Labelbox Python API reference

Release 3.11.0

Labelbox

Aug 24, 2022

CONTENTS:

CHAPTER

ONE

CLIENT

class labelbox.client.**Client**(*api_key=None*, *endpoint='https://api.labelbox.com/graphql'*, *enable_experimental=False*, *app_url='https://app.labelbox.com'*)

Bases: `object`

A Labelbox client.

Contains info necessary for connecting to a Labelbox server (URL, authentication key). Provides functions for querying and creating top-level data objects (Projects, Datasets).

__init__(*api_key=None*, *endpoint='https://api.labelbox.com/graphql'*, *enable_experimental=False*, *app_url='https://app.labelbox.com'*)

Creates and initializes a Labelbox Client.

Logging is defaulted to level WARNING. To receive more verbose output to console, update *logging.level* to the appropriate level.

```
>>> logging.basicConfig(level = logging.INFO)
>>> client = Client("<APIKEY>")
```

Parameters

- **api_key** (*str*) – API key. If None, the key is obtained from the "LABELBOX_API_KEY" environment variable.
- **endpoint** (*str*) – URL of the Labelbox server to connect to.
- **enable_experimental** (*bool*) – Indicates whether or not to use experimental features
- **app_url** (*str*) – host url for all links to the web app

Raises *labelbox.exceptions.AuthenticationError* – If no *api_key* is provided as an argument or via the environment variable.

create_dataset(*iam_integration='DEFAULT'*, ***kwargs*) → *labelbox.schema.dataset.Dataset*

Creates a Dataset object on the server.

Attribute values are passed as keyword arguments.

```
>>> project = client.get_project("<project_uid>")
>>> dataset = client.create_dataset(name="<dataset_name>", projects=project)
```

Parameters

- **iam_integration** (*IAMIntegration*) – Uses the default integration. Optionally specify another integration or set as None to not use delegated access

- ****kwargs** – Keyword arguments with Dataset attribute values.

Returns A new Dataset object.

Raises *InvalidAttributeError* – If the Dataset type does not contain any of the attribute names given in kwargs.

`create_feature_schema`(*normalized*)

Creates a feature schema from normalized data.

```
>>> normalized = {'tool': 'polygon',  'name': 'cat', 'color': 'black'}
>>> feature_schema = client.create_feature_schema(normalized)
```

Or use the Tool or Classification objects. It is especially useful for complex tools.

```
>>> normalized = Tool(tool=Tool.Type.BBOX, name="cat", color = 'black').
↪asdict()
>>> feature_schema = client.create_feature_schema(normalized)
```

Subclasses are also supported

```
>>> normalized =  Tool(
        tool=Tool.Type.SEGMENTATION,
        name="cat",
        classifications=[
            Classification(
                class_type=Classification.Type.TEXT,
                instructions="name"
            )
        ]
    )
>>> feature_schema = client.create_feature_schema(normalized)
```

More details can be found here: https://github.com/Labelbox/labelbox-python/blob/develop/examples/basics/ontologies.ipynb

Parameters **normalized** (*dict*) – A normalized tool or classification payload. See above for details

Returns The created FeatureSchema.

`create_model`(*name*, *ontology_id*) → *labelbox.schema.model.Model*

Creates a Model object on the server.

```
>>> model = client.create_model(<model_name>, <ontology_id>)
```

Parameters

- **name** (*string*) – Name of the model
- **ontology_id** (*string*) – ID of the related ontology

Returns A new Model object.

Raises *InvalidAttributeError* – If the Model type does not contain any of the attribute names given in kwargs.

create_ontology(*name*, *normalized*) → *labelbox.schema.ontology.Ontology*

Creates an ontology from normalized data

```
>>> normalized = {"tools" : [{'tool': 'polygon',  'name': 'cat', 'color':
↪'black'}], "classifications" : []}
>>> ontology = client.create_ontology("ontology-name", normalized)
```

Or use the ontology builder. It is especially useful for complex ontologies

```
>>> normalized = OntologyBuilder(tools=[Tool(tool=Tool.Type.BBOX, name="cat
↪", color = 'black')]).asdict()
>>> ontology = client.create_ontology("ontology-name", normalized)
```

To reuse existing feature schemas, use *create_ontology_from_feature_schemas()* More details can be found here:

https://github.com/Labelbox/labelbox-python/blob/develop/examples/basics/ontologies.ipynb

Parameters

- **name** (*str*) – Name of the ontology
- **normalized** (*dict*) – A normalized ontology payload. See above for details.

Returns The created Ontology

create_ontology_from_feature_schemas(*name*, *feature_schema_ids*) → *labelbox.schema.ontology.Ontology*

Creates an ontology from a list of feature schema ids

Parameters

- **name** (*str*) – Name of the ontology
- **feature_schema_ids** (*List[str]*) – List of feature schema ids corresponding to top level tools and classifications to include in the ontology

Returns The created Ontology

create_project(***kwargs*) → *labelbox.schema.project.Project*

Creates a Project object on the server.

Attribute values are passed as keyword arguments.

```
>>> project = client.create_project(name="<project_name>", description="
↪<project_description>")
```

Parameters ****kwargs** – Keyword arguments with Project attribute values.

Returns A new Project object.

Raises ***InvalidAttributeError*** – If the Project type does not contain any of the attribute names given in kwargs.

execute(*query=None*, *params=None*, *data=None*, *files=None*, *timeout=30.0*, *experimental=False*)

Sends a request to the server for the execution of the given query.

Checks the response for errors and wraps errors in appropriate *labelbox.exceptions.LabelboxError* subtypes.

Parameters

- **query** (*str*) – The query to execute.
- **params** (*dict*) – Query parameters referenced within the query.
- **data** (*str*) – json string containing the query to execute
- **files** (*dict*) – file arguments for request
- **timeout** (*float*) – Max allowed time for query execution, in seconds.

Returns dict, parsed JSON response.

Raises

- *labelbox.exceptions.AuthenticationError* – If authentication failed.
- *labelbox.exceptions.InvalidQueryError* – If *query* is not syntactically or semantically valid (checked server-side).
- *labelbox.exceptions.ApiLimitError* – If the server API limit was exceeded. See "How to import data" in the online documentation to see API limits.
- *labelbox.exceptions.TimeoutError* – If response was not received in *timeout* seconds.
- *labelbox.exceptions.NetworkError* – If an unknown error occurred most likely due to connection issues.
- *labelbox.exceptions.LabelboxError* – If an unknown error of any kind occurred.
- **ValueError** – If query and data are both None.

get_data_row(*data_row_id*)

Returns returns a single data row given the data row id

Return type *DataRow*

get_data_row_ids_for_external_ids(*external_ids: List[str]*) → Dict[str, List[str]]

Returns a list of data row ids for a list of external ids. There is a max of 1500 items returned at a time.

Parameters **external_ids** – List of external ids to fetch data row ids for

Returns A dict of external ids as keys and values as a list of data row ids that correspond to that external id.

get_data_row_metadata_ontology() → *labelbox.schema.data_row_metadata.DataRowMetadataOntology*

Returns The ontology for Data Row Metadata for an organization

Return type *DataRowMetadataOntology*

get_dataset(*dataset_id*) → *labelbox.schema.dataset.Dataset*

Gets a single Dataset with the given ID.

```
>>> dataset = client.get_dataset("<dataset_id>")
```

Parameters **dataset_id** (*str*) – Unique ID of the Dataset.

Returns The sought Dataset.

