

---

# Python SDK reference

*Release 3.36.0*

**Labelbox**

**Jan 23, 2023**



**CONTENTS:**

<b>1</b>	<b>Client</b>	<b>1</b>
<b>2</b>	<b>AssetAttachment</b>	<b>11</b>
<b>3</b>	<b>Benchmark</b>	<b>13</b>
<b>4</b>	<b>BulkImportRequest</b>	<b>15</b>
<b>5</b>	<b>DataRow</b>	<b>17</b>
<b>6</b>	<b>Dataset</b>	<b>21</b>
<b>7</b>	<b>Label</b>	<b>25</b>
<b>8</b>	<b>LabelingFrontend</b>	<b>27</b>
<b>9</b>	<b>LabelingFrontendOptions</b>	<b>29</b>
<b>10</b>	<b>LabelingParameterOverride</b>	<b>31</b>
<b>11</b>	<b>Ontology</b>	<b>33</b>
<b>12</b>	<b>Organization</b>	<b>35</b>
<b>13</b>	<b>Project</b>	<b>37</b>
<b>14</b>	<b>Review</b>	<b>43</b>
<b>15</b>	<b>Task</b>	<b>45</b>
<b>16</b>	<b>User</b>	<b>47</b>
<b>17</b>	<b>Webhook</b>	<b>49</b>
<b>18</b>	<b>Exceptions</b>	<b>51</b>
<b>19</b>	<b>Pagination</b>	<b>53</b>
<b>20</b>	<b>Enums</b>	<b>55</b>
<b>21</b>	<b>ModelRun</b>	<b>57</b>
<b>22</b>	<b>Model</b>	<b>61</b>

<b>23 DataRowMetadata</b>	<b>63</b>
<b>24 AnnotationImport</b>	<b>67</b>
<b>25 Batch</b>	<b>73</b>
<b>26 ResourceTag</b>	<b>75</b>
<b>27 Slice</b>	<b>77</b>
<b>28 CatalogSlice</b>	<b>79</b>
<b>Python Module Index</b>	<b>81</b>
<b>Index</b>	<b>83</b>

## CLIENT

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

Bases: object

A Labelbox client.

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

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

Creates and initializes a Labelbox Client.

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

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

### Parameters

- **api\_key** (*str*) – API key. If None, the key is obtained from the “LABELBOX\_API\_KEY” environment variable.
- **endpoint** (*str*) – URL of the Labelbox server to connect to.
- **enable\_experimental** (*bool*) – Indicates whether or not to use experimental features
- **app\_url** (*str*) – host url for all links to the web app

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

```
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': 'cl7tpjzw30031ka6g4evqdfof', '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 that 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 of global keys that were successfully cleared.

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

### 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'}]
```

**create\_dataset**(*iam\_integration='DEFAULT', \*\*kwargs*) → *labelbox.schema.dataset.Dataset*

Creates a Dataset object on the server.

Attribute values are passed as keyword arguments.

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

#### Parameters

- **iam\_integration** (*IAMIntegration*) – Uses the default integration. Optionally specify another integration or set as None to not use delegated access
- **\*\*kwargs** – Keyword arguments with Dataset attribute values.

**Returns** A new Dataset object.

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

#### **create\_feature\_schema**(*normalized*)

**Creates a feature schema from normalized data.**

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

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

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

**Subclasses are also supported**

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

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

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

**Returns** The created FeatureSchema.

#### **create\_model**(*name, ontology\_id*) → *labelbox.schema.model.Model*

Creates a Model object on the server.

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

#### Parameters

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

**Returns** A new Model object.

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

**create\_ontology**(*name, normalized, 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.

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

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

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

#### Parameters

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

**Returns** dict, parsed JSON response.

#### Raises

- *labelbox.exceptions.AuthenticationError* – If authentication failed.
- *labelbox.exceptions.InvalidQueryError* – If *query* is not syntactically or semantically valid (checked server-side).
- *labelbox.exceptions.ApiLimitError* – If the server API limit was exceeded. See “How to import data” in the online documentation to see API limits.
- *labelbox.exceptions.TimeoutError* – If response was not received in *timeout* seconds.
- *labelbox.exceptions.NetworkError* – If an unknown error occurred most likely due to connection issues.

- **`labelbox.exceptions.LabelboxError`** – If an unknown error of any kind occurred.
- **`ValueError`** – If query and data are both None.

**`get_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.

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

Fetches all the models the user has access to.

