
Labelbox Python API reference

Release 2.6.0

Labelbox

Jul 27, 2021

CONTENTS:

1	Client	1
2	AssetAttachment	5
3	AssetMetadata	7
4	Benchmark	9
5	BulkImportRequest	11
6	DataRow	13
7	Dataset	15
8	Label	19
9	LabelingFrontend	21
10	LabelingFrontendOptions	23
11	LabelingParameterOverride	25
12	Ontology	27
13	Organization	29
14	Prediction	31
15	PredictionModel	33
16	Project	35
17	Review	41
18	Task	43
19	User	45
20	Webhook	47
21	Exceptions	49
22	Pagination	51

23 Enums	53
Python Module Index	55
Index	57

CLIENT

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

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)  
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.

```
>>> import logger  
>>> 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

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

```
create_dataset (**kwargs)
```

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 ****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_project (***kwargs*)

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, params=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.
- **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.

get_dataset (*dataset_id*)

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*)
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_labeling_frontends (*where=None*)
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_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*)
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 ()

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

Return type Roles

get_user()

Gets the current User database object.

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


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, or IMAGE_OVERLAY

Type `str`

attachment_value

URL to an external file or a string of text

Type `str`

class `AttachmentType` (*value*)

Bases: `enum.Enum`

An enumeration.

ASSETMETADATA

```
class labelbox.schema.asset_metadata.AssetMetadata (*args, **kwargs)
```

```
    Bases: labelbox.orm.db_object.DbObject
```

AssetMetadata is deprecated. Use *AssetAttachment* instead

```
class MetaType (value)
```

```
    Bases: enum.Enum
```

An enumeration.

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

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

    property errors
        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

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

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 = 30) → 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

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

    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

metadata

This Relationship is Deprecated. Please use *DataRow.attachments()* instead

Type Relationship

predictions

ToMany relationship to Prediction

Type Relationship

static bulk_delete (*data_rows*)

Deletes all the given DataRows.

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

create_attachment (*attachment_type, attachment_value*)

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.

create_metadata (*meta_type, meta_value*)

This function is deprecated. Use create_attachment instead

Returns AssetMetadata

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

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 (***kwargs*)

 Creates a single DataRow belonging to this dataset.

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

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

Raises

- **`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*)

Creates multiple `DataRow` objects based on the given *items*.

Each element in *items* can be either a *str* or a *dict*. If it is a *str*, then it is interpreted as a local file path. The file is uploaded to Labelbox and a `DataRow` referencing it is created.

If an item is a *dict*, then it could support one of the two following structures

1. For static imagery, video, and text it should map `DataRow` fields (or their names) to values. At the minimum an *item* passed as a *dict* must contain a `DataRow.row_data` key and value.
2. For tiled imagery the dict must match the import structure specified in the link below <https://docs.labelbox.com/data-model/en/index-en#tiled-imagery-import>

```
>>> dataset.create_data_rows([
>>>     {DataRow.row_data: "http://my_site.com/photos/img_01.jpg"},
>>>     "path/to/file2.jpg",
>>>     {"tileLayerUrl" : "http://", ...}
>>> ])
```

For an example showing how to upload tiled data_rows see the following notebook: https://github.com/Labelbox/labelbox-python/blob/ms/develop/model_assisted_labeling/tiled_imagery_mal.ipynb

Parameters *items* (*iterable of (dict or str)*) – See above for details.

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`.

`data_row_for_external_id` (*external_id*)

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*)

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`.
- **`limit`** (*int*) – The maximum number of data rows to return for the given *external_id*

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.

export_data_rows (*timeout_seconds=120*)

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

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.

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 (anno-
    tations).

    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)
        Deletes all the given Labels.

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

create_benchmark()

Creates a Benchmark for this Label.

Returns The newly created Benchmark.

create_review(kwargs)**

Creates a Review for this label.

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

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`

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
```


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

ONTOLOGY

```
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: List[labelbox.schema.ontology.Tool]
                                              = <factory>,      classifications:
                                              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)
```


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

invite_limit () → labelbox.schema.invite.InviteLimit

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

Returns InviteLimit

invite_user (*email*: str, *role*: labelbox.schema.role.Role, *project_roles*: Optional[List[labelbox.schema.role.ProjectRole]] = None) → labelbox.schema.invite.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

This function is currently experimental and has a few limitations that will be resolved in future releases 1. If you try to add an unsupported you will get an error referring to invalid foreign keys

- In this case *role.get_roles* is likely not getting the right ids

2. 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)

3. Some server side response are unclear (e.g. if the user invites themselves *None* is returned which the SDK raises as a *LabelboxError*)

remove_user (*user: labelbox.schema.user.User*)

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

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

PREDICTION

class `labelbox.schema.prediction.Prediction` (*client, field_values*)

Bases: `labelbox.orm.db_object.DbObject`

A prediction created by a `PredictionModel`. Legacy editor only.

