Python SDK reference

Release 3.41.0

Labelbox

Mar 17, 2023

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'*, *rest_endpoint='https://api.labelbox.com/api/v1'*)

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'*, *rest_endpoint='https://api.labelbox.com/api/v1'*)

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.

assign_global_keys_to_data_rows(*global_key_to_data_row_inputs: List[Dict[str, str]]*, *timeout_seconds=60*) → Dict[str, Union[str, List[Any]]]

Assigns global keys to data rows.

Parameters **global_key.** (*A list of dicts containing data_row_id and*) –

Returns

Dictionary containing 'status', 'results' and 'errors'.

'Status' contains the outcome of this job. It can be one of 'Success', 'Partial Success', or 'Failure'.

'Results' contains the successful global_key assignments, including global_keys that have been sanitized to Labelbox standards.

'Errors' contains global_key assignments that failed, along with the reasons for failure.

Examples

```
>>> global_key_data_row_inputs = [
    {"data_row_id": "cl7asgri20yvo075b4vtfedjb", "global_key": "key1"},
    {"data_row_id": "cl7asgri10yvg075b4pz176ht", "global_key": "key2"},
    ]
>>> job_result = client.assign_global_keys_to_data_rows(global_key_data_row_
↪inputs)
>>> print(job_result['status'])
Partial Success
>>> print(job_result['results'])
[{'data_row_id': 'cl7tv9wry00hlka6gai588ozv', 'global_key': 'gk', 'sanitized':␣
↪False}]
>>> print(job_result['errors'])
[{'data_row_id': 'cl7tpjzw30031ka6g4evqdfoy', 'global_key': 'gk"', 'error':
↪'Invalid global key'}]
```

clear_global_keys(*global_keys: List[str]*, *timeout_seconds=60*) → Dict[str, Union[str, List[Any]]]

Clears global keys for the data rows tha correspond to the global keys provided.

Parameters **keys** (*A list of global*) –

Returns

Dictionary containing 'status', 'results' and 'errors'.

'Status' contains the outcome of this job. It can be one of 'Success', 'Partial Success', or 'Failure'.

'Results' contains a list global keys that were successfully cleared.

'Errors' contains a list of global_keys correspond to the data rows that could not be modified, accessed by the user, or not found.

Examples

```
>>> job_result = client.clear_global_keys(["key1","key2","notfoundkey"])
>>> print(job_result['status'])
Partial Success
>>> print(job_result['results'])
['key1', 'key2']
>>> print(job_result['errors'])
[{'global_key': 'notfoundkey', 'error': 'Failed to find data row matching␣
↪provided global key'}]
```

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,
                name="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*, *media_type=None*) → *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.
- **media_type** (*MediaType or None*) – Media type of a new ontology

Returns The created Ontology

create_ontology_from_feature_schemas(*name*, *feature_schema_ids*, *media_type=None*) → *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
- **media_type** (*MediaType or None*) – Media type of a new 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>",
        media_type=MediaType.Image,
        queue_mode=QueueMode.Batch
    )
```

Parameters

- **name** (*str*) – A name for the project
- **description** (*str*) – A short summary for the project
- **media_type** (*MediaType*) – The type of assets that this project will accept
- **queue_mode** (*Optional[QueueMode]*) – The queue mode to use
- **auto_audit_percentage** (*Optional[float]*) – The percentage of data rows that will require more than 1 label
- **auto_audit_number_of_labels** (*Optional[float]*) – Number of labels required for data rows selected for multiple labeling (auto_audit_percentage)

Returns A new Project object.

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

delete_feature_schema_from_ontology(*ontology_id: str*, *feature_schema_id: str*) → labelbox.schema.ontology.DeleteFeatureFromOntologyResult

Deletes or archives a feature schema from an ontology. If the feature schema is a root level node with associated labels, it will be archived. If the feature schema is a nested node in the ontology and does not have associated labels, it will be deleted. If the feature schema is a nested node in the ontology and has associated labels, it will not be deleted.

Parameters

- **ontology_id** (*str*) – The ID of the ontology.
- **feature_schema_id** (*str*) – The ID of the feature schema.

Returns The result of the feature schema removal.

Return type DeleteFeatureFromOntologyResult

Example