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

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

**Returns** An iterable of Models (typically a *PaginatedCollection*).

**get\_ontologies**(*name\_contains*) → *labelbox.pagination.PaginatedCollection*

Fetches all ontologies with names that match the *name\_contains* string.

**Parameters** *name\_contains* (*str*) – the string to search ontology names by

**Returns** *PaginatedCollection* of Ontologies with names that match *name\_contains*

**get\_ontology**(*ontology\_id*) → *labelbox.schema.ontology.Ontology*

Fetches an Ontology by id.

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

**Returns** Ontology

**get\_organization**() → *labelbox.schema.organization.Organization*

Gets the Organization DB object of the current user.

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

**get\_project**(*project\_id*)

Gets a single Project with the given ID.

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

**Parameters** *project\_id* (*str*) – Unique ID of the Project.

**Returns** The sought Project.

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

**get\_projects**(*where=None*) → List[labelbox.schema.project.Project]

Fetches all the projects the user has access to.

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

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

**Returns** An iterable of Projects (typically a PaginatedCollection).

**get\_roles**() → List[labelbox.schema.role.Role]

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

**Return type** Roles

**get\_user**() → labelbox.schema.user.User

Gets the current User database object.

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



## **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.





## 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`

**delete()**  $\rightarrow$  `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

## 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*) → *AssetAttachment*

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

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

**Parameters**

- **attachment\_type** (*str*) – Asset attachment type, must be one of: VIDEO, IMAGE, TEXT, IMAGE\_OVERLAY (*AssetAttachment.AttachmentType*)
- **attachment\_value** (*str*) – Asset attachment value.

**Returns** *AssetAttachment* DB object.

**Raises** **ValueError** – asset\_type must be one of the supported types.

**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.





## 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.



**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.

## 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.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

## 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’: ‘ffff’}

**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 Meaning 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 themselves *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



## 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

**batches()** → *labelbox.pagination.PaginatedCollection*

Fetch all batches that belong to this project

**Returns** A `PaginatedCollection` of `Batch`'es

**bulk\_import\_requests()** → *labelbox.pagination.PaginatedCollection*

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

**create\_batch**(*name: str, data\_rows: List[Union[str, labelbox.schema.data\_row.DataRow]], priority: int = 5, consensus\_settings: Optional[Dict[str, float]] = None*)

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

**Parameters**

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

**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.

**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

**review\_metrics**(*net\_score*) → int

Returns this Project's review metrics.

**Parameters** **net\_score** (*None or Review.NetScore*) – Indicates desired metric.

**Returns** int, aggregation count of reviews for given *net\_score*.

**set\_labeling\_parameter\_overrides**(*data*) → bool

Adds labeling parameter overrides to this project.

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

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

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

**Priority:**

- **Data will be labeled in priority order.**
  - A lower number priority is labeled first.
  - Minimum priority is 1.
- **Priority is not the queue position.**
  - The position is determined by the relative priority.
  - **E.g.** [(data\_row\_1, 5,1), (data\_row\_2, 2,1), (data\_row\_3, 10,1)] will be assigned in the following order: [data\_row\_2, data\_row\_1, data\_row\_3]
- Datarows with parameter overrides will appear before datarows without overrides.
- **The priority only effects items in the queue.**

- Assigning a priority will not automatically add the item back into the queue.

**Number of labels:**

- **The number of times a data row should be labeled.**
  - Creates duplicate data rows in a project (one for each number of labels).
- **New duplicated data rows will be added to the queue.**
  - Already labeled duplicates will not be sent back to the queue.
- **The queue will never assign the same datarow to a single labeler more than once.**
  - **If the number of labels is greater than the number of labelers working on a project then the extra items will remain in the queue (this can be fixed by removing the override at any time).**
- Setting this to 1 will result in the default behavior (no duplicates).

**Returns** bool, indicates if the operation was a success.

**setup**(*labeling\_frontend*, *labeling\_frontend\_options*) → None

Finalizes the Project setup.

**Parameters**

- **labeling\_frontend** ([LabelingFrontend](#)) – Which UI to use to label the data.
- **labeling\_frontend\_options** (*dict or str*) – Labeling frontend options, a.k.a. project ontology. If given a *dict* it will be converted to *str* using *json.dumps*.

**setup\_editor**(*ontology*) → None

Sets up the project using the Pictor editor.

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

**unset\_labeling\_parameter\_overrides**(*data\_rows*) → bool

Removes labeling parameter overrides to this project.

