

Create a new Bot :param name: :param return_credentials: with True well return the password when create :return: Bot object

delete(*bot_id=None*, *bot_email=None*)

Delete a Bot :param bot_id: bot id to delete :param bot_email: bot email to delete :return: True

get(*bot_email=None*, *bot_id=None*, *bot_name=None*)

Get a Bot object :param bot_email: get bot by email :param bot_id: get bot by id :param bot_name: get bot by name :return: Bot object

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

Get project's bots list. :return: List of Bots objects

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

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

Inputs for all types:

Parameters

- **service_id** – Id of services to be triggered
- **trigger_type** – can be cron or event. use enum dl.TriggerType for the full list
- **name** – name of the trigger
- **webhook_id** – id for webhook to be called
- **function_name** – the function name to be called when triggered. must be defined in the package
- **project_id** – project id where trigger will work
- **active** – optional - True/False, default = True

Inputs for event trigger: :param filters: optional - Item/Annotation metadata filters, default = none :param resource: optional - Dataset/Item/Annotation/ItemStatus, default = Item :param actions: optional - Created/Updated/Deleted, default = create :param execution_mode: how many time trigger should be activate. 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 cron: cron spec specifying when it should run. more information: https://en.wikipedia.org/wiki/Cron :param pipeline_id: Id of pipeline to be triggered :param pipeline: pipeline entity to be triggered :param 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

delete(*trigger_id=None*, *trigger_name=None*)

Delete Trigger object

Parameters

- **trigger_id** –
- **trigger_name** –

Returns True

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

Get Trigger object :param trigger_id: :param trigger_name: :return: Trigger object

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

List project packages :param filters: :return:

name_validation(*name: str*)

Parameters **name** –

resource_information(*resource*, *resource_type*, *action='Created'*)

return which function should run on a item (based on global triggers)

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*

Parameters **trigger** – Trigger entity

Returns Trigger entity

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

create(*service_id=None*, *execution_input=None*, *function_name=None*, *resource: Optional[dtlpy.entities.package_function.PackageInputType] = None*, *item_id=None*, *dataset_id=None*, *annotation_id=None*, *project_id=None*, *sync=False*, *stream_logs=False*, *return_output=False*, *return_curl_only=False*, *timeout=None*) → *dtlpy.entities.execution.Execution*

Execute a function on an existing service

Parameters

- **service_id** – service id to execute on
- **execution_input** – input dictionary or list of FunctionIO entities
- **function_name** – function name to run
- **resource** – input type.
- **item_id** – optional - input to function
- **dataset_id** – optional - input to function
- **annotation_id** – optional - input to function
- **project_id** – resource's project
- **sync** – wait for function to end
- **stream_logs** – prints logs of the new execution. only works with sync=True
- **return_output** – if True and sync is True - will return the output directly
- **return_curl_only** – return the cURL of the creation WITHOUT actually do it
- **timeout** – int, seconds to wait until TimeoutError is raised. if <=0 - wait until done - by default wait take the service timeout

Returns

get(*execution_id=None*, *sync=False*) → *dtlpy.entities.execution.Execution*

Get Service execution object

Parameters

- **execution_id** –
- **sync** – wait for the execution to finish

Returns Service execution object

increment(*execution: dtlpy.entities.execution.Execution*)

Increment attempts :param execution: :return: int

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

List service executions :param filters: dl.Filters entity to filters items :return:

logs(*execution_id*, *follow=True*, *until_completed=True*)
executions logs :param execution_id: :param follow: :param until_completed: :return: 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

Parameters

- **execution_id** –
- **status** – ExecutionStatus
- **percent_complete** –
- **message** –
- **output** –
- **service_version** –