Refer to `BulkImportRequest` if using the new Editor.

updated_at

Type `datetime`

created_at

Type `datetime`

label

Type `str`

agreement

Type `float`

organization

ToOne relationship to `Organization`

Type `Relationship`

prediction_model

ToOne relationship to `PredictionModel`

Type `Relationship`

data_row

ToOne relationship to `DataRow`

Type `Relationship`

project

ToOne relationship to `Project`

Type `Relationship`

PREDICTIONMODEL

class `labelbox.schema.prediction.PredictionModel` (*client, field_values*)

Bases: `labelbox.orm.db_object.DbObject`

A PredictionModel creates a Prediction. Legacy editor only.

Refer to BulkImportRequest if using the new Editor.

updated_at

Type datetime

created_at

Type datetime

name

Type str

slug

Type str

version

Type int

created_by

ToOne relationship to User

Type Relationship

organization

ToOne relationship to Organization

Type Relationship

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
```

reviews

ToMany relationship to Review

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

active_prediction_model

ToOne relationship to PredictionModel

Type Relationship

predictions

ToMany relationship to Prediction

Type Relationship

ontology

ToOne relationship to Ontology

Type Relationship

create_label (***kwargs*)

Creates a label on a Legacy Editor project. Not supported in the new Editor. :param ***kwargs*: Label attributes. At minimum, the label *DataRow*.

create_prediction (*label, data_row, prediction_model=None*)

Creates a Prediction within a Legacy Editor Project. Not supported in the new Editor.

Parameters

- **label** (*str*) – The *label* field of the new Prediction.
- **data_row** (*DataRow*) – The *DataRow* for which the Prediction is created.
- **prediction_model** (*PredictionModel or None*) – The *PredictionModel* within which the new Prediction is created. If *None* then this Project's *active_prediction_model* is used.

Returns A newly created Prediction.

Raises **labelbox.exceptions.InvalidQueryError** – if given *prediction_model* is *None* and this Project's *active_prediction_model* is also *None*.

create_prediction_model (*name, version*)

Creates a PredictionModel connected to a Legacy Editor Project.

Parameters

- **name** (*str*) – The new PredictionModel’s name.
- **version** (*int*) – The new PredictionModel’s version.

Returns A newly created PredictionModel.

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*)

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 (*timeout_seconds=60*)

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

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*)

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*)

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.

labeler_performance ()

Returns the labeler performances for this Project.

Returns A PaginatedCollection of LabelerPerformance objects.

labels (*datasets=None, order_by=None*)

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()

Fetch all current members for this project

Returns A `PaginatedCollection` of `ProjectMember`s

review_metrics(*net_score*)

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*)

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*)

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*.

unset_labeling_parameter_overrides (*data_rows*)

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.

upload_annotations (*name: str*, *annotations: Union[str, pathlib.Path, Iterable[Dict]]*, *validate: bool = True*) → *labelbox.schema.bulk_import_request.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*)

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

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

Raises ValueError –

- project must be setup * instructions file must end with one of “.text”, “.txt”, “.pdf”, “.html”

upsert_review_queue (*quota_factor*)

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*

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.
```


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
```

```
    refresh()
```

```
        Refreshes Task data from the server.
```

```
    wait_till_done(timeout_seconds=60)
```

```
        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.
```


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: labelbox.schema.project.Project)  
        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`: `labelbox.schema.role.Role`)

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`: `labelbox.schema.project.Project`, `role`: `labelbox.schema.role.Role`)

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.

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, RE-
        VIEW_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)
    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 notifica-
          tions 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()

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.

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.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.

PAGINATION