- This will remove unlabeled duplicates in the queue.

**Parameters** *data\_rows* (*iterable*) – An iterable of DataRows.

**Returns** bool, indicates if the operation was a success.

**update**(\*\**kwargs*)

Updates this 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

## 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`

**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()**  $\rightarrow$  `None`

Refreshes Task data from the server.

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

Fetch the result for an import task.

**wait\_till\_done**(*timeout\_seconds=300*)  $\rightarrow$  `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.



## 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.

## 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.

## 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.



## 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



## ENUMS

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

Bases: `enum.Enum`

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

State	Description
RUN- NING	Indicates that the import job is not done yet.
FAILED	Indicates the import job failed. Check <i>AnnotationImport.errors</i> for more information
FIN- ISHED	Indicates the import job is no longer running. Check <i>AnnotationImport.statuses</i> 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 using MEA continue using BulkImportRequest. AnnotationImports are in beta and will change soon.

State	Description
RUN- NING	Indicates that the import job is not done yet.
FAILED	Indicates the import job failed. Check <i>BulkImportRequest.errors</i> for more information
FIN- ISHED	Indicates the import job is no longer running. Check <i>BulkImportRequest.statuses</i> 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



## MODEL RUN

**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]]*) → *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 *time-out\_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, timeout\_seconds=3600*)

Adds data rows to a Model Run without any associated labels :param data\_row\_ids: data row ids to add to mea :type data\_row\_ids: list :param timeout\_seconds: Max waiting time, in seconds. :type timeout\_seconds: float

**Returns** ID of newly generated async task

**upsert\_labels**(*label\_ids, timeout\_seconds=3600*)

Adds data rows and labels to a Model Run :param label\_ids: label ids to insert :type label\_ids: list :param timeout\_seconds: Max waiting time, in seconds. :type timeout\_seconds: float

**Returns** ID of newly generated async task

**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 associated 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





## 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.



## 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 specifying 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

## ANNOTATIONIMPORT

**class** `labelbox.schema.annotation_import.AnnotationImport`(*client, field\_values*)

Bases: `labelbox.orm.db_object.DbObject`

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

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

See *AnnotationImport.statuses* for more details.

- This information will expire after 24 hours.

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

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

- This information will expire after 24 hours.

**refresh()** → None

Synchronizes values of all fields with the database.

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

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

See below table for more details

- This information will expire after 24 hours.

**wait\_until\_done**(*sleep\_time\_seconds: int = 10, show\_progress: bool = False*) → None

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

**class** `labelbox.schema.annotation_import.LabelImport`(*client, field\_values*)

Bases: `labelbox.schema.annotation_import.AnnotationImport`

**classmethod create\_from\_file**(*client: labelbox.client.Client, project\_id: str, name: str, path: str*) → *labelbox.schema.annotation\_import.LabelImport*

Create a label import job from a file of annotations

**Parameters**

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

**Returns** LabelImport

**classmethod** `create_from_objects`(*client*: [labelbox.client.Client](#), *project\_id*: str, *name*: str, *labels*: List[Dict[str, Any]]) → [labelbox.schema.annotation\\_import.LabelImport](#)

Create a label import job from an in memory dictionary

**Parameters**

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

**Returns** LabelImport

**classmethod** `create_from_url`(*client*: [labelbox.client.Client](#), *project\_id*: str, *name*: str, *url*: str) → [labelbox.schema.annotation\\_import.LabelImport](#)

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

**Parameters**

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

**Returns** LabelImport

**classmethod** `from_name`(*client*: [labelbox.client.Client](#), *project\_id*: str, *name*: str, *as\_json*: bool = False) → [labelbox.schema.annotation\\_import.LabelImport](#)

Retrieves an label import job.