Raises *labelbox.exceptions.ResourceNotFoundError* – If there is no Dataset with the given ID.

get_datasets(*where=None*) → List[*labelbox.schema.dataset.Dataset*]

Fetches one or more datasets.

```
>>> datasets = client.get_datasets(where=(Dataset.name == "<dataset_name>") &␣
↪(Dataset.description == "<dataset_description>"))
```

Parameters **where** (*Comparison, LogicalOperation or None*) – The *where* clause for filtering.

Returns An iterable of Datasets (typically a PaginatedCollection).

get_feature_schema(*feature_schema_id*)

Fetches a feature schema. Only supports top level feature schemas.

Parameters **feature_schema_id** (*str*) – The id of the feature schema to query for

Returns FeatureSchema

get_feature_schemas(*name_contains*) → *labelbox.pagination.PaginatedCollection*

Fetches top level feature schemas with names that match the *name_contains* string

Parameters **name_contains** (*str*) – the string to search top level feature schema names by

Returns PaginatedCollection of FeatureSchemas with names that match *name_contains*

get_labeling_frontends(*where=None*) → List[*labelbox.schema.labeling_frontend.LabelingFrontend*]

Fetches all the labeling frontends.

```
>>> frontend = client.get_labeling_frontends(where=LabelingFrontend.name ==
↪"Editor")
```

Parameters **where** (*Comparison, LogicalOperation or None*) – The *where* clause for filtering.

Returns An iterable of LabelingFrontends (typically a PaginatedCollection).

get_model(*model_id*) → *labelbox.schema.model.Model*

Gets a single Model with the given ID.

```
>>> model = client.get_model("<model_id>")
```

Parameters **model_id** (*str*) – Unique ID of the Model.

Returns The sought Model.

Raises ***labelbox.exceptions.ResourceNotFoundError*** – If there is no Model with the given ID.

get_model_run(*model_run_id: str*) → *labelbox.schema.model_run.ModelRun*

Gets a single ModelRun with the given ID.

```
>>> model_run = client.get_model_run("<model_run_id>")
```

Parameters **model_run_id** (*str*) – Unique ID of the ModelRun.

Returns A ModelRun object.

get_models(*where=None*) → List[*labelbox.schema.model.Model*]

Fetches all the models the user has access to.

```
>>> models = client.get_models(where=(Model.name == "<model_name>"))
```

Parameters **where** (*Comparison, LogicalOperation or None*) – The *where* clause for filtering.

Returns An iterable of Models (typically a PaginatedCollection).

get_ontologies(*name_contains*) → *labelbox.pagination.PaginatedCollection*

Fetches all ontologies with names that match the name_contains string.

Parameters **name_contains** (*str*) – the string to search ontology names by

Returns PaginatedCollection of Ontologies with names that match *name_contains*

get_ontology(*ontology_id*) → *labelbox.schema.ontology.Ontology*

Fetches an Ontology by id.

Parameters **ontology_id** (*str*) – The id of the ontology to query for

Returns Ontology

get_organization() → *labelbox.schema.organization.Organization*

Gets the Organization DB object of the current user.

```
>>> organization = client.get_organization()
```

get_project(*project_id*)

Gets a single Project with the given ID.

```
>>> project = client.get_project("<project_id>")
```

Parameters **project_id** (*str*) – Unique ID of the Project.

Returns The sought Project.

Raises ***labelbox.exceptions.ResourceNotFoundError*** – If there is no Project with the given ID.

get_projects(*where=None*) → List[*labelbox.schema.project.Project*]

Fetches all the projects the user has access to.

```
>>> projects = client.get_projects(where=(Project.name == "<project_name>") &␣
↪(Project.description == "<project_description>"))
```

Parameters **where** (*Comparison, LogicalOperation or None*) – The *where* clause for filtering.

Returns An iterable of Projects (typically a PaginatedCollection).

get_roles() → List[labelbox.schema.role.Role]

Returns Provides information on available roles within an organization. Roles are used for user management.

Return type Roles

get_user() → *labelbox.schema.user.User*

Gets the current User database object.

```
>>> user = client.get_user()
```

CHAPTER

TWO

ASSETATTACHMENT

class labelbox.schema.asset_attachment.**AssetAttachment**(*client*, *field_values*)

Bases: labelbox.orm.db_object.DbObject

Asset attachment provides extra context about an asset while labeling.

attachment_type

IMAGE, VIDEO, TEXT, IMAGE_OVERLAY, or HTML

Type str

attachment_value

URL to an external file or a string of text

Type str

class AttachmentType(*value*)

Bases: enum.Enum

An enumeration.

delete() → None

Deletes an attachment on the data row.

CHAPTER

THREE

BENCHMARK

class labelbox.schema.benchmark.**Benchmark**(*client*, *field_values*)

Bases: labelbox.orm.db_object.DbObject

Represents a benchmark label.

The Benchmarks tool works by interspersing data to be labeled, for which there is a benchmark label, to each person labeling. These labeled data are compared against their respective benchmark and an accuracy score between 0 and 100 percent is calculated.

created_at

Type datetime

last_activity

Type datetime

average_agreement

Type float

completed_count

Type int

created_by

ToOne relationship to User

Type Relationship

reference_label

ToOne relationship to Label

Type Relationship

CHAPTER

FOUR

BULKIMPORTREQUEST

class labelbox.schema.bulk_import_request.**BulkImportRequest**(*client*, *field_values*)

Bases: labelbox.orm.db_object.DbObject

Represents the import job when importing annotations.

name

Type str

state

FAILED, RUNNING, or FINISHED (Refers to the whole import job)

Type Enum

input_file_url

URL to your web-hosted NDJSON file

Type str

error_file_url

NDJSON that contains error messages for failed annotations

Type str

status_file_url

NDJSON that contains status for each annotation

Type str

created_at

UTC timestamp for date BulkImportRequest was created

Type datetime

project

ToOne relationship to Project

Type Relationship

created_by

ToOne relationship to User

Type Relationship

delete() → None

Deletes the import job and also any annotations created by this import.

Returns None

property errors: List[Dict[str, Any]]

Errors for each individual annotation uploaded. This is a subset of statuses

Returns List of dicts containing error messages. Empty list means there were no errors See *BulkImportRequest.statuses* for more details.

- This information will expire after 24 hours.

property inputs: List[Dict[str, Any]]

Inputs for each individual annotation uploaded. This should match the ndjson annotations that you have uploaded.

Returns Uploaded ndjson.

- This information will expire after 24 hours.

refresh() → None

Synchronizes values of all fields with the database.

property statuses: List[Dict[str, Any]]

Status for each individual annotation uploaded.

Returns A status for each annotation if the upload is done running. See below table for more details

Field	Description
uuid	Specifies the annotation for the status row.
dataRow	JSON object containing the Labelbox data row ID for the annotation.
status	Indicates SUCCESS or FAILURE.
errors	An array of error messages included when status is FAILURE. Each error has a name, message and optional (key might not exist) additional_info.

- This information will expire after 24 hours.

wait_until_done(*sleep_time_seconds: int = 5*) → None

Blocks import job until certain conditions are met.

Blocks until the BulkImportRequest.state changes either to *BulkImportRequestState.FINISHED* or *BulkImportRequestState.FAILED*, periodically refreshing object's state.

Parameters **sleep_time_seconds** (*str*) – a time to block between subsequent API calls

labelbox.schema.bulk_import_request.**get_mal_schemas**(*ontology*)

Converts a project ontology to a dict for easier lookup during ndjson validation

Parameters **ontology** (Ontology) –

Returns Useful for looking up a tool from a given feature schema id

Return type Dict

labelbox.schema.bulk_import_request.**parse_classification**(*tool*)

Parses a classification from an ontology. Only radio, checklist, and text are supported for mal

Parameters **tool** (*dict*) –

Returns dict

CHAPTER

FIVE

DATAROW

class labelbox.schema.data_row.**DataRow**(**args*, ***kwargs*)

Bases: labelbox.orm.db_object.DbObject, labelbox.orm.db_object.Updateable, labelbox.orm.db_object.BulkDeletable

Internal Labelbox representation of a single piece of data (e.g. image, video, text).

external_id

User-generated file name or identifier

Type str

row_data

Paths to local files are uploaded to Labelbox's server. Otherwise, it's treated as an external URL.