```
class labelbox.pagination.PaginatedCollection(client: Client, query: str, params:  
Dict[str, str], dereferencing: Dict[str,  
Any], obj_class: Type[DBObject], cur-  
sor_path: Optional[Dict[str, Any]] =  
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: Dict[str, Any], obj_class:  
Type[DBObject], cursor_path: Optional[Dict[str, Any]] = 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

ENUMS

class `labelbox.schema.enums.BulkImportRequestState` (*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 <i>BulkImportRequest.errors</i> for more information
FINISHED	Indicates the import job is no longer running. Check <i>BulkImportRequest.statuses</i> for more information

PYTHON MODULE INDEX

I

- `labelbox.client`, 1
- `labelbox.exceptions`, 49
- `labelbox.pagination`, 51
- `labelbox.schema.asset_attachment`, 5
- `labelbox.schema.asset_metadata`, 7
- `labelbox.schema.benchmark`, 9
- `labelbox.schema.bulk_import_request`, 11
- `labelbox.schema.data_row`, 13
- `labelbox.schema.dataset`, 15
- `labelbox.schema.enums`, 53
- `labelbox.schema.label`, 19
- `labelbox.schema.labeling_frontend`, 21
- `labelbox.schema.ontology`, 27
- `labelbox.schema.organization`, 29
- `labelbox.schema.prediction`, 31
- `labelbox.schema.project`, 35
- `labelbox.schema.review`, 41
- `labelbox.schema.task`, 43
- `labelbox.schema.user`, 45
- `labelbox.schema.webhook`, 47

Symbols

`__init__()` (*labelbox.client.Client* method), 1
`__init__()` (*labelbox.pagination.PaginatedCollection* method), 51

A

`active_prediction_model` (*labelbox.schema.project.Project* attribute), 36
`agreement` (*labelbox.schema.label.Label* attribute), 19
`agreement` (*labelbox.schema.prediction.Prediction* attribute), 31
`ApiLimitError`, 49
`AssetAttachment` (class in *labelbox.schema.asset_attachment*), 5
`AssetAttachment.AttachmentType` (class in *labelbox.schema.asset_attachment*), 5
`AssetMetadata` (class in *labelbox.schema.asset_metadata*), 7
`AssetMetadata.MetaType` (class in *labelbox.schema.asset_metadata*), 7
`attachment_type` (*labelbox.schema.asset_attachment.AssetAttachment* attribute), 5
`attachment_value` (*labelbox.schema.asset_attachment.AssetAttachment* attribute), 5
`attachments` (*labelbox.schema.data_row.DataRow* attribute), 13
`AuthenticationError`, 49
`AuthorizationError`, 49
`auto_audit_number_of_labels` (*labelbox.schema.project.Project* attribute), 35
`auto_audit_percentage` (*labelbox.schema.project.Project* attribute), 35
`average_agreement` (*labelbox.schema.benchmark.Benchmark* attribute), 9

B

`Benchmark` (class in *labelbox.schema.benchmark*), 9
`benchmark_agreement` (*labelbox.schema.label.Label* attribute), 19

`benchmarks` (*labelbox.schema.project.Project* attribute), 36
`bulk_delete()` (*labelbox.schema.data_row.DataRow* static method), 14
`bulk_delete()` (*labelbox.schema.label.Label* static method), 19
`BulkImportRequest` (class in *labelbox.schema.bulk_import_request*), 11
`BulkImportRequestState` (class in *labelbox.schema.enums*), 53

C

`classification_schema_count` (*labelbox.schema.ontology.Ontology* attribute), 27
`classifications` (*labelbox.schema.ontology.OntologyBuilder* attribute), 28
`classifications()` (*labelbox.schema.ontology.Ontology* method), 27
`Client` (class in *labelbox.client*), 1
`completed_count` (*labelbox.schema.benchmark.Benchmark* attribute), 9
`completion_percentage` (*labelbox.schema.task.Task* attribute), 43
`create()` (*labelbox.schema.webhook.Webhook* static method), 47
`create_attachment()` (*labelbox.schema.data_row.DataRow* method), 14
`create_benchmark()` (*labelbox.schema.label.Label* method), 19
`create_data_row()` (*labelbox.schema.dataset.Dataset* method), 15
`create_data_rows()` (*labelbox.schema.dataset.Dataset* method), 16
`create_dataset()` (*labelbox.client.Client* method), 1
`create_label()` (*labelbox.schema.project.Project*