**Parameters**

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

**Returns** LabelImport

**property** `parent_id`: str

Identifier for this import. Used to refresh the status

**class** `labelbox.schema.annotation_import.MALPredictionImport`(*client*, *field\_values*)

Bases: [labelbox.schema.annotation\\_import.AnnotationImport](#)

**classmethod** `create_from_file`(*client*: [labelbox.client.Client](#), *project\_id*: str, *name*: str, *path*: str) → [labelbox.schema.annotation\\_import.MALPredictionImport](#)

Create an MAL prediction import job from a file of annotations

**Parameters**

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



**Returns** MALPredictionImport

**classmethod** `create_from_objects`(*client*: [labelbox.client.Client](#), *project\_id*: *str*, *name*: *str*, *predictions*: *List[Dict[str, Any]]*) → [labelbox.schema.annotation\\_import.MALPredictionImport](#)

Create an MAL prediction import job from an in memory dictionary

**Parameters**

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

**Returns** MALPredictionImport

**classmethod** `create_from_url`(*client*: [labelbox.client.Client](#), *project\_id*: *str*, *name*: *str*, *url*: *str*) → [labelbox.schema.annotation\\_import.MALPredictionImport](#)

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

**Parameters**

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

**Returns** MALPredictionImport

**classmethod** `from_name`(*client*: [labelbox.client.Client](#), *project\_id*: *str*, *name*: *str*, *as\_json*: *bool* = *False*) → [labelbox.schema.annotation\\_import.MALPredictionImport](#)

Retrieves an MAL import job.

**Parameters**

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

**Returns** MALPredictionImport

**property** `parent_id`: *str*

Identifier for this import. Used to refresh the status

**class** `labelbox.schema.annotation_import.MEAPredictionImport`(*client*, *field\_values*)

Bases: [labelbox.schema.annotation\\_import.AnnotationImport](#)

**classmethod** `create_from_file`(*client*: [labelbox.client.Client](#), *model\_run\_id*: *str*, *name*: *str*, *path*: *str*) → [labelbox.schema.annotation\\_import.MEAPredictionImport](#)

Create an MEA prediction import job from a file of annotations

**Parameters**

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

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

**Returns** MEAPredictionImport

**classmethod** **create\_from\_objects**(*client*: [labelbox.client.Client](#), *model\_run\_id*: *str*, *name*, *predictions*: *List[Dict[str, Any]]*) → [labelbox.schema.annotation\\_import.MEAPredictionImport](#)

Create an MEA prediction import job from an in memory dictionary

**Parameters**

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

**Returns** MEAPredictionImport

**classmethod** **create\_from\_url**(*client*: [labelbox.client.Client](#), *model\_run\_id*: *str*, *name*: *str*, *url*: *str*) → [labelbox.schema.annotation\\_import.MEAPredictionImport](#)

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

**Parameters**

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

**Returns** MEAPredictionImport

**classmethod** **from\_name**(*client*: [labelbox.client.Client](#), *model\_run\_id*: *str*, *name*: *str*, *as\_json*: *bool* = *False*) → [labelbox.schema.annotation\\_import.MEAPredictionImport](#)

Retrieves an MEA import job.

**Parameters**

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

**Returns** MEAPredictionImport

**property** **parent\_id**: *str*

Identifier for this import. Used to refresh the status

**class** [labelbox.schema.annotation\\_import.METoMALPredictionImport](#)(*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.METoMALPredictionImport](#)