```
>>> client.remove_feature_schema_from_ontology(<ontology_id>, <feature_schema_
↪id>)
```

delete_unused_feature_schema(*feature_schema_id: str*) → None

Deletes a feature schema if it is not used by any ontologies or annotations :param feature_schema_id: The id of the feature schema to delete :type feature_schema_id: str

Example

```
>>> client.delete_unused_feature_schema("cleabc1my012ioqvu5anyaabc")
```

delete_unused_ontology(*ontology_id: str*) → None

Deletes an ontology if it is not used by any annotations :param ontology_id: The id of the ontology to delete :type ontology_id: str

Example

```
>>> client.delete_unused_ontology("cleabc1my012ioqvu5anyaabc")
```

execute(*query=None*, *params=None*, *data=None*, *files=None*, *timeout=60.0*, *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_catalog_slice(*slice_id*) → *labelbox.schema.slice.CatalogSlice*

Fetches a Catalog Slice by ID.

Parameters **slice_id** (*str*) – The ID of the Slice

Returns CatalogSlice

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_ids_for_global_keys(*global_keys: List[str]*, *timeout_seconds=60*) → Dict[str, Union[str, List[Any]]]

Gets data row ids for a list of global keys.

Deprecation Notice: This function will soon no longer return 'Deleted Data Rows' as part of the 'results'. Global keys for deleted data rows will soon be placed under 'Data Row not found' portion.

Parameters **keys** (*A list of global*) –

Returns

Dictionary containing 'status', 'results' and 'errors'.

'Status' contains the outcome of this job. It can be one of 'Success', 'Partial Success', or 'Failure'.

'Results' contains a list of the fetched corresponding data row ids in the input order. For data rows that cannot be fetched due to an error, or data rows that do not exist, empty string is returned at the position of the respective global_key. More error information can be found in the 'Errors' section.

'Errors' contains a list of global_keys that could not be fetched, along with the failure reason

Examples

```
>>> job_result = client.get_data_row_ids_for_global_keys(["key1","key2"])
>>> print(job_result['status'])
Partial Success
>>> print(job_result['results'])
['cl7tv9wry00hlka6gai588ozv', 'cl7tv9wxg00hpka6gf8sh81bj']
>>> print(job_result['errors'])
[{'global_key': 'asdf', 'error': 'Data Row not found'}]
```

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_model_slice(*slice_id*) → labelbox.schema.slice.ModelSlice

Fetches a Model Slice by ID.

Parameters **slice_id** (*str*) – The ID of the Slice

Returns ModelSlice

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_unused_feature_schemas(*after: Optional[str] = None*) → List[str]

Returns a list of unused feature schema ids :param after: The cursor to use for pagination :type after: str

Returns A list of unused feature schema ids

Example

To get the first page of unused feature schema ids (100 at a time) >>> client.get_unused_feature_schemas() To get the next page of unused feature schema ids >>> client.get_unused_feature_schemas("cleabc1my012ioqvu5anyaabc")

get_unused_ontologies(*after: Optional[str] = None*) → List[str]

Returns a list of unused ontology ids :param after: The cursor to use for pagination :type after: str

Returns A list of unused ontology ids

Example

To get the first page of unused ontology ids (100 at a time) >>> client.get_unused_ontologies() To get the next page of unused ontology ids >>> client.get_unused_ontologies("cleabc1my012ioqvu5anyaabc")

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

Gets the current User database object.

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

insert_feature_schema_into_ontology(*feature_schema_id: str*, *ontology_id: str*, *position: int*) → None

Inserts a feature schema into an ontology. If the feature schema is already in the ontology, it will be moved to the new position. :param feature_schema_id: The feature schema id to upsert :type feature_schema_id: str :param ontology_id: The id of the ontology to insert the feature schema into :type ontology_id: str :param position: The position number of the feature schema in the ontology :type position: int

Example