`method`), 36
`create_metadata()` (`labelbox.schema.data_row.DataRow` method), 14
`create_prediction()` (`labelbox.schema.project.Project` method), 36
`create_prediction_model()` (`labelbox.schema.project.Project` method), 36
`create_project()` (`labelbox.client.Client` method), 1
`create_review()` (`labelbox.schema.label.Label` method), 20
`created_at` (`labelbox.schema.benchmark.Benchmark` attribute), 9
`created_at` (`labelbox.schema.bulk_import_request.BulkImportRequest` attribute), 11
`created_at` (`labelbox.schema.data_row.DataRow` attribute), 13
`created_at` (`labelbox.schema.dataset.Dataset` attribute), 15
`created_at` (`labelbox.schema.ontology.Ontology` attribute), 27
`created_at` (`labelbox.schema.organization.Organization` attribute), 29
`created_at` (`labelbox.schema.prediction.Prediction` attribute), 31
`created_at` (`labelbox.schema.prediction.PredictionModel` attribute), 33
`created_at` (`labelbox.schema.project.Project` attribute), 35
`created_at` (`labelbox.schema.review.Review` attribute), 41
`created_at` (`labelbox.schema.task.Task` attribute), 43
`created_at` (`labelbox.schema.user.User` attribute), 45
`created_at` (`labelbox.schema.webhook.Webhook` attribute), 47
`created_by` (`labelbox.schema.benchmark.Benchmark` attribute), 9
`created_by` (`labelbox.schema.bulk_import_request.BulkImportRequest` attribute), 11
`created_by` (`labelbox.schema.data_row.DataRow` attribute), 13
`created_by` (`labelbox.schema.dataset.Dataset` attribute), 15
`created_by` (`labelbox.schema.label.Label` attribute), 19
`created_by` (`labelbox.schema.ontology.Ontology` attribute), 27
`created_by` (`labelbox.schema.prediction.PredictionModel` attribute), 33
`created_by` (`labelbox.schema.project.Project` attribute), 35
`created_by` (`labelbox.schema.review.Review` attribute), 41
`created_by` (`labelbox.schema.task.Task` attribute), 43
`created_by` (`labelbox.schema.user.User` attribute), 45
`customization_options` (`labelbox.schema.labeling_frontend.LabelingFrontendOptions` attribute), 23

D

`data_row` (`labelbox.schema.label.Label` attribute), 19
`data_row` (`labelbox.schema.prediction.Prediction` attribute), 31
`data_row_for_external_id()` (`labelbox.schema.dataset.Dataset` method), 16
`data_rows` (`labelbox.schema.dataset.Dataset` attribute), 15
`data_rows_for_external_id()` (`labelbox.schema.dataset.Dataset` method), 16
`DataRow` (class in `labelbox.schema.data_row`), 13
`Dataset` (class in `labelbox.schema.dataset`), 15
`dataset` (`labelbox.schema.data_row.DataRow` attribute), 13
`datasets` (`labelbox.schema.project.Project` attribute), 35
`delete()` (`labelbox.schema.webhook.Webhook` method), 48
`description` (`labelbox.schema.dataset.Dataset` attribute), 15
`description` (`labelbox.schema.labeling_frontend.LabelingFrontendOptions` attribute), 21
`description` (`labelbox.schema.ontology.Ontology` attribute), 27
`description` (`labelbox.schema.project.Project` attribute), 35

E

`email` (`labelbox.schema.user.User` attribute), 45
`enable_model_assisted_labeling()` (`labelbox.schema.project.Project` method), 37
`error_file_url` (`labelbox.schema.bulk_import_request.BulkImportRequest` attribute), 11
`errors()` (`labelbox.schema.bulk_import_request.BulkImportRequest` property), 11
`execute()` (`labelbox.client.Client` method), 2
`export_data_rows()` (`labelbox.schema.dataset.Dataset` method), 17
`export_issues()` (`labelbox.schema.project.Project` method), 37
`export_labels()` (`labelbox.schema.project.Project` method), 37
`export_queued_data_rows()` (`labelbox.schema.project.Project` method), 37