Returns

rerun(*execution:* dtlpy.entities.execution.Execution, *sync: bool = False*)
Increment attempts :param execution: :param sync: :return: int

terminate(*execution:* dtlpy.entities.execution.Execution)
Terminate Execution :param execution: :return:

update(*execution:* dtlpy.entities.execution.Execution) → *dtlpy.entities.execution.Execution*
Update execution changes to platform :param execution: execution entity :return: execution entity

wait(*execution_id*, *timeout=None*)
Get Service execution object

Parameters

- **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

2.12 Pipelines

class Pipelines(*client_api: dtlpy.services.api_client.ApiClient*, *project: Optional[*dtlpy.entities.project.Project*] = None*)
Bases: `object`

Pipelines Repository

create(*name=None*, *project_id=None*, *pipeline_json=None*) → *dtlpy.entities.pipeline.Pipeline*
Create a new pipeline :param name: str - pipeline name :param project_id: str - project id :param pipeline_json: dict - json contain the pipeline fields :return: Pipeline object

delete(*pipeline: Optional[*dtlpy.entities.pipeline.Pipeline*] = None*, *pipeline_name=None*, *pipeline_id=None*)
Delete Pipeline object

Parameters

- **pipeline** –
- **pipeline_name** –
- **pipeline_id** –

Returns True

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 execute :param pipeline: entities.Pipeline object :param pipeline_id: pipeline id :param pipeline_name: pipeline name :param execution_input: list of the dl.FunctionIO or dict of pipeline input - example {'item': 'item_id'} :return: entities.PipelineExecution object

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

Get Pipeline object

Parameters

- **pipeline_name** – str
- **pipeline_id** – str
- **fetch** – optional - fetch entity from platform, default taken from cookie

Returns Pipeline object

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

install a pipeline :param pipeline: :return: Composition object

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

List project pipelines :param filters: :param project_id: :return:

open_in_web(*pipeline=None*, *pipeline_id=None*, *pipeline_name=None*)

Parameters

- **pipeline** –
- **pipeline_id** –
- **pipeline_name** –

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

pause a pipeline :param pipeline: :return: Composition object

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

Update pipeline changes to platform

Parameters **pipeline** –

Returns pipeline entity

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

create(*pipeline_id: Optional[str] = None*, *execution_input=None*)

execute a pipeline and return the execute :param pipeline_id: pipeline id :param execution_input: list of the dl.FunctionIO or dict of pipeline input - example {'item': 'item_id'} :return: entities.PipelineExecution object

get(*pipeline_execution_id: str*, *pipeline_id: Optional[str] = None*) → *dtlpy.entities.pipeline.Pipeline*

Get Pipeline Execution object

Parameters

- **pipeline_execution_id** – str
- **pipeline_id** – str

Returns Pipeline object

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

List project pipeline executions :param filters: :return:

2.13 General Commands

class Commands(*client_api: dtlpy.services.api_client.ApiClient*)

Bases: `object`

Service Commands repository

abort(*command_id*)

Abort Command

:param command_id :return:

get(*command_id=None*, *url=None*) → *dtlpy.entities.command.Command*

Get Service command object

Parameters

- **command_id** –
- **url** – command url

Returns Command object

list()

List of commands :return:

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

CHAPTER

THREE

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 member to the Organization

Returns True

delete_member(*user_id: str, sure: bool = False, really: bool = False*)

delete member from the Organization

Returns True

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

list_members(*role: Optional[*dtlpy.entities.organization.MemberOrgRole*] = None*)

list all organization members

open_in_web()

Open the organizations in web platform

to_json()

Returns platform _json format of object

Returns platform json format of object

update(*plan: str*)
Update the Organization

Returns Organization object

update_member(*email: str*, *role:* dtlpy.entities.organization.MemberOrgRole = *MemberOrgRole.MEMBER*)
Update the member role

Returns True

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 :param sure: are you sure you want to delete? :param really: really really? :return: True

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

update(*new_name: str*)
Update the integrations name :param 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

checkout()
Checkout the project

Returns

delete(*sure=False*, *really=False*)
Delete the project forever!

Parameters

- **sure** – are you sure you want to delete?
- **really** – really really?