Type str

updated_at

Type datetime

created_at

Type datetime

media_attributes

generated media attributes for the datarow

Type dict

metadata_fields

metadata associated with the datarow

Type list

dataset

ToOne relationship to Dataset

Type Relationship

created_by

ToOne relationship to User

Type Relationship

organization

ToOne relationship to Organization

Type Relationship

labels

ToMany relationship to Label

Type Relationship

attachments

Type Relationship

static bulk_delete(*data_rows*) → None

Deletes all the given DataRows.

Parameters **data_rows** (*list of DataRow*) – The DataRows to delete.

create_attachment(*attachment_type*, *attachment_value*) → *AssetAttachment*

Adds an AssetAttachment to a DataRow. Labelers can view these attachments while labeling.

```
>>> datarow.create_attachment("TEXT", "This is a text message")
```

Parameters

- **attachment_type** (*str*) – Asset attachment type, must be one of: VIDEO, IMAGE, TEXT, IMAGE_OVERLAY (AssetAttachment.AttachmentType)
- **attachment_value** (*str*) – Asset attachment value.

Returns *AssetAttachment* DB object.

Raises **ValueError** – asset_type must be one of the supported types.

CHAPTER

SIX

DATASET

class labelbox.schema.dataset.**Dataset**(*client*, *field_values*)

Bases: labelbox.orm.db_object.DbObject, labelbox.orm.db_object.Updateable, labelbox.orm.db_object.Deletable

A Dataset is a collection of DataRows.

name

Type str

description

Type str

updated_at

Type datetime

created_at

Type datetime

row_count

The number of rows in the dataset. Fetch the dataset again to update since this is cached.

Type int

projects

ToMany relationship to Project

Type Relationship

data_rows

ToMany relationship to DataRow

Type Relationship

created_by

ToOne relationship to User

Type Relationship

organization

ToOne relationship to Organization

Type Relationship

create_data_row(*items=None*, ***kwargs*) → *DataRow*

Creates a single DataRow belonging to this dataset.

```
>>> dataset.create_data_row(row_data="http://my_site.com/photos/img_01.jpg")
```

Parameters

- **items** – Dictionary containing new *DataRow* data. At a minimum, must contain *row_data* or *DataRow.row_data*.
- ****kwargs** – Key-value arguments containing new *DataRow* data. At a minimum, must contain *row_data*.

Raises

- ***InvalidQueryError*** – If both dictionary and *kwargs* are provided as inputs
- ***InvalidQueryError*** – If *DataRow.row_data* field value is not provided in *kwargs*.
- ***InvalidAttributeError*** – in case the DB object type does not contain any of the field names given in *kwargs*.

create_data_rows(*items*) → Union[*Task*, List[Any]]

Asynchronously bulk upload data rows

Use this instead of *Dataset.create_data_rows_sync* uploads for batches that contain more than 1000 data rows.

Parameters **items** (*iterable of (dict or str)*) – See the docstring for *Dataset._create_descriptor_file* for more information

Returns Task representing the data import on the server side. The Task can be used for inspecting task progress and waiting until it's done.

Raises

- ***InvalidQueryError*** – If the *items* parameter does not conform to the specification above or if the server did not accept the DataRow creation request (unknown reason).
- ***ResourceNotFoundError*** – If unable to retrieve the Task for the import process. This could imply that the import failed.
- ***InvalidAttributeError*** – If there are fields in *items* not valid for a DataRow.
- **ValueError** – When the upload parameters are invalid

create_data_rows_sync(*items*) → None

Synchronously bulk upload data rows.

Use this instead of *Dataset.create_data_rows* for smaller batches of data rows that need to be uploaded quickly. Cannot use this for uploads containing more than 1000 data rows. Each data row is also limited to 5 attachments.

Parameters **items** (*iterable of (dict or str)*) – See the docstring for *Dataset._create_descriptor_file* for more information.

Returns None. If the function doesn't raise an exception then the import was successful.

Raises

- ***InvalidQueryError*** – If the *items* parameter does not conform to the specification in Dataset._create_descriptor_file or if the server did not accept the DataRow creation request (unknown reason).

- *InvalidAttributeError* – If there are fields in *items* not valid for a DataRow.
- **ValueError** – When the upload parameters are invalid

data_row_for_external_id(*external_id*) → *DataRow*

Convenience method for getting a single *DataRow* belonging to this *Dataset* that has the given *external_id*.

Parameters **external_id** (*str*) – External ID of the sought *DataRow*.

Returns A single *DataRow* with the given ID.

Raises *labelbox.exceptions.ResourceNotFoundError* – If there is no *DataRow* in this *DataSet* with the given external ID, or if there are multiple *DataRows* for it.

data_rows_for_external_id(*external_id*, *limit=10*) → List[*DataRow*]

Convenience method for getting a multiple *DataRow* belonging to this *Dataset* that has the given *external_id*.

Parameters

- **external_id** (*str*) – External ID of the sought *DataRow*.
- **limit** (*int*) – The maximum number of data rows to return for the given external_id

Returns A list of *DataRow* with the given ID.

Raises *labelbox.exceptions.ResourceNotFoundError* – If there is no *DataRow* in this *DataSet* with the given external ID, or if there are multiple *DataRows* for it.

export_data_rows(*timeout_seconds=120*) → Generator

Returns a generator that produces all data rows that are currently attached to this dataset.

Note: For efficiency, the data are cached for 30 minutes. Newly created data rows will not appear until the end of the cache period.

Parameters **timeout_seconds** (*float*) – Max waiting time, in seconds.

Returns Generator that yields DataRow objects belonging to this dataset.

Raises *LabelboxError* – if the export fails or is unable to download within the specified time.

CHAPTER

SEVEN

LABEL

class labelbox.schema.label.**Label**(**args*, ***kwargs*)

Bases: `labelbox.orm.db_object.DbObject`, `labelbox.orm.db_object.Updateable`, `labelbox.orm.db_object.BulkDeletable`

Label represents an assessment on a DataRow. For example one label could contain 100 bounding boxes (annotations).

label

Type str

seconds_to_label

Type float

agreement

Type float

benchmark_agreement

Type float

is_benchmark_reference

Type bool

project

ToOne relationship to Project

Type Relationship

data_row

ToOne relationship to DataRow

Type Relationship

reviews

ToMany relationship to Review

Type Relationship

created_by

ToOne relationship to User

Type Relationship

static bulk_delete(*labels*) → None

Deletes all the given Labels.

Parameters **labels** (*list of Label*) – The Labels to delete.

create_benchmark() → *Benchmark*

Creates a Benchmark for this Label.

Returns The newly created Benchmark.

create_review(**kwargs*) → *Review*

Creates a Review for this label.

Parameters ****kwargs** – Review attributes. At a minimum, a *Review.score* field value must be provided.

CHAPTER

EIGHT

LABELINGFRONTEND

class labelbox.schema.labeling_frontend.**LabelingFrontend**(*client*, *field_values*)

Bases: labelbox.orm.db_object.DbObject

Label editor.