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



## 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

## RESOURCE TAG

```
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
```





## 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
```



## 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

|

labelbox.client, 1  
labelbox.exceptions, 51  
labelbox.pagination, 53  
labelbox.schema.annotation\_import, 67  
labelbox.schema.asset\_attachment, 11  
labelbox.schema.batch, 73  
labelbox.schema.benchmark, 13  
labelbox.schema.bulk\_import\_request, 15  
labelbox.schema.data\_row, 17  
labelbox.schema.data\_row\_metadata, 63  
labelbox.schema.dataset, 21  
labelbox.schema.enums, 55  
labelbox.schema.label, 25  
labelbox.schema.labeling\_frontend, 27  
labelbox.schema.model, 61  
labelbox.schema.model\_run, 57  
labelbox.schema.ontology, 33  
labelbox.schema.organization, 35  
labelbox.schema.project, 37  
labelbox.schema.resource\_tag, 75  
labelbox.schema.review, 43  
labelbox.schema.slice, 79  
labelbox.schema.task, 45  
labelbox.schema.user, 47  
labelbox.schema.webhook, 49



## Symbols

`__init__()` (*labelbox.client.Client* method), 1  
`__init__()` (*labelbox.pagination.PaginatedCollection* method), 53

## A

`add_predictions()` (*labelbox.schema.model\_run.ModelRun* method), 57  
`agreement` (*labelbox.schema.label.Label* attribute), 25  
`AnnotationImport` (class in *labelbox.schema.annotation\_import*), 67  
`AnnotationImportState` (class in *labelbox.schema.enums*), 55  
`ApiLimitError`, 51  
`AssetAttachment` (class in *labelbox.schema.asset\_attachment*), 11  
`AssetAttachment.AttachmentType` (class in *labelbox.schema.asset\_attachment*), 11  
`assign_global_keys_to_data_rows()` (*labelbox.client.Client* method), 1  
`attachment_type` (*labelbox.schema.asset\_attachment.AssetAttachment* attribute), 11  
`attachment_value` (*labelbox.schema.asset\_attachment.AssetAttachment* attribute), 11  
`attachments` (*labelbox.schema.data\_row.DataRow* attribute), 18  
`AuthenticationError`, 51  
`AuthorizationError`, 51  
`auto_audit_number_of_labels` (*labelbox.schema.project.Project* attribute), 37  
`auto_audit_percentage` (*labelbox.schema.project.Project* attribute), 37  
`average_agreement` (*labelbox.schema.benchmark.Benchmark* attribute), 13

## B

`Batch` (class in *labelbox.schema.batch*), 73

`batches()` (*labelbox.schema.project.Project* method), 38  
`Benchmark` (class in *labelbox.schema.benchmark*), 13  
`benchmark_agreement` (*labelbox.schema.label.Label* attribute), 25  
`benchmarks` (*labelbox.schema.project.Project* attribute), 38  
`bulk_delete()` (*labelbox.schema.data\_row.DataRow* static method), 18  
`bulk_delete()` (*labelbox.schema.data\_row\_metadata.DataRowMetadataOntology* method), 63  
`bulk_delete()` (*labelbox.schema.label.Label* static method), 25  
`bulk_export()` (*labelbox.schema.data\_row\_metadata.DataRowMetadataOntology* method), 63  
`bulk_import_requests()` (*labelbox.schema.project.Project* method), 38  
`bulk_upsert()` (*labelbox.schema.data\_row\_metadata.DataRowMetadataOntology* method), 63  
`BulkImportRequest` (class in *labelbox.schema.bulk\_import\_request*), 15  
`BulkImportRequestState` (class in *labelbox.schema.enums*), 55

## C

`CatalogSlice` (class in *labelbox.schema.slice*), 79  
`classification_schema_count` (*labelbox.schema.ontology.Ontology* attribute), 33  
`classifications` (*labelbox.schema.ontology.OntologyBuilder* attribute), 34  
`classifications()` (*labelbox.schema.ontology.Ontology* method), 33  
`clear_global_keys()` (*labelbox.client.Client* method), 2  
`Client` (class in *labelbox.client*), 1  
`CollectionJobStatus` (class in *label-*

<code>box.schema.enums</code> ), 55	<code>create_from_url()</code> (labelbox.schema.annotation_import.MEAPredictionImport class method), 70
<code>color</code> (labelbox.schema.resource_tag.ResourceTag attribute), 75	<code>create_model()</code> (labelbox.client.Client method), 3
<code>completed_count</code> (labelbox.schema.benchmark.Benchmark attribute), 13	<code>create_model_run()</code> (labelbox.schema.model.Model method), 61
<code>completion_percentage</code> (labelbox.schema.task.Task attribute), 45	<code>create_ontology()</code> (labelbox.client.Client method), 4
<code>ConfidenceNotSupportedException</code> , 51	<code>create_ontology_from_feature_schemas()</code> (labelbox.client.Client method), 4
<code>create()</code> (labelbox.schema.webhook.Webhook static method), 49	<code>create_project()</code> (labelbox.client.Client method), 4
<code>create_attachment()</code> (labelbox.schema.data_row.DataRow method), 18	<code>create_resource_tag()</code> (labelbox.schema.organization.Organization method), 35