Returns True

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

open_in_web()
Open the project in web platform

Returns

to_json()
Returns platform _json format of object

Returns platform json format of object

update(*system_metadata=False*)
Update the project

Returns Project object

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** – _json response from host
- **project** – project entity
- **client_api** – ApiClient entity
- **users** – Users repository

Returns User object

`to_json()`
Returns platform _json format of object

Returns platform json format of object

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** –
- **children** –
- **attributes** –
- **display_label** –
- **label** –
- **recipe_id** – optional
- **ontology_id** – optional
- **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** –
- **ontology_id** – optional
- **recipe_id** – optional

Returns label entities

`checkout()`
Checkout the dataset

Returns

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** – 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

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

Delete a dataset forever!

Parameters

- **sure** – are you sure you want to delete?
- **really** – really really?

Returns

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*)

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** – 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(dl.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

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*)

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** – 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 - True, if semantic annotations should 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

Returns *List* of local_path per each downloaded item

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 thc convert to specific model structure

Parameters

- **partition** – *dl.SnapshotPartitionType* name of the partition
- **local_path** – local path directory to download the data
- **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** – _json response from host
- **client_api** – ApiClient entity
- **datasets** – Datasets repository
- **is_fetched** – is Entity fetched from Platform

Returns Dataset object

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** – 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

Returns

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** – dl.entities.Filters to add the specific partitions constraint to

Returns dl.PagedEntities

set_readonly(*state: bool*)

Set dataset readonly mode :param state: :return:

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

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

Param recipe_id

Param recipe

Returns

sync(*wait=True*)

Sync dataset with external storage :param wait: wait the command to finish :return:

to_json()

Returns platform _json format of object

Returns platform json format of object

`update`(*system_metadata=False*)

Update dataset field

Parameters `system_metadata` – bool - True, if you want to change metadata system

Returns

`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` –
- `color` –
- `children` –
- `attributes` –
- `display_label` –
- `label` –
- `recipe_id` – optional
- `ontology_id` – optional
- `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` –
- `ontology_id` – optional
- `recipe_id` – optional
- `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='/'*)

Upload annotations to dataset.

Parameters

- `local_path` – str - local folder where the annotations files is.
- `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

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 :return:

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

class ExternalStorage(*value*)

Bases: `str`, `enum.Enum`

An enumeration.

3.4 Item

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 :param dst_dataset_id: destination dataset id :param remote_filepath: complete filepath :param metadata: new metadata to add :param with_annotations: clone annotations :param with_metadata: clone metadata :param with_task_annotations_status: clone task annotations status :param allow_many: *bool* if True use multiple clones in single dataset is allowed, (default=False) :param wait: wait the command to finish

Returns Item

delete()

Delete item from platform :return: 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*)

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

Returns Output (list)

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

Build an item entity object from a json :param project: project entity :param _json: _json response from host :param dataset: dataset in which the annotation's item is located :param client_api: ApiClient entity :param is_fetched: is Entity fetched from Platform :return: Item object

move(*new_path*)

Move item from one folder to another in Platform If the directory doesn't exist it will be created :param new_path: new full path to move item to. :return: 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 :return: platform json format of object

update(*system_metadata=False*)

Update items metadata :param system_metadata: bool - True, if you want to change metadata system :return: 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** – frame number
- **fixed** – is fixed
- **object_visible** – does the annotated object is visible

Returns annotation object

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** – first frame number
- **end_frame_num** – last frame number
- **start_time** – starting time for video
- **end_time** – ending time for video
- **fixed** – is fixed
- **object_visible** – does the annotated object is visible

Returns annotation object

delete()

Remove an annotation from item :return: True

download(*filepath*, *annotation_format:* dtlpy.entities.annotation.ViewAnnotationOptions = *ViewAnnotationOptions.MASK*, *height=None*, *width=None*, *thickness=1*, *with_text=False*, *alpha=None*)