Represents an HTML / JavaScript UI that is used to generate labels. "Editor" is the default Labeling Frontend that comes in every organization. You can create new labeling frontends for an organization.

name

Type str

description

Type str

iframe_url_path

Type str

projects

ToMany relationship to Project

Type Relationship

CHAPTER

NINE

LABELINGFRONTENDOPTIONS

class labelbox.schema.labeling_frontend.**LabelingFrontendOptions**(*client*, *field_values*)

Bases: labelbox.orm.db_object.DbObject

Label interface options.

customization_options

Type str

project

ToOne relationship to Project

Type Relationship

labeling_frontend

ToOne relationship to LabelingFrontend

Type Relationship

organization

ToOne relationship to Organization

Type Relationship

CHAPTER

TEN

LABELINGPARAMETEROVERRIDE

class labelbox.schema.project.**LabelingParameterOverride**(*client*, *field_values*)

Bases: labelbox.orm.db_object.DbObject

Customizes the order of assets in the label queue.

priority

A prioritization score.

Type int

number_of_labels

Number of times an asset should be labeled.

Type int

CHAPTER

ELEVEN

ONTOLOGY

class labelbox.schema.ontology.**FeatureSchema**(*client*, *field_values*)

Bases: labelbox.orm.db_object.DbObject

labelbox.schema.ontology.**FeatureSchemaId**

alias of labelbox.schema.ontology.ConstrainedStrValue

class labelbox.schema.ontology.**Ontology**(**args*, ***kwargs*)

Bases: labelbox.orm.db_object.DbObject

An ontology specifies which tools and classifications are available to a project. This is read only for now. .. attribute:: name

type str

description

Type str

updated_at

Type datetime

created_at

Type datetime

normalized

Type json

object_schema_count

Type int

classification_schema_count

Type int

projects

ToMany relationship to Project

Type Relationship

created_by

ToOne relationship to User

Type Relationship

classifications() → List[labelbox.schema.ontology.Classification]

Get list of classifications in an Ontology.

tools() → List[labelbox.schema.ontology.Tool]

Get list of tools (AKA objects) in an Ontology.

class labelbox.schema.ontology.**OntologyBuilder**(*tools: typing.List[labelbox.schema.ontology.Tool] = <factory>*, *classifications: typing.List[labelbox.schema.ontology.Classification] = <factory>*)

Bases: object

A class to help create an ontology for a Project. This should be used for making Project ontologies from scratch. OntologyBuilder can also pull from an already existing Project's ontology.

There are no required instantiation arguments.

To create an ontology, use the asdict() method after fully building your ontology within this class, and inserting it into project.setup() as the "labeling_frontend_options" parameter.

Example

builder = OntologyBuilder() ... frontend = list(client.get_labeling_frontends())[0] project.setup(frontend, builder.asdict())

tools

(list)

classifications

(list)

labelbox.schema.ontology.**SchemaId**

alias of labelbox.schema.ontology.ConstrainedStrValue

CHAPTER

TWELVE

ORGANIZATION

class labelbox.schema.organization.**Organization**(**args*, ***kwargs*)

Bases: labelbox.orm.db_object.DbObject

An Organization is a group of Users.

It is associated with data created by Users within that Organization. Typically all Users within an Organization have access to data created by any User in the same Organization.

updated_at

Type datetime

created_at

Type datetime

name

Type str

users

ToMany relationship to User

Type Relationship

projects

ToMany relationship to Project

Type Relationship

webhooks

ToMany relationship to Webhook

Type Relationship

create_resource_tag(*tag: Dict[str, str]*) → labelbox.schema.resource_tag.ResourceTag

Creates a resource tag.

```
>>> tag = {'text': 'tag-1',  'color': 'ffffff'}
```

Parameters **tag** (*dict*) – A resource tag {'text': 'tag-1', 'color': 'fffff'}

Returns The created resource tag.

get_default_iam_integration() → Optional[IAMIntegration]

Returns the default IAM integration for the organization. Will return None if there are no default integrations for the org.

get_iam_integrations() → List[IAMIntegration]

Returns all IAM Integrations for an organization

get_resource_tags() → List[labelbox.schema.resource_tag.ResourceTag]

Returns all resource tags for an organization

invite_limit() → labelbox.schema.invite.InviteLimit

Retrieve invite limits for the org This already accounts for users currently in the org Meaining that *used = users + invites, remaining = limit - (users + invites)*

Returns InviteLimit

invite_user(*email: str*, *role: Role*, *project_roles: Optional[List[ProjectRole]] = None*) → Invite

Invite a new member to the org. This will send the user an email invite

Parameters

- **email** (*str*) – email address of the user to invite
- **role** (*Role*) – Role to assign to the user
- **project_roles** (*Optional[List[ProjectRoles]]*) – List of project roles to assign to the User (if they have a project based org role).

Returns Invite for the user

Notes

1. **Multiple invites can be sent for the same email. This can only be resolved in the UI for now.**
   - Future releases of the SDK will support the ability to query and revoke invites to solve this problem (and/or checking on the backend)
2. Some server side response are unclear (e.g. if the user invites themself *None* is returned which the SDK raises as a *LabelboxError*)

remove_user(*user:* User) → None

Deletes a user from the organization. This cannot be undone without sending another invite.

Parameters **user** (User) – The user to delete from the org

CHAPTER

THIRTEEN

PROJECT

class labelbox.schema.project.**Project**(*client*, *field_values*)

Bases: labelbox.orm.db_object.DbObject, labelbox.orm.db_object.Updateable, labelbox.orm.db_object.Deletable

A Project is a container that includes a labeling frontend, an ontology, datasets and labels.

name

Type str

description

Type str

updated_at

Type datetime

created_at

Type datetime

setup_complete

Type datetime

last_activity_time

Type datetime

auto_audit_number_of_labels

Type int

auto_audit_percentage

Type float

datasets

ToMany relationship to Dataset

Type Relationship

created_by

ToOne relationship to User

Type Relationship

`organization`

ToOne relationship to Organization

Type Relationship

`labeling_frontend`

ToOne relationship to LabelingFrontend

Type Relationship

`labeling_frontend_options`

ToMany relationship to LabelingFrontendOptions

Type Relationship

`labeling_parameter_overrides`

ToMany relationship to LabelingParameterOverride

Type Relationship

`webhooks`

ToMany relationship to Webhook

Type Relationship

`benchmarks`

ToMany relationship to Benchmark

Type Relationship

`ontology`

ToOne relationship to Ontology

Type Relationship

`class QueueMode(value)`

Bases: `enum.Enum`

An enumeration.

`batches() → labelbox.pagination.PaginatedCollection`

Fetch all batches that belong to this project

Returns A `PaginatedCollection of `Batch`es

`bulk_import_requests() → labelbox.pagination.PaginatedCollection`

Returns bulk import request objects which are used in model-assisted labeling. These are returned with the oldest first, and most recent last.

`create_batch(name: str, data_rows: List[str], priority: int = 5)`

Create a new batch for a project. Batches is in Beta and subject to change

Parameters

- **name** – a name for the batch, must be unique within a project
- **data_rows** – Either a list of *DataRows* or Data Row ids
- **priority** – An optional priority for the Data Rows in the Batch. 1 highest -> 5 lowest

enable_model_assisted_labeling(*toggle: bool = True*) → bool

Turns model assisted labeling either on or off based on input

Parameters **toggle** (*bool*) – True or False boolean

Returns True if toggled on or False if toggled off

export_issues(*status=None*) → str

Calls the server-side Issues exporting that returns the URL to that payload.

Parameters **status** (*string*) – valid values: Open, Resolved

Returns URL of the data file with this Project's issues.

export_labels(*download=False*, *timeout_seconds=600*, ***kwargs*) → Optional[Union[str, List[Dict[Any, Any]]]]

Calls the server-side Label exporting that generates a JSON payload, and returns the URL to that payload.