```
>>> client.insert_feature_schema_into_ontology("cleabc1my012ioqvu5anyaabc",
↪"clefdvwl7abcgefgu3lyvcde", 2)
```

is_feature_schema_archived(*ontology_id: str*, *feature_schema_id: str*) → bool

Returns true if a feature schema is archived in the specified ontology, returns false otherwise.

Parameters

- **feature_schema_id** (*str*) – The ID of the feature schema
- **ontology_id** (*str*) – The ID of the ontology

Returns bool

unarchive_feature_schema_node(*ontology_id: str*, *root_feature_schema_id: str*) → None

Unarchives a feature schema node in an ontology. Only root level feature schema nodes can be unarchived. :param ontology_id: The ID of the ontology :type ontology_id: str :param root_feature_schema_id: The ID of the root level feature schema :type root_feature_schema_id: str

Returns None

update_feature_schema_title(*feature_schema_id: str*, *title: str*) → *labelbox.schema.ontology.FeatureSchema*

Updates a title of a feature schema :param feature_schema_id: The id of the feature schema to update :type feature_schema_id: str :param title: The new title of the feature schema :type title: str

Returns The updated feature schema

Example

```
>>> client.update_feature_schema_title("cleabc1my012ioqvu5anyaabc", "New Title")
```

upsert_feature_schema(*feature_schema: Dict*) → *labelbox.schema.ontology.FeatureSchema*

Upserts a feature schema :param feature_schema: Dict representing the feature schema to upsert

Returns The upserted feature schema

Example

Insert a new feature schema >>> tool = Tool(name="tool", tool=Tool.Type.BOUNDING_BOX, color="#FF0000") >>> client.upsert_feature_schema(tool.asdict()) Update an existing feature schema >>> tool = Tool(feature_schema_id="cleabc1my012ioqvu5anyaabc", name="tool", tool=Tool.Type.BOUNDING_BOX, color="#FF0000") >>> client.upsert_feature_schema(tool.asdict())

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, IMAGE_OVERLAY, HTML, RAW_TEXT, or TEXT_URL. TEXT attachment type is deprecated.

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 or name

Return type Dict, 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

global_key

User-generated globally unique 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 data row

Type dict

metadata_fields

metadata associated with the data row

Type list

metadata

metadata associated with the data row as list of DataRowMetadataField. When importing Data Rows with metadata, use *metadata_fields* instead

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*, *attachment_name=None*) → *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.
- **attachment_name** (*str*) – (Optional) Asset attachment name.

Returns *AssetAttachment* DB object.

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

get_winning_label_id(*project_id: str*) → Optional[str]

Retrieves the winning label ID, i.e. the one that was marked as the best for a particular data row, in a project's workflow.

Parameters **project_id** (*str*) – ID of the project containing the data row

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.

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*, *include_metadata: bool = False*) → 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.
- **include_metadata** (*bool*) – True to return related DataRow metadata

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

queue_mode

Type string

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

`task_queues`

ToMany relationship to TaskQueue

Type Relationship

`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: Optional[List[Union[str,* labelbox.schema.data_row.DataRow*]]] = None*, *priority: int = 5*, *consensus_settings: Optional[Dict[str, float]] = None*, *global_keys: Optional[List[str]] = None*)

Create a new batch for a project. One of *global_keys* or *data_rows* must be provided but not both.

Parameters

- **name** – a name for the batch, must be unique within a project

- **data_rows** – Either a list of *DataRows* or Data Row ids.
- **global_keys** – global keys for data rows to add to the batch.
- **priority** – An optional priority for the Data Rows in the Batch. 1 highest -> 5 lowest
- **consensus_settings** – An optional dictionary with consensus settings: {'number_of_labels': 3, 'coverage_percentage': 0.1}

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=1800*, ***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" or "YYYY-MM-DD hh:mm:ss"
- **end** (*str*) – Latest date for labels, formatted "YYYY-MM-DD" or "YYYY-MM-DD hh:mm:ss"
- **last_activity_start** (*str*) – Will include all labels that have had any updates to data rows, issues, comments, metadata, or reviews since this timestamp. formatted "YYYY-MM-DD" or "YYYY-MM-DD hh:mm:ss"
- **last_activity_end** (*str*) – Will include all labels that do not have any updates to data rows, issues, comments, metadata, or reviews after this timestamp. formatted "YYYY-MM-DD" or "YYYY-MM-DD hh:mm:ss"

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*, *include_metadata: bool = False*) → List[Dict[str, str]]

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

Parameters

- **timeout_seconds** (*float*) – Max waiting time, in seconds.
- **include_metadata** (*bool*) – True to return related DataRow metadata

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.

export_v2(*task_name: Optional[str] = None*, *filters: Optional[labelbox.schema.export_filters.ProjectExportFilters] = None*, *params: Optional[labelbox.schema.export_params.ProjectExportParams] = None*) → *labelbox.schema.task.Task*

Creates a project run export task with the given params and returns the task.

For more information visit: https://docs.labelbox.com/docs/exports-v2#export-from-a-project-python-sdk