`extend_reservations()` (*labelbox.schema.project.Project method*), 37
`external_id` (*labelbox.schema.data_row.DataRow attribute*), 13

G

`get_dataset()` (*labelbox.client.Client method*), 2
`get_datasets()` (*labelbox.client.Client method*), 2
`get_labeling_frontends()` (*labelbox.client.Client method*), 3
`get_mal_schemas()` (*in module labelbox.schema.bulk_import_request*), 12
`get_organization()` (*labelbox.client.Client method*), 3
`get_project()` (*labelbox.client.Client method*), 3
`get_projects()` (*labelbox.client.Client method*), 3
`get_roles()` (*labelbox.client.Client method*), 3
`get_user()` (*labelbox.client.Client method*), 3

I

`iframe_url_path` (*labelbox.schema.labeling_frontend.LabelingFrontend attribute*), 21
`InconsistentOntologyException`, 49
`input_file_url` (*labelbox.schema.bulk_import_request.BulkImportRequest attribute*), 11
`inputs()` (*labelbox.schema.bulk_import_request.BulkImportRequest property*), 12
`intercom_hash` (*labelbox.schema.user.User attribute*), 45
`InternalServerError`, 49
`InvalidAttributeError`, 49
`InvalidQueryError`, 49
`invite_limit()` (*labelbox.schema.organization.Organization method*), 29
`invite_user()` (*labelbox.schema.organization.Organization method*), 29
`is_benchmark_reference` (*labelbox.schema.label.Label attribute*), 19
`is_external_viewer` (*labelbox.schema.user.User attribute*), 45
`is_viewer` (*labelbox.schema.user.User attribute*), 45

L

`Label` (*class in labelbox.schema.label*), 19
`label` (*labelbox.schema.label.Label attribute*), 19
`label` (*labelbox.schema.prediction.Prediction attribute*), 31
`label` (*labelbox.schema.review.Review attribute*), 41
`labelbox.client`
 module, 1

`labelbox.exceptions`
 module, 49
`labelbox.pagination`
 module, 51
`labelbox.schema.asset_attachment`
 module, 5
`labelbox.schema.asset_metadata`
 module, 7
`labelbox.schema.benchmark`
 module, 9
`labelbox.schema.bulk_import_request`
 module, 11
`labelbox.schema.data_row`
 module, 13
`labelbox.schema.dataset`
 module, 15
`labelbox.schema.enums`
 module, 53
`labelbox.schema.label`
 module, 19
`labelbox.schema.labeling_frontend`
 module, 21
`labelbox.schema.ontology`
 module, 27
`labelbox.schema.organization`
 module, 29
`labelbox.schema.prediction`
 module, 31
`labelbox.schema.project`
 module, 35
`labelbox.schema.review`
 module, 41
`labelbox.schema.task`
 module, 43
`labelbox.schema.user`
 module, 45
`labelbox.schema.webhook`
 module, 47
`LabelboxError`, 49
`labeler_performance()` (*labelbox.schema.project.Project method*), 37
`labeling_frontend` (*labelbox.schema.labeling_frontend.LabelingFrontendOptions attribute*), 23
`labeling_frontend` (*labelbox.schema.project.Project attribute*), 36
`labeling_frontend_options` (*labelbox.schema.project.Project attribute*), 36
`labeling_parameter_overrides` (*labelbox.schema.project.Project attribute*), 36
`LabelingFrontend` (*class in labelbox.schema.labeling_frontend*), 21
`labels` (*labelbox.schema.data_row.DataRow attribute*), 13

`labels()` (*labelbox.schema.project.Project* method), 37

`last_activity` (*labelbox.schema.benchmark.Benchmark* attribute), 9

`last_activity_time` (*labelbox.schema.project.Project* attribute), 35

M

`MalformedQueryException`, 49

`MALValidationError`, 49

`media_attributes` (*labelbox.schema.data_row.DataRow* attribute), 13

`members()` (*labelbox.schema.project.Project* method), 37

`metadata` (*labelbox.schema.data_row.DataRow* attribute), 14

`module`

- `labelbox.client`, 1
- `labelbox.exceptions`, 49
- `labelbox.pagination`, 51
- `labelbox.schema.asset_attachment`, 5
- `labelbox.schema.asset_metadata`, 7
- `labelbox.schema.benchmark`, 9
- `labelbox.schema.bulk_import_request`, 11
- `labelbox.schema.data_row`, 13
- `labelbox.schema.dataset`, 15
- `labelbox.schema.enums`, 53
- `labelbox.schema.label`, 19
- `labelbox.schema.labeling_frontend`, 21
- `labelbox.schema.ontology`, 27
- `labelbox.schema.organization`, 29
- `labelbox.schema.prediction`, 31
- `labelbox.schema.project`, 35
- `labelbox.schema.review`, 41
- `labelbox.schema.task`, 43
- `labelbox.schema.user`, 45
- `labelbox.schema.webhook`, 47