Will only generate a new URL at a max frequency of 30 min.

Parameters

- **download** (*bool*) – Returns the url if False
- **timeout_seconds** (*float*) – Max waiting time, in seconds.
- **start** (*str*) – Earliest date for labels, formatted "YYYY-MM-DD"
- **end** (*str*) – Latest date for labels, formatted "YYYY-MM-DD"

Returns URL of the data file with this Project's labels. If the server didn't generate during the *timeout_seconds* period, None is returned.

export_queued_data_rows(*timeout_seconds=120*) → List[Dict[str, str]]

Returns all data rows that are currently enqueued for this project.

Parameters **timeout_seconds** (*float*) – Max waiting time, in seconds.

Returns Data row fields for all data rows in the queue as json

Raises ***LabelboxError*** – if the export fails or is unable to download within the specified time.

extend_reservations(*queue_type*) → int

Extends all the current reservations for the current user on the given queue type. :param queue_type: Either "LabelingQueue" or "ReviewQueue" :type queue_type: str

Returns int, the number of reservations that were extended.

label_generator(*timeout_seconds=600*, ***kwargs*)

Download text and image annotations, or video annotations.

For a mixture of text/image and video, use project.export_labels()

Returns LabelGenerator for accessing labels

labeler_performance() → *labelbox.pagination.PaginatedCollection*

Returns the labeler performances for this Project.

Returns A PaginatedCollection of LabelerPerformance objects.

labels(*datasets=None*, *order_by=None*) → *labelbox.pagination.PaginatedCollection*

Custom relationship expansion method to support limited filtering.

Parameters

- **datasets** (*iterable of Dataset*) – Optional collection of Datasets whose Labels are sought. If not provided, all Labels in this Project are returned.
- **order_by** (*None or (Field, Field.Order)*) – Ordering clause.

members() → *labelbox.pagination.PaginatedCollection*

Fetch all current members for this project

Returns A `PaginatedCollection of `ProjectMember`s

queue_mode() → *labelbox.schema.project.Project.QueueMode*

Provides the status of if queue mode is enabled in the project.

review_metrics(*net_score*) → int

Returns this Project's review metrics.

Parameters **net_score** (*None or* Review.NetScore) – Indicates desired metric.

Returns int, aggregation count of reviews for given *net_score*.

set_labeling_parameter_overrides(*data*) → bool

Adds labeling parameter overrides to this project.

See information on priority here: https://docs.labelbox.com/en/configure-editor/queue-system#reservation-system

```
>>> project.set_labeling_parameter_overrides([
>>>     (data_row_1, 2, 3), (data_row_2, 1, 4)])
```

Parameters **data** (*iterable*) – An iterable of tuples. Each tuple must contain (DataRow, priority<int>, number_of_labels<int>) for the new override.

Priority:

- **Data will be labeled in priority order.**
  - A lower number priority is labeled first.
  - Minimum priority is 1.
- **Priority is not the queue position.**
  - The position is determined by the relative priority.
  - **E.g. [(data_row_1, 5,1), (data_row_2, 2,1), (data_row_3, 10,1)]** will be assigned in the following order: [data_row_2, data_row_1, data_row_3]
- Datarows with parameter overrides will appear before datarows without overrides.
- **The priority only effects items in the queue.**
  - Assigning a priority will not automatically add the item back into the queue.

Number of labels:

- **The number of times a data row should be labeled.**
  - Creates duplicate data rows in a project (one for each number of labels).
- **New duplicated data rows will be added to the queue.**
  - Already labeled duplicates will not be sent back to the queue.
- **The queue will never assign the same datarow to a single labeler more than once.**

- **If the number of labels is greater than the number of labelers working on a project then** the extra items will remain in the queue (this can be fixed by removing the override at any time).

- Setting this to 1 will result in the default behavior (no duplicates).

Returns bool, indicates if the operation was a success.

setup(*labeling_frontend*, *labeling_frontend_options*) → None

Finalizes the Project setup.

Parameters

- **labeling_frontend** (LabelingFrontend) – Which UI to use to label the data.
- **labeling_frontend_options** (*dict or str*) – Labeling frontend options, a.k.a. project ontology. If given a *dict* it will be converted to *str* using *json.dumps*.

setup_editor(*ontology*) → None

Sets up the project using the Pictor editor.

Parameters **ontology** (Ontology) – The ontology to attach to the project

unset_labeling_parameter_overrides(*data_rows*) → bool

Removes labeling parameter overrides to this project.

- This will remove unlabeled duplicates in the queue.

Parameters **data_rows** (*iterable*) – An iterable of DataRows.

Returns bool, indicates if the operation was a success.

update(***kwargs*)

Updates this DB object with new values. Values should be passed as key-value arguments with field names as keys:

```
>>> db_object.update(name="New name", title="A title")
```

Kwargs: Key-value arguments defining which fields should be updated for which values. Keys must be field names in this DB object's type.

Raise:

InvalidAttributeError: if there exists a key in *kwargs* that's not a field in this object type.

update_project_resource_tags(*resource_tag_ids: List[str]*) → List[labelbox.schema.resource_tag.ResourceTag]

Creates project resource tags

Parameters **resource_tag_ids** –

Returns a list of ResourceTag ids that was created.

upload_annotations(*name: str*, *annotations: Union[str, pathlib.Path, Iterable[Dict]]*, *validate: bool = False*) → *BulkImportRequest*

Uploads annotations to a new Editor project.

Parameters

- **name** (*str*) – name of the BulkImportRequest job
- **annotations** (*str or Path or Iterable*) – url that is publicly accessible by Labelbox containing an ndjson file OR local path to an ndjson file OR iterable of annotation rows
- **validate** (*bool*) – Whether or not to validate the payload before uploading.

Returns BulkImportRequest

upsert_instructions(*instructions_file: str*) → None

- Uploads instructions to the UI. Running more than once will replace the instructions

Parameters **instructions_file** (*str*) – Path to a local file. * Must be a pdf or html file

Raises **ValueError** –

- project must be setup * instructions file must have a ".pdf" or ".html" extension

upsert_review_queue(*quota_factor*) → None

Sets the the proportion of total assets in a project to review.
More information can be found here: https://docs.labelbox.com/en/quality-assurance/review-labels#configure-review-percentage

Parameters **quota_factor** (*float*) – Which part (percentage) of the queue to reinitiate. Between 0 and 1.

video_label_generator(*timeout_seconds=600*, ***kwargs*)

Download video annotations
Returns LabelGenerator for accessing labels for each video

class labelbox.schema.project.**ProjectMember**(*client*, *field_values*)

Bases: labelbox.orm.db_object.DbObject

CHAPTER

FOURTEEN

REVIEW

class labelbox.schema.review.**Review**(*client*, *field_values*)

Bases: labelbox.orm.db_object.DbObject, labelbox.orm.db_object.Deletable, labelbox.orm.db_object.Updateable

Reviewing labeled data is a collaborative quality assurance technique.

A Review object indicates the quality of the assigned Label. The aggregated review numbers can be obtained on a Project object.

created_at

Type datetime

updated_at

Type datetime

score

Type float

created_by

ToOne relationship to User
Type Relationship

organization

ToOne relationship to Organization
Type Relationship

project

ToOne relationship to Project
Type Relationship

label

ToOne relationship to Label
Type Relationship

class **NetScore**(*value*)

Bases: enum.Enum

Negative, Zero, or Positive.