```
>>>     task = project.export_v2(
>>>         filters={
>>>             "last_activity_at": ["2000-01-01 00:00:00", "2050-01-01 00:00:00
↪"],
>>>             "label_created_at": ["2000-01-01 00:00:00", "2050-01-01 00:00:00
↪"]
>>>         },
>>>         params={
>>>             "include_performance_details": False,
>>>             "include_labels": True
>>>         })
>>>     task.wait_till_done()
>>>     task.result
```

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.

get_queue_mode() → labelbox.schema.queue_mode.QueueMode

Provides the queue mode used for this project.

Deprecation notice: This method is deprecated and will be removed in a future version. To obtain the queue mode of a project, simply refer to the queue_mode attribute of a Project.

For more information, visit https://docs.labelbox.com/reference/migrating-to-workflows#upcoming-changes

Returns: the QueueMode for this project

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

move_data_rows_to_task_queue(*data_row_ids: List[str]*, *task_queue_id: str*)

Moves data rows to the specified task queue.

Parameters

- **data_row_ids** – a list of data row ids to be moved
- **task_queue_id** – the task queue id to be moved to, or None to specify the "Done" queue

Returns None if successful, or a raised error on failure

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

`task_queues`() → List[*labelbox.schema.task_queue.TaskQueue*]

Fetch all task queues that belong to this project

Returns A `List of `TaskQueue`s

`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 project with the specified attributes

Parameters **kwargs** – a dictionary containing attributes to be upserted

Note that the queue_mode cannot be changed after a project has been created.

Additionally, the quality setting cannot be changed after a project has been created. The quality mode for a project is inferred through the following attributes: Benchmark:

auto_audit_number_of_labels = 1 auto_audit_percentage = 1.0

Consensus: auto_audit_number_of_labels > 1 auto_audit_percentage <= 1.0

Attempting to switch between benchmark and consensus modes is an invalid operation and will result in an error.

`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.

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]]

Fetch the error associated with an import task.

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

Fetch data rows which failed to be created for an import task.

refresh() → None

Refreshes Task data from the server.

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

Fetch the result for an import task.

wait_till_done(*timeout_seconds: int = 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

TASK QUEUE

class labelbox.schema.task_queue.**TaskQueue**(*client*, **args*, ***kwargs*)

Bases: labelbox.orm.db_object.DbObject

a task queue

Attributes name description queue_type data_row_count

Relationships project organization pass_queue fail_queue

CHAPTER

SEVENTEEN

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

EIGHTEEN

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

NINETEEN

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.**ConfidenceNotSupportedException**

Bases: Exception

Raised when confidence is specified for unsupported annotation type

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.**ProcessingWaitTimeout**

Bases: Exception

Raised when waiting for the data rows to be processed takes longer than allowed

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

TWENTY

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

get_many(*n: int*)

Iterates over self and returns first n results This method is idempotent

Parameters **n** (*int*) – Number of elements to retrieve

get_one()

Iterates over self and returns first value This method is idempotent

CHAPTER

TWENTYONE

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

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

Bases: enum.Enum

Status of an asynchronous job over a collection.

- – State
  – Description
- – SUCCESS
  – Indicates job has successfully processed entire collection of data
- – PARTIAL SUCCESS
  – Indicates some data in the collection has succeeded and other data have failed
- – FAILURE
  – Indicates job has failed to process entire collection of data

CHAPTER

TWENTYTWO

MODELRUN

class labelbox.schema.model_run.**DataSplit**(*value*)

Bases: enum.Enum

An enumeration.

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

Bases: labelbox.orm.db_object.DbObject

class **Status**(*value*)

Bases: enum.Enum

An enumeration.

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

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.

get_config() → Dict[str, Any]

Gets Model Run's training metadata :returns: training metadata as a dictionary

reset_config() → Dict[str, Any]

Resets Model Run's training metadata config :returns: Model Run id and reset training metadata

update_config(*config: Dict[str, Any]*) → Dict[str, Any]

Updates the Model Run's training metadata config :param config: A dictionary of keys and values :type config: dict

Returns Model Run id and updated training metadata

upsert_data_rows(*data_row_ids=None*, *global_keys=None*, *timeout_seconds=3600*)

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

Returns ID of newly generated async task

upsert_labels(*label_ids: Optional[List[str]] = None*, *project_id: Optional[str] = None*, *timeout_seconds=3600*)

Adds data rows and labels to a Model Run :param label_ids: label ids to insert :type label_ids: list :param project_id: project uuid, all project labels will be uploaded

Either label_ids OR project_id is required but NOT both

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

Returns ID of newly generated async task

upsert_predictions_and_send_to_project(*name: str*, *predictions: Union[str, pathlib.Path, Iterable[Dict]]*, *project_id: str*, *priority: Optional[int] = 5*) → *MEAPredictionImport*

Provides a convenient way to execute the following steps in a single function call:

1. Upload predictions to a Model
2. Create a batch from data rows that had predictions assocated with them
3. Attach the batch to a project
4. Add those same predictions to the project as MAL annotations

Note that partial successes are possible. If it is important that all stages are successful then check the status of each individual task with task.errors. E.g.