Save annotation to file :param filepath: local path to where annotation will be downloaded to :param annotation_format: options: list(dl.ViewAnnotationOptions) :param height: image height :param width: image width :param thickness: thickness :param with_text: get mask with text :param alpha: opacity value [0 1], default 1 :return: filepath

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 :param _json: platform json :param item: item :param client_api: ApiClient entity :param annotations: :param is_video: :param fps: :param item_metadata: :param dataset :param is_audio: :return: annotation object

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** – item to annotate
- **annotation_definition** – annotation type object
- **object_id** – object_id
- **automated** – is automated
- **metadata** – metadata
- **frame_num** – optional - first frame number if video annotation
- **parent_id** – add parent annotation ID
- **start_time** – optional - start time if video annotation
- **item_height** – annotation item's height
- **item_width** – annotation item's width

Returns annotation object

set_frame(*frame*)
Set annotation to frame state :param frame: frame number :return: True

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 :param image: empty or image to draw on :param thickness: line thickness :param with_text: add label to annotation :param height: height :param width: width :param annotation_format: list(dl.ViewAnnotationOptions) :param color: optional - color tuple :param label_instance_dict: the instance labels :param alpha: opacity value [0 1], default 1 :return: list or single ndarray of the annotations

to_json()
Convert annotation object to a platform json representation :return: platform json

update(*system_metadata=False*)
Update an existing annotation in host. :param system_metadata: True, if you want to change metadata system

Returns Annotation object

update_status(*status:* dtlpy.entities.annotation.AnnotationStatus = *AnnotationStatus.ISSUE*)
Open an issue on the annotation

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

Returns Annotation object or None

upload()
Create a new annotation in host :return:

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

An enumeration.

class AnnotationType(*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 :param annotation: annotation :param _json: annotation type object - must be same type as annotation :param fps: frame number :return: FrameAnnotation object

classmethod new(*annotation*, *annotation_definition*, *frame_num*, *fixed*, *object_visible=True*)
new frame state to annotation :param annotation: annotation :param annotation_definition: annotation type object - must be same type as annotation :param frame_num: frame number :param fixed: is fixed :param object_visible: does the annotated object is visible :return: 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(dl.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*)

Parameters **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 thw 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

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 "angel"

classmethod from_segmentation(*mask*, *label*, *attributes=None*)

Convert binary mask to Polygon

Parameters

- **mask** – binary mask (0,1)
- **label** – annotation label
- **attributes** – annotations list of attributes

Returns Box annotations list to each separated segmentation

show(*image*, *thickness*, *with_text*, *height*, *width*, *annotation_format*, *color*, *alpha=1*)

Show annotation as ndarray :param image: empty or image to draw on :param thickness: :param with_text: not required :param height: item height :param width: item width :param annotation_format: options: list(dl.ViewAnnotationOptions) :param color: color :param alpha: opacity value [0 1], default 1 :return: ndarray

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

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

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

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 :param field: Metadata field / attribute :param values: field values :param operator: optional - in, gt, lt, eq, ne :param method: Optional - or/and :return:

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

join a query to the filter :param field: :param values: :param operator: optional - in, gt, lt, eq, ne :param method: optional - str - FiltersMethod.AND, FiltersMethod.OR

generate_url_query_params(*url*)

:param url"

has_field(*field*)

Parameters **field** –

pop(*field*)

Parameters **field** –

pop_join(*field*)

Parameters **field** –

prepare(*operation=None*, *update=None*, *query_only=False*, *system_update=None*, *system_metadata=False*)

To dictionary for platform call :param operation: :param update: :param query_only: :param system_update: :param system_metadata: True, if you want to change metadata system :return: dict

sort_by(*field*, *value:* dtlpy.entities.filters.FiltersOrderByDirection *= FiltersOrderByDirection.ASCENDING*)

Parameters

- **field** –
- **value** – FiltersOrderByDirection.ASCENDING, FiltersOrderByDirection.DESCENDING

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()
Delete recipe from platform

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

`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` –
- `add` –
- `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