CHAPTER

FIFTEEN

TASK

class labelbox.schema.task.**Task**(*client*, *field_values*)

Bases: labelbox.orm.db_object.DbObject

Represents a server-side process that might take a longer time to process. Allows the Task state to be updated and checked on the client side.

updated_at

Type datetime

created_at

Type datetime

name

Type str

status

Type str

completion_percentage

Type float

created_by

ToOne relationship to User

Type Relationship

organization

ToOne relationship to Organization

Type Relationship

property errors: Optional[Dict[str, Any]]

Downloads the result file from Task

refresh() → None

Refreshes Task data from the server.

property result: List[Dict[str, Any]]

Fetch the result for a task

wait_till_done(*timeout_seconds=300*) → None

Waits until the task is completed. Periodically queries the server to update the task attributes.

Parameters **timeout_seconds** (*float*) – Maximum time this method can block, in seconds. Defaults to one minute.

CHAPTER

SIXTEEN

USER

class labelbox.schema.user.**User**(*client*, *field_values*)

Bases: labelbox.orm.db_object.DbObject

A User is a registered Labelbox user (for example you) associated with data they create or import and an Organization they belong to.

updated_at

Type datetime

created_at

Type datetime

email

Type str

name

Type str

nickname

Type str

intercom_hash

Type str

picture

Type str

is_viewer

Type bool

is_external_viewer

Type bool

organization

ToOne relationship to Organization

Type Relationship

created_tasks

ToMany relationship to Task

Type Relationship

projects

ToMany relationship to Project

Type Relationship

remove_from_project(*project:* Project) → None

Removes a User from a project. Only used for project based users. Project based user means their org role is "NONE"

Parameters **project** (Project) – Project to remove user from

update_org_role(*role: Role*) → None

Updated the `User`s organization role.

See client.get_roles() to get all valid roles If you a user is converted from project level permissions to org level permissions and then convert back, their permissions will remain for each individual project

Parameters **role** (*Role*) – The role that you want to set for this user.

upsert_project_role(*project:* Project, *role: Role*) → None

Updates or replaces a User's role in a project.

Parameters

- **project** (Project) – The project to update the users permissions for
- **role** (*Role*) – The role to assign to this user in this project.

CHAPTER

SEVENTEEN

WEBHOOK

class labelbox.schema.webhook.**Webhook**(*client*, *field_values*)

Bases: labelbox.orm.db_object.DbObject, labelbox.orm.db_object.Updateable

Represents a server-side rule for sending notifications to a web-server whenever one of several predefined actions happens within a context of a Project or an Organization.

updated_at

Type datetime

created_at

Type datetime

url

Type str

topics

LABEL_CREATED, LABEL_UPDATED, LABEL_DELETED REVIEW_CREATED, REVIEW_UPDATED, REVIEW_DELETED

Type str

status

ACTIVE, INACTIVE, REVOKED

Type str

class Status(*value*)

Bases: enum.Enum

An enumeration.

class Topic(*value*)

Bases: enum.Enum

An enumeration.

static create(*client*, *topics*, *url*, *secret*, *project*) → *labelbox.schema.webhook.Webhook*

Creates a Webhook.

Parameters

- **client** (Client) – The Labelbox client used to connect to the server.
- **topics** (*list of str*) – A list of topics this Webhook should get notifications for. Must be one of Webhook.Topic
- **url** (*str*) – The URL to which notifications should be sent by the Labelbox server.
- **secret** (*str*) – A secret key used for signing notifications.

- **project** (*Project or None*) – The project for which notifications should be sent. If None notifications are sent for all events in your organization.

Returns A newly created Webhook.

Raises **ValueError** – If the topic is not one of Topic or status is not one of Status

Information on configuring your server can be found here (this is where the url points to and the secret is set).
https://docs.labelbox.com/en/configure-editor/webhooks-setup#setup-steps

delete() → None

Deletes the webhook

update(*topics=None*, *url=None*, *status=None*)

Updates the Webhook.

Parameters

- **topics** (*Optional[List[Topic]]*) – The new topics.
- **Optional[str)** (*url*) – The new URL value.
- **status** (*Optional[Status]*) – The new status. If an argument is set to None then no updates will be made to that field.

CHAPTER

EIGHTEEN

EXCEPTIONS

exception labelbox.exceptions.**ApiLimitError**(*message*, *cause=None*)

Bases: *labelbox.exceptions.LabelboxError*

Raised when the user performs too many requests in a short period of time.

exception labelbox.exceptions.**AuthenticationError**(*message*, *cause=None*)

Bases: *labelbox.exceptions.LabelboxError*

Raised when an API key fails authentication.

exception labelbox.exceptions.**AuthorizationError**(*message*, *cause=None*)

Bases: *labelbox.exceptions.LabelboxError*

Raised when a user is unauthorized to perform the given request.

exception labelbox.exceptions.**InconsistentOntologyException**

Bases: Exception

exception labelbox.exceptions.**InternalServerError**(*message*, *cause=None*)

Bases: *labelbox.exceptions.LabelboxError*

Nondescript prisma or 502 related errors.

Meant to be retryable.

TODO: these errors need better messages from platform

exception labelbox.exceptions.**InvalidAttributeError**(*db_object_type*, *field*)

Bases: *labelbox.exceptions.LabelboxError*

Raised when a field (name or Field instance) is not valid or found for a specific DB object type.

exception labelbox.exceptions.**InvalidQueryError**(*message*, *cause=None*)

Bases: *labelbox.exceptions.LabelboxError*

Indicates a malconstructed or unsupported query (either by GraphQL in general or by Labelbox specifically). This can be the result of either client or server side query validation.

exception labelbox.exceptions.**LabelboxError**(*message*, *cause=None*)

Bases: Exception

Base class for exceptions.

exception labelbox.exceptions.**MALValidationError**(*message*, *cause=None*)

Bases: *labelbox.exceptions.LabelboxError*

Raised when user input is invalid for MAL imports.

exception labelbox.exceptions.**MalformedQueryException**

Bases: Exception

Raised when the user submits a malformed query.

exception labelbox.exceptions.**NetworkError**(*cause*)

Bases: *labelbox.exceptions.LabelboxError*

Raised when an HTTPError occurs.

exception labelbox.exceptions.**OperationNotAllowedException**

Bases: Exception

Raised when user does not have permissions to a resource or has exceeded usage limit

exception labelbox.exceptions.**ResourceConflict**(*message*, *cause=None*)

Bases: *labelbox.exceptions.LabelboxError*

Exception raised when a given resource conflicts with another.

exception labelbox.exceptions.**ResourceCreationError**(*message*, *cause=None*)

Bases: *labelbox.exceptions.LabelboxError*

Indicates that a resource could not be created in the server side due to a validation or transaction error

exception labelbox.exceptions.**ResourceNotFoundError**(*db_object_type*, *params*)

Bases: *labelbox.exceptions.LabelboxError*

Exception raised when a given resource is not found.

exception labelbox.exceptions.**TimeoutError**(*message*, *cause=None*)

Bases: *labelbox.exceptions.LabelboxError*

Raised when a request times-out.

exception labelbox.exceptions.**UuidError**(*message*, *cause=None*)

Bases: *labelbox.exceptions.LabelboxError*

Raised when there are repeat Uuid's in bulk import request.

exception labelbox.exceptions.**ValidationFailedError**(*message*, *cause=None*)

Bases: *labelbox.exceptions.LabelboxError*

Exception raised for when a GraphQL query fails validation (query cost, etc.) E.g. a query that is too expensive, or depth is too deep.