```
>>>    mea_import_job, batch, mal_import_job = upsert_predictions_and_send_to_
↪project(name, predictions, project_id)
>>>    # handle mea import job successfully created (check for job failure or␣
↪partial failures)
>>>    print(mea_import_job.status, mea_import_job.errors)
>>>    if batch is None:
>>>        # Handle batch creation failure
>>>    if mal_import_job is None:
>>>        # Handle mal_import_job creation failure
>>>    else:
>>>        # handle mal import job successfully created (check for job failure␣
↪or partial failures)
>>>        print(mal_import_job.status, mal_import_job.errors)
```

Parameters

- **name** (*str*) – name of the AnnotationImport job as well as the name of the batch import

- **predictions** (*Iterable*) – iterable of annotation rows
- **project_id** (*str*) – id of the project to import into
- **priority** (*int*) – priority of the job

Returns Tuple[MEAPredictionImport, Batch, MEAToMALPredictionImport] If any of these steps fail the return value will be None.

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

Bases: labelbox.orm.db_object.DbObject

CHAPTER

TWENTYTHREE

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*, *config=None*) → *ModelRun*

Creates a model run belonging to this model.

Parameters

- **name** (*string*) – The name for the model run.
- **config** (*json*) – Model run's training metadata config

Returns ModelRun, the created model run.

delete() → None

Deletes specified model.

Returns Query execution success.

CHAPTER

TWENTYFOUR

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.

create_schema(*name: str*, *kind:* labelbox.schema.data_row_metadata.DataRowMetadataKind, *options: Optional[List[str]] = None*) → labelbox.schema.data_row_metadata.DataRowMetadataSchema

Create metadata schema