N

`name` (*labelbox.schema.bulk_import_request.BulkImportRequest* attribute), 11

`name` (*labelbox.schema.dataset.Dataset* attribute), 15

`name` (*labelbox.schema.labeling_frontend.LabelingFrontend* attribute), 21

`name` (*labelbox.schema.organization.Organization* attribute), 29

`name` (*labelbox.schema.prediction.PredictionModel* attribute), 33

`name` (*labelbox.schema.project.Project* attribute), 35

`name` (*labelbox.schema.task.Task* attribute), 43

`name` (*labelbox.schema.user.User* attribute), 45

`NetworkError`, 50

`nickname` (*labelbox.schema.user.User* attribute), 45

`normalized` (*labelbox.schema.ontology.Ontology* attribute), 27

`number_of_labels` (*labelbox.schema.project.LabelingParameterOverride* attribute), 25

O

`object_schema_count` (*labelbox.schema.ontology.Ontology* attribute), 27

`Ontology` (class in *labelbox.schema.ontology*), 27

`ontology` (*labelbox.schema.project.Project* attribute), 36

`OntologyBuilder` (class in *labelbox.schema.ontology*), 27

`Organization` (class in *labelbox.schema.organization*), 29

`organization` (*labelbox.schema.data_row.DataRow* attribute), 13

`organization` (*labelbox.schema.dataset.Dataset* attribute), 15

`organization` (*labelbox.schema.labeling_frontend.LabelingFrontendOptions* attribute), 23