CHAPTER

NINETEEN

PAGINATION

class labelbox.pagination.**PaginatedCollection**(*client: Client*, *query: str*, *params: Dict[str, str]*, *dereferencing: Union[List[str], Dict[str, Any]]*, *obj_class: Union[Type[DbObject], Callable[[Any, Any], Any]]*, *cursor_path: Optional[List[str]] = None*, *experimental: bool = False*)

Bases: `object`

An iterable collection of database objects (Projects, Labels, etc...).

Implements automatic (transparent to the user) paginated fetching during iteration. Intended for use by library internals and not by the end user. For a list of attributes see __init__(...) documentation. The params of __init__ map exactly to object attributes.

__init__(*client: Client*, *query: str*, *params: Dict[str, str]*, *dereferencing: Union[List[str], Dict[str, Any]]*, *obj_class: Union[Type[DbObject], Callable[[Any, Any], Any]]*, *cursor_path: Optional[List[str]] = None*, *experimental: bool = False*)

Creates a PaginatedCollection.

Parameters

- **client** (*labelbox.Client*) – the client used for fetching data from DB.
- **query** (*str*) – Base query used for pagination. It must contain two '%d' placeholders, the first for pagination 'skip' clause and the second for the 'first' clause.
- **params** (*dict*) – Query parameters.
- **dereferencing** (*iterable*) – An iterable of str defining the keypath that needs to be dereferenced in the query result in order to reach the paginated objects of interest.
- **obj_class** (*type*) – The class of object to be instantiated with each dict containing db values.
- **cursor_path** – If not None, this is used to find the cursor
- **experimental** – Used to call experimental endpoints

CHAPTER

TWENTY

ENUMS

class labelbox.schema.enums.**AnnotationImportState**(*value*)

Bases: enum.Enum

State of the import job when importing annotations (RUNNING, FAILED, or FINISHED).

State	Description
RUNNING	Indicates that the import job is not done yet.
FAILED	Indicates the import job failed. Check *AnnotationImport.errors* for more information
FINISHED	Indicates the import job is no longer running. Check *AnnotationImport.statuses* for more information

class labelbox.schema.enums.**BulkImportRequestState**(*value*)

Bases: enum.Enum

State of the import job when importing annotations (RUNNING, FAILED, or FINISHED).

If you are not usinig MEA continue using BulkImportRequest. AnnotationImports are in beta and will change soon.

State	Description
RUNNING	Indicates that the import job is not done yet.
FAILED	Indicates the import job failed. Check *BulkImportRequest.errors* for more information
FINISHED	Indicates the import job is no longer running. Check *BulkImportRequest.statuses* for more information

CHAPTER

TWENTYONE

MODELRUN

class labelbox.schema.model_run.**ModelRun**(*client*, *field_values*)

Bases: labelbox.orm.db_object.DbObject

add_predictions(*name: str*, *predictions: Union[str, pathlib.Path, Iterable[Dict]]*) → *MEAPredictionImport*

Uploads predictions to a new Editor project. :param name: name of the AnnotationImport job :type name: str :param predictions: url that is publicly accessible by Labelbox containing an ndjson file OR local path to an ndjson file OR iterable of annotation rows

Returns AnnotationImport

delete()

Deletes specified model run.

Returns Query execution success.

delete_model_run_data_rows(*data_row_ids*)

Deletes data rows from model runs.

Parameters **data_row_ids** (*list*) – List of data row ids to delete from the model run.

Returns Query execution success.

export_labels(*download: bool = False*, *timeout_seconds: int = 600*) → Optional[Union[str, List[Dict[Any, Any]]]]

Experimental. To use, make sure client has enable_experimental=True.

Fetches Labels from the ModelRun

Parameters **download** (*bool*) – Returns the url if False

Returns URL of the data file with this ModelRun's labels. If download=True, this instead returns the contents as NDJSON format. If the server didn't generate during the *timeout_seconds* period, None is returned.

upsert_data_rows(*data_row_ids*, *timeout_seconds=60*)

Adds data rows to a model run without any associated labels :param data_row_ids: data row ids to add to mea :type data_row_ids: list :param timeout_seconds: Max waiting time, in seconds. :type timeout_seconds: float

Returns ID of newly generated async task

upsert_labels(*label_ids*, *timeout_seconds=60*)

Adds data rows and labels to a model run :param label_ids: label ids to insert :type label_ids: list :param timeout_seconds: Max waiting time, in seconds. :type timeout_seconds: float

Returns ID of newly generated async task

class labelbox.schema.model_run.**ModelRunDataRow**(*client*, *model_id*, **args*, ***kwargs*)

Bases: labelbox.orm.db_object.DbObject

CHAPTER

TWENTYTWO

MODEL

class labelbox.schema.model.**Model**(*client*, *field_values*)

Bases: labelbox.orm.db_object.DbObject

A model represents a program that has been trained and can make predictions on new data. .. attribute:: name

type str

model_runs

ToMany relationship to ModelRun

Type Relationship

create_model_run(*name*) → *ModelRun*

Creates a model run belonging to this model.

Parameters **name** (*string*) – The name for the model run.

Returns ModelRun, the created model run.

delete() → None

Deletes specified model.

Returns Query execution success.

CHAPTER

TWENTYTHREE

DATAROWMETADATA

class labelbox.schema.data_row_metadata.**DataRowMetadataKind**(*value*)

Bases: enum.Enum

An enumeration.

class labelbox.schema.data_row_metadata.**DataRowMetadataOntology**(*client*)

Bases: object

Ontology for data row metadata

Metadata provides additional context for a data rows. Metadata is broken into two classes reserved and custom. Reserved fields are defined by Labelbox and used for creating specific experiences in the platform.

```
>>> mdo = client.get_data_row_metadata_ontology()
```

bulk_delete(*deletes: List[labelbox.schema.data_row_metadata.DeleteDataRowMetadata]*) → List[labelbox.schema.data_row_metadata.DataRowMetadataBatchResponse]

Delete metadata from a datarow by specifiying the fields you want to remove

```
>>> delete = DeleteDataRowMetadata(
>>>                 data_row_id="datarow-id",
>>>                 fields=[
>>>                        "schema-id-1",
>>>                        "schema-id-2"
>>>                        ...
>>>                    ]
>>>    )
>>> mdo.batch_delete([metadata])
```

Parameters **deletes** – Data row and schema ids to delete

Returns list of unsuccessful deletions. An empty list means all data rows were successfully deleted.

bulk_export(*data_row_ids: List[str]*) → List[labelbox.schema.data_row_metadata.DataRowMetadata]

Exports metadata for a list of data rows

```
>>> mdo.bulk_export([data_row.uid for data_row in data_rows])
```

Parameters **data_row_ids** – List of data data rows to fetch metadata for

Returns A list of DataRowMetadata. There will be one DataRowMetadata for each data_row_id passed in. This is true even if the data row does not have any meta data. Data rows without metadata will have empty *fields*.

bulk_upsert(*metadata: List[labelbox.schema.data_row_metadata.DataRowMetadata]*) → List[labelbox.schema.data_row_metadata.DataRowMetadataBatchResponse]

Upsert datarow metadata

```
>>> metadata = DataRowMetadata(
>>>                 data_row_id="datarow-id",
>>>                 fields=[
>>>                        DataRowMetadataField(schema_id="schema-id", value=
↪"my-message"),
>>>                        ...
>>>                    ]
>>>    )
>>> mdo.batch_upsert([metadata])
```

Parameters **metadata** – List of DataRow Metadata to upsert

Returns list of unsuccessful upserts. An empty list means the upload was successful.

parse_metadata(*unparsed: List[Dict[str, List[Union[str, Dict]]]]*) → List[labelbox.schema.data_row_metadata.DataRowMetadata]

Parse metadata responses

```
>>> mdo.parse_metadata([metadata])
```

Parameters **unparsed** – An unparsed metadata export

Returns List of *DataRowMetadata*

Return type metadata

labelbox.schema.data_row_metadata.**String**

alias of labelbox.schema.data_row_metadata.ConstrainedStrValue

CHAPTER

TWENTYFOUR

ANNOTATIONIMPORT

class labelbox.schema.annotation_import.**AnnotationImport**(*client*, *field_values*)

Bases: labelbox.orm.db_object.DbObject

property errors: List[Dict[str, Any]]

Errors for each individual annotation uploaded. This is a subset of statuses :returns: List of dicts containing error messages. Empty list means there were no errors

See *AnnotationImport.statuses* for more details.