```
>>> mdo.create_schema(name, kind, options)
```

Parameters

- **name** (*str*) – Name of metadata schema
- **kind** (DataRowMetadataKind) – Kind of metadata schema as *DataRowMetadataKind*
- **options** (*List[str]*) – List of Enum options

Returns Created metadata schema as *DataRowMetadataSchema*

Raises **KeyError** – When provided name is not a valid custom metadata

delete_schema(*name: str*) → bool

Delete metadata schema

```
>>> mdo.delete_schema(name)
```

Parameters **name** – Name of metadata schema to delete

Returns True if deletion is successful, False if unsuccessful

Raises **KeyError** – When provided name is not a valid custom metadata

get_by_name(*name: str*) → Union[labelbox.schema.data_row_metadata.DataRowMetadataSchema, Dict[str, labelbox.schema.data_row_metadata.DataRowMetadataSchema]]

Get metadata by name

```
>>> mdo.get_by_name(name)
```

Parameters **name** (*str*) – Name of metadata schema

Returns Metadata schema as *DataRowMetadataSchema* or dict, in case of Enum metadata

Raises **KeyError** – When provided name is not presented in neither reserved nor custom metadata list

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

parse_metadata_fields(*unparsed: List[Dict[str, Dict]]*) → List[labelbox.schema.data_row_metadata.DataRowMetadataField]

Parse metadata fields as list of *DataRowMetadataField*

```
>>> mdo.parse_metadata_fields([metadata_fields])
```

Parameters **unparsed** – An unparsed list of metadata represented as a dict containing 'schemaId' and 'value'

Returns List of *DataRowMetadataField*

Return type metadata

parse_upsert_metadata(*metadata_fields*) → List[Dict[str, Any]]

Converts either *DataRowMetadataField* or a dictionary representation of *DataRowMetadataField* into a validated, flattened dictionary of metadata fields that are used to create data row metadata. Used internally in *Dataset.create_data_rows()*

Parameters **metadata_fields** – List of *DataRowMetadataField* or a dictionary representation of *DataRowMetadataField*

Returns List of dictionaries representing a flattened view of metadata fields

refresh_ontology()

Update the *DataRowMetadataOntology* instance with the latest metadata ontology schemas

update_enum_option(*name: str*, *option: str*, *new_option: str*) → labelbox.schema.data_row_metadata.DataRowMetadataSchema

Update Enum metadata schema option

```
>>> mdo.update_enum_option(name, option, new_option)
```

Parameters

- **name** (*str*) – Name of metadata schema to update
- **option** (*str*) – Name of Enum option to update
- **new_option** (*str*) – New name of Enum option

Returns Updated metadata schema as *DataRowMetadataSchema*

Raises **KeyError** – When provided name is not a valid custom metadata

update_schema(*name: str*, *new_name: str*) → labelbox.schema.data_row_metadata.DataRowMetadataSchema

Update metadata schema