`organization` (*labelbox.schema.prediction.Prediction* attribute), 31

`organization` (*labelbox.schema.prediction.PredictionModel* attribute), 33

`organization` (*labelbox.schema.project.Project* attribute), 35

`organization` (*labelbox.schema.review.Review* attribute), 41

`organization` (*labelbox.schema.task.Task* attribute), 43

`organization` (*labelbox.schema.user.User* attribute), 45

P

`PaginationCollection` (class in *labelbox.pagination*), 51

`parse_classification()` (in module *labelbox.schema.bulk_import_request*), 12

`picture` (*labelbox.schema.user.User* attribute), 45

`Prediction` (class in *labelbox.schema.prediction*), 31

`prediction_model` (*labelbox.schema.prediction.Prediction* attribute), 31

`predictions` (*labelbox.schema.data_row.DataRow* attribute), 14

- [predictions \(labelbox.schema.project.Project attribute\), 36](#)
[priority \(labelbox.schema.project.LabelingParameterOverride attribute\), 25](#)
[Project \(class in labelbox.schema.project\), 35](#)
[project \(labelbox.schema.bulk_import_request.BulkImportRequest attribute\), 11](#)
[project \(labelbox.schema.label.Label attribute\), 19](#)
[project \(labelbox.schema.labeling_frontend.LabelingFrontendOptions attribute\), 23](#)
[project \(labelbox.schema.prediction.Prediction attribute\), 31](#)
[project \(labelbox.schema.review.Review attribute\), 41](#)
[ProjectMember \(class in labelbox.schema.project\), 39](#)
[projects \(labelbox.schema.dataset.Dataset attribute\), 15](#)
[projects \(labelbox.schema.labeling_frontend.LabelingFrontend attribute\), 21](#)
[projects \(labelbox.schema.ontology.Ontology attribute\), 27](#)
[projects \(labelbox.schema.organization.Organization attribute\), 29](#)
[projects \(labelbox.schema.user.User attribute\), 45](#)
- ## R
- [reference_label \(labelbox.schema.benchmark.Benchmark attribute\), 9](#)
[refresh\(\) \(labelbox.schema.bulk_import_request.BulkImportRequest method\), 12](#)
[refresh\(\) \(labelbox.schema.task.Task method\), 43](#)
[remove_from_project\(\) \(labelbox.schema.user.User method\), 45](#)
[remove_user\(\) \(labelbox.schema.organization.Organization method\), 30](#)
[ResourceNotFoundError, 50](#)
[Review \(class in labelbox.schema.review\), 41](#)
[Review.NetScore \(class in labelbox.schema.review\), 41](#)
[review_metrics\(\) \(labelbox.schema.project.Project method\), 38](#)
[reviews \(labelbox.schema.label.Label attribute\), 19](#)
[reviews \(labelbox.schema.project.Project attribute\), 35](#)
[row_data \(labelbox.schema.data_row.DataRow attribute\), 13](#)
- ## S
- [score \(labelbox.schema.review.Review attribute\), 41](#)
[seconds_to_label \(labelbox.schema.label.Label attribute\), 19](#)
[set_labeling_parameter_overrides\(\) \(labelbox.schema.project.Project method\), 38](#)
[setup_complete \(labelbox.schema.project.Project attribute\), 35](#)
[state \(labelbox.schema.bulk_import_request.BulkImportRequest attribute\), 11](#)
[status \(labelbox.schema.task.Task attribute\), 43](#)
[status \(labelbox.schema.webhook.Webhook attribute\), 47](#)
[status_file_url \(labelbox.schema.bulk_import_request.BulkImportRequest attribute\), 11](#)
[statuses\(\) \(labelbox.schema.bulk_import_request.BulkImportRequest property\), 12](#)
- ## T
- [Task \(class in labelbox.schema.task\), 43](#)
[TimeoutError, 50](#)
[tools \(labelbox.schema.ontology.OntologyBuilder attribute\), 28](#)
[tools\(\) \(labelbox.schema.ontology.Ontology method\), 27](#)
[topics \(labelbox.schema.webhook.Webhook attribute\), 47](#)
- ## U
- [update_labeling_parameter_overrides\(\) \(labelbox.schema.project.Project method\), 39](#)
[update\(\) \(labelbox.schema.webhook.Webhook method\), 48](#)
[update_org_role\(\) \(labelbox.schema.user.User method\), 46](#)
[updated_at \(labelbox.schema.data_row.DataRow attribute\), 13](#)
[updated_at \(labelbox.schema.dataset.Dataset attribute\), 15](#)
[updated_at \(labelbox.schema.ontology.Ontology attribute\), 27](#)
[updated_at \(labelbox.schema.organization.Organization attribute\), 29](#)
[updated_at \(labelbox.schema.prediction.Prediction attribute\), 31](#)
[updated_at \(labelbox.schema.prediction.PredictionModel attribute\), 33](#)
[updated_at \(labelbox.schema.project.Project attribute\), 35](#)
[updated_at \(labelbox.schema.review.Review attribute\), 41](#)
[updated_at \(labelbox.schema.task.Task attribute\), 43](#)
[updated_at \(labelbox.schema.user.User attribute\), 45](#)

`updated_at` (*labelbox.schema.webhook.Webhook* attribute), 47
`upload_annotations()` (*labelbox.schema.project.Project* method), 39
`upsert_instructions()` (*labelbox.schema.project.Project* method), 39
`upsert_project_role()` (*labelbox.schema.user.User* method), 46
`upsert_review_queue()` (*labelbox.schema.project.Project* method), 39
`url` (*labelbox.schema.webhook.Webhook* attribute), 47
`User` (class in *labelbox.schema.user*), 45
`users` (*labelbox.schema.organization.Organization* attribute), 29
`UuidError`, 50

V

`ValidationFailedError`, 50
`version` (*labelbox.schema.prediction.PredictionModel* attribute), 33

W

`wait_till_done()` (*labelbox.schema.task.Task* method), 43
`wait_until_done()` (*labelbox.schema.bulk_import_request.BulkImportRequest* method), 12
`Webhook` (class in *labelbox.schema.webhook*), 47
`Webhook.Status` (class in *labelbox.schema.webhook*), 47
`Webhook.Topic` (class in *labelbox.schema.webhook*), 47
`webhooks` (*labelbox.schema.organization.Organization* attribute), 29
`webhooks` (*labelbox.schema.project.Project* attribute), 36