- This information will expire after 24 hours.

property inputs: List[Dict[str, Any]]

Inputs for each individual annotation uploaded. This should match the ndjson annotations that you have uploaded. :returns: Uploaded ndjson.

- This information will expire after 24 hours.

refresh() → None

Synchronizes values of all fields with the database.

property statuses: List[Dict[str, Any]]

Status for each individual annotation uploaded. :returns: A status for each annotation if the upload is done running.

See below table for more details

- This information will expire after 24 hours.

wait_until_done(*sleep_time_seconds: int = 10*, *show_progress: bool = False*) → None

Blocks import job until certain conditions are met. Blocks until the AnnotationImport.state changes either to *AnnotationImportState.FINISHED* or *AnnotationImportState.FAILED*, periodically refreshing object's state. :param sleep_time_seconds: a time to block between subsequent API calls :type sleep_time_seconds: int :param show_progress: should show progress bar :type show_progress: bool

class labelbox.schema.annotation_import.**LabelImport**(*client*, *field_values*)

Bases: *labelbox.schema.annotation_import.AnnotationImport*

classmethod create_from_file(*client:* labelbox.client.Client, *project_id: str*, *name: str*, *path: str*) → *labelbox.schema.annotation_import.LabelImport*

Create a label import job from a file of annotations

Parameters

- **client** – Labelbox Client for executing queries
- **project_id** – Project to import labels into
- **name** – Name of the import job. Can be used to reference the task later
- **path** – Path to ndjson file containing annotations

Returns LabelImport

classmethod create_from_objects(*client:* labelbox.client.Client, *project_id: str*, *name: str*, *labels: List[Dict[str, Any]]*) → *labelbox.schema.annotation_import.LabelImport*

Create a label import job from an in memory dictionary

Parameters

- **client** – Labelbox Client for executing queries
- **project_id** – Project to import labels into
- **name** – Name of the import job. Can be used to reference the task later
- **labels** – List of labels

Returns LabelImport

classmethod create_from_url(*client:* labelbox.client.Client, *project_id: str*, *name: str*, *url: str*) → *labelbox.schema.annotation_import.LabelImport*

Create a label annotation import job from a url The url must point to a file containing label annotations.

Parameters

- **client** – Labelbox Client for executing queries
- **project_id** – Project to import labels into
- **name** – Name of the import job. Can be used to reference the task later
- **url** – Url pointing to file to upload

Returns LabelImport

classmethod from_name(*client:* labelbox.client.Client, *project_id: str*, *name: str*, *as_json: bool = False*) → *labelbox.schema.annotation_import.LabelImport*

Retrieves an label import job.

Parameters

- **client** – Labelbox Client for executing queries
- **project_id** – ID used for querying import jobs
- **name** – Name of the import job.

Returns LabelImport

property parent_id: str

Identifier for this import. Used to refresh the status

class labelbox.schema.annotation_import.**MALPredictionImport**(*client*, *field_values*)

Bases: *labelbox.schema.annotation_import.AnnotationImport*

classmethod create_from_file(*client:* labelbox.client.Client, *project_id: str*, *name: str*, *path: str*) → *labelbox.schema.annotation_import.MALPredictionImport*

Create an MAL prediction import job from a file of annotations

Parameters

- **client** – Labelbox Client for executing queries
- **project_id** – Project to import labels into
- **name** – Name of the import job. Can be used to reference the task later
- **path** – Path to ndjson file containing annotations

Returns MALPredictionImport

classmethod create_from_objects(*client:* labelbox.client.Client, *project_id: str*, *name: str*, *predictions: List[Dict[str, Any]]*) → *labelbox.schema.annotation_import.MALPredictionImport*

Create an MAL prediction import job from an in memory dictionary

Parameters

- **client** – Labelbox Client for executing queries
- **project_id** – Project to import labels into
- **name** – Name of the import job. Can be used to reference the task later
- **predictions** – List of prediction annotations

Returns MALPredictionImport

classmethod create_from_url(*client:* labelbox.client.Client, *project_id: str*, *name: str*, *url: str*) → *labelbox.schema.annotation_import.MALPredictionImport*

Create an MAL prediction import job from a url The url must point to a file containing prediction annotations.

Parameters

- **client** – Labelbox Client for executing queries
- **project_id** – Project to import labels into
- **name** – Name of the import job. Can be used to reference the task later
- **url** – Url pointing to file to upload

Returns MALPredictionImport

classmethod from_name(*client:* labelbox.client.Client, *project_id: str*, *name: str*, *as_json: bool = False*) → *labelbox.schema.annotation_import.MALPredictionImport*

Retrieves an MAL import job.

Parameters

- **client** – Labelbox Client for executing queries
- **project_id** – ID used for querying import jobs
- **name** – Name of the import job.

Returns MALPredictionImport

property parent_id: str

Identifier for this import. Used to refresh the status

class labelbox.schema.annotation_import.**MEAPredictionImport**(*client*, *field_values*)

Bases: *labelbox.schema.annotation_import.AnnotationImport*

classmethod create_from_file(*client:* labelbox.client.Client, *model_run_id: str*, *name: str*, *path: str*) → *labelbox.schema.annotation_import.MEAPredictionImport*

Create an MEA prediction import job from a file of annotations

Parameters

- **client** – Labelbox Client for executing queries
- **model_run_id** – Model run to import labels into
- **name** – Name of the import job. Can be used to reference the task later

- **path** – Path to ndjson file containing annotations

Returns MEAPredictionImport

classmethod create_from_objects(*client:* labelbox.client.Client, *model_run_id: str*, *name*, *predictions*) → *labelbox.schema.annotation_import.MEAPredictionImport*

Create an MEA prediction import job from an in memory dictionary

Parameters

- **client** – Labelbox Client for executing queries
- **model_run_id** – Model run to import labels into
- **name** – Name of the import job. Can be used to reference the task later
- **predictions** – List of prediction annotations

Returns MEAPredictionImport

classmethod create_from_url(*client:* labelbox.client.Client, *model_run_id: str*, *name: str*, *url: str*) → *labelbox.schema.annotation_import.MEAPredictionImport*

Create an MEA prediction import job from a url The url must point to a file containing prediction annotations.

Parameters

- **client** – Labelbox Client for executing queries
- **model_run_id** – Model run to import labels into
- **name** – Name of the import job. Can be used to reference the task later
- **url** – Url pointing to file to upload

Returns MEAPredictionImport

classmethod from_name(*client:* labelbox.client.Client, *model_run_id: str*, *name: str*, *as_json: bool = False*) → *labelbox.schema.annotation_import.MEAPredictionImport*

Retrieves an MEA import job.

Parameters

- **client** – Labelbox Client for executing queries
- **model_run_id** – ID used for querying import jobs
- **name** – Name of the import job.

Returns MEAPredictionImport

property parent_id: str

Identifier for this import. Used to refresh the status

PYTHON MODULE INDEX

l

INDEX

Symbols

A

B

C

D

E

F

G

I

L

M

N

O

P

V

W