```
>>> mdo.update_schema(name, new_name)
```

Parameters

- **name** (*str*) – Current name of metadata schema
- **new_name** (*str*) – New name of metadata schema

Returns Updated metadata schema as *DataRowMetadataSchema*

Raises **KeyError** – When provided name is not a valid custom metadata

labelbox.schema.data_row_metadata.**String**

alias of labelbox.schema.data_row_metadata.ConstrainedStrValue

CHAPTER

TWENTYFIVE

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*, *project_id: str*, *name: str*, *labels: Union[List[Dict[str, Any]], List[*Label*]]*) → *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*, *project_id: str*, *name: str*, *predictions: Union[List[Dict[str, Any]], List[*Label*]]*) → *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*, *model_run_id: str*, *name*, *predictions: Union[List[Dict[str, Any]], List[*Label*]]*) → *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

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

Bases: *labelbox.schema.annotation_import.AnnotationImport*

classmethod create_for_model_run_data_rows(*client:* labelbox.client.Client, *model_run_id: str*, *data_row_ids: List[str]*, *project_id: str*, *name: str*) → *labelbox.schema.annotation_import.MEAToMALPredictionImport*

Create an MEA to MAL prediction import job from a list of data row ids of a specific model run

Parameters

- **client** – Labelbox Client for executing queries
- **data_row_ids** – A list of data row ids
- **model_run_id** – model run id

Returns MEAToMALPredictionImport

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

Retrieves an MEA to 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

CHAPTER

TWENTYSIX

BATCH

class labelbox.schema.batch.**Batch**(*client*, *project_id*, **args*, *failed_data_row_ids=None*, ***kwargs*)

Bases: labelbox.orm.db_object.DbObject

A Batch is a group of data rows submitted to a project for labeling

name

Type str

created_at

Type datetime

updated_at

Type datetime

deleted

Type bool

project

ToOne relationship to Project

Type Relationship

created_by

ToOne relationship to User

Type Relationship

delete() → None

Deletes the given batch.

Note: Batch deletion for batches that has labels is forbidden.

Parameters **batch** (Batch) – Batch to remove queued data rows from

delete_labels(*set_labels_as_template=False*) → None

Deletes labels that were created for data rows in the batch.

Parameters

- **batch** (Batch) – Batch to remove queued data rows from
- **set_labels_as_template** (*bool*) – When set to true, the deleted labels will be kept as templates.

export_data_rows(*timeout_seconds=120*, *include_metadata: bool = False*) → Generator

Returns a generator that produces all data rows that are currently in this batch.

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.
- **include_metadata** (*bool*) – True to return related DataRow metadata

Returns Generator that yields DataRow objects belonging to this batch.

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

project() → *Project*

Returns Project which this Batch belongs to

Raises ***LabelboxError*** – if the project is not found

remove_queued_data_rows() → None

Removes remaining queued data rows from the batch and labeling queue.

Parameters **batch** (Batch) – Batch to remove queued data rows from

CHAPTER

TWENTYSEVEN

RESOURCETAG

class labelbox.schema.resource_tag.**ResourceTag**(*client*, *field_values*)

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

Resource tag to label and identify your labelbox resources easier.

text

Type str

color

Type str

project_resource_tag

ToMany relationship to ProjectResourceTag

Type Relationship

CHAPTER

TWENTYEIGHT

SLICE

class labelbox.schema.slice.**Slice**(*client*, *field_values*)

Bases: labelbox.orm.db_object.DbObject

A Slice is a saved set of filters (saved query). This is an abstract class and should not be instantiated.

name

Type datetime

description

Type datetime

created_at

Type datetime

updated_at

Type datetime

filter

Type json

CHAPTER

TWENTYNINE

CATALOGSLICE

class labelbox.schema.slice.**CatalogSlice**(*client*, *field_values*)

Bases: *labelbox.schema.slice.Slice*

Represents a Slice used for filtering data rows in Catalog.

get_data_row_ids() → *labelbox.pagination.PaginatedCollection*

Fetches all data row ids that match this Slice

Returns A PaginatedCollection of data row ids

PYTHON MODULE INDEX

INDEX

Symbols

A

B

C

D

E

F

M

N

O

P

Q

R

S

T

U

V

W