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** –

- **add** –
- **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*)

Parameters

- **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** –
- **wait** – wait the command to finish

Returns

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

Parameters

- **assignment_name** – assignment name
- **assignee_id** – list of assignee for the assignment
- **items** – items list for the assignment

- **filters** – Filters entity or a dictionary containing filters parameters

Returns

create_qa_task(*due_date*, *assignee_ids*)

Parameters

- **due_date** –
- **assignee_ids** –

delete(*wait=True*)

Delete task from platform :param wait: wait the command to finish :return: True

get_items(*filters=None*)

Parameters **filters** –

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

update(*system_metadata=False*)

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*)

Parameters

- **dataset** – dataset entity

- **filters** – Filters entity or a dictionary containing filters parameters

Returns

open_in_web()
Open the assignment in web platform

Returns

reassign(*assignee_id*, *wait=True*)
Reassign an assignment :param assignee_id: :param wait: wait the command to finish :return: Assignment object

redistribute(*workload*, *wait=True*)
Redistribute an assignment :param workload: :param wait: wait the command to finish :return: Assignment object

set_status(*status: str*, *operation: str*, *item_id: str*)
Update item status within task

Parameters

- **status** – str - string the describes the status
- **operation** – str - 'create' or 'delete'
- **item_id** – str

:return : Boolean

to_json()
Returns platform _json format of object

Returns platform json format of object

update(*system_metadata=False*)

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

Returns

class Workload(*workload: list = NOTHING*)
Bases: `object`

Workload object

add(*assignee_id*)
add a assignee :param 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

- **max_attempts** – Maximum execution retries in-case of a service reset
- **on_reset** –
- **drain_time** –
- **execution_timeout** –
- **run_execution_as_process** –
- **module_name** –
- **pod_type** –
- **bot** –
- **verify** –
- **force** – optional - terminate old replicas immediately
- **agent_versions** –
- **sdk_version** –
- **runtime** –
- **init_input** –
- **revision** –
- **service_name** –

Returns

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** –
- **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** – PackageCode object - defines how to store the package code
- **checkout** – save package to local checkout
- **src_path** – location of pacjage 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

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 KubernetesAutuscalerType(*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

@rtype: List[UserSetting] @param project_id: str @param task_id: str @param dataset_id: str @param org_id: str @param user_email: str @param slots: List[PackageSlot] @param role: Role @param prevent_override: bool @param visible: bool @param icon: str @param kwargs: system @return: 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*)

Parameters

- **_json** – platform json
- **client_api** – ApiClient entity
- **package** –
- **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 :param size: :param checkpoint: :param start: iso format time :param end: iso format time :param follow: filters :param text: :param execution_id: :param function_name: :param replica_id: :param system: :param view: :param until_completed: :return: Service entity

open_in_web()

Open the service in web platform

Returns

pause()

Returns

resume()

Returns

status()

Get Service status

Returns True

update(*force=False*)

Update Service changes to platform :param force: :return: 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

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

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*)

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

update()

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*)

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

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*)

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

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** – ExecutionStatus
- **percent_complete** –
- **message** –
- **output** –
- **service_version** –

Returns

rerun()
Re-run

Returns

terminate()
Terminate execution

Returns

to_json()
Returns platform _json format of object

Returns platform json format of object

update()
Update execution changes to platform :return: execution entity

wait()
Wait for execution

Returns

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 :param execution_input: list of the dl.FunctionIO or dict of pipeline input - example {'item': 'item_id'} :return: 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

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

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*)

Parameters

- **page_offset** –
- **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*)

Parameters

- **page_offset** –
- **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 Boolean

to_json()

Returns platform _json format of object

Returns platform json format of object

wait(*timeout=0*, *step=5*)

Wait for Command to finish

Parameters

- **timeout** – int, seconds to wait until TimeoutError is raised. if 0 - wait until done
- **step** – 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

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex

PYTHON MODULE INDEX

d

INDEX

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W