<code>create_batch()</code> (labelbox.schema.project.Project method), 38	<code>create_review()</code> (labelbox.schema.label.Label method), 26
<code>create_benchmark()</code> (labelbox.schema.label.Label method), 26	<code>create_schema()</code> (labelbox.schema.data_row_metadata.DataRowMetadataOntology method), 64
<code>create_data_row()</code> (labelbox.schema.dataset.Dataset method), 21	<code>created_at</code> (labelbox.schema.batch.Batch attribute), 73
<code>create_data_rows()</code> (labelbox.schema.dataset.Dataset method), 22	<code>created_at</code> (labelbox.schema.benchmark.Benchmark attribute), 13
<code>create_data_rows_sync()</code> (labelbox.schema.dataset.Dataset method), 22	<code>created_at</code> (labelbox.schema.bulk_import_request.BulkImportRequest attribute), 15
<code>create_dataset()</code> (labelbox.client.Client method), 2	<code>created_at</code> (labelbox.schema.data_row.DataRow attribute), 17
<code>create_feature_schema()</code> (labelbox.client.Client method), 3	<code>created_at</code> (labelbox.schema.dataset.Dataset attribute), 21
<code>create_for_model_run_data_rows()</code> (labelbox.schema.annotation_import.MEAToMALPredictionImport class method), 70	<code>created_at</code> (labelbox.schema.ontology.Ontology attribute), 33
<code>create_from_file()</code> (labelbox.schema.annotation_import.LabelImport class method), 67	<code>created_at</code> (labelbox.schema.organization.Organization attribute), 35
<code>create_from_file()</code> (labelbox.schema.annotation_import.MALPredictionImport class method), 68	<code>created_at</code> (labelbox.schema.project.Project attribute), 37
<code>create_from_file()</code> (labelbox.schema.annotation_import.MEAPredictionImport class method), 69	<code>created_at</code> (labelbox.schema.review.Review attribute), 43
<code>create_from_objects()</code> (labelbox.schema.annotation_import.LabelImport class method), 68	<code>created_at</code> (labelbox.schema.slice.Slice attribute), 77
<code>create_from_objects()</code> (labelbox.schema.annotation_import.MALPredictionImport class method), 69	<code>created_at</code> (labelbox.schema.task.Task attribute), 45
<code>create_from_objects()</code> (labelbox.schema.annotation_import.MEAPredictionImport class method), 70	<code>created_at</code> (labelbox.schema.user.User attribute), 47
<code>create_from_url()</code> (labelbox.schema.annotation_import.LabelImport class method), 68	<code>created_at</code> (labelbox.schema.webhook.Webhook attribute), 49
<code>create_from_url()</code> (labelbox.schema.annotation_import.MALPredictionImport class method), 69	<code>created_by</code> (labelbox.schema.batch.Batch attribute), 73
	<code>created_by</code> (labelbox.schema.benchmark.Benchmark attribute), 13
	<code>created_by</code> (labelbox.schema.bulk_import_request.BulkImportRequest attribute), 15
	<code>created_by</code> (labelbox.schema.data_row.DataRow attribute), 18
	<code>created_by</code> (labelbox.schema.dataset.Dataset attribute), 21
	<code>created_by</code> (labelbox.schema.label.Label attribute), 25
	<code>created_by</code> (labelbox.schema.ontology.Ontology attribute), 33
	<code>created_by</code> (labelbox.schema.project.Project attribute), 37



`created_by` (*labelbox.schema.review.Review* attribute), 43

`created_by` (*labelbox.schema.task.Task* attribute), 45

`created_tasks` (*labelbox.schema.user.User* attribute), 47

`customization_options` (*labelbox.schema.labeling\_frontend.LabelingFrontendOptions* attribute), 29

## D

`data_row` (*labelbox.schema.label.Label* attribute), 25

`data_row_for_external_id()` (*labelbox.schema.dataset.Dataset* method), 23

`data_rows` (*labelbox.schema.dataset.Dataset* attribute), 21

`data_rows_for_external_id()` (*labelbox.schema.dataset.Dataset* method), 23

`DataRow` (class in *labelbox.schema.data\_row*), 17

`DataRowMetadataKind` (class in *labelbox.schema.data\_row\_metadata*), 63

`DataRowMetadataOntology` (class in *labelbox.schema.data\_row\_metadata*), 63

`Dataset` (class in *labelbox.schema.dataset*), 21

`dataset` (*labelbox.schema.data\_row.DataRow* attribute), 17

`datasets` (*labelbox.schema.project.Project* attribute), 37

`DataSplit` (class in *labelbox.schema.model\_run*), 57

`delete()` (*labelbox.schema.asset\_attachment.AssetAttachment* method), 11

`delete()` (*labelbox.schema.batch.Batch* method), 73

`delete()` (*labelbox.schema.bulk\_import\_request.BulkImportRequest* method), 15

`delete()` (*labelbox.schema.model.Model* method), 61

`delete()` (*labelbox.schema.model\_run.ModelRun* method), 57

`delete()` (*labelbox.schema.webhook.Webhook* method), 50

`delete_labels()` (*labelbox.schema.batch.Batch* method), 73

`delete_model_run_data_rows()` (*labelbox.schema.model\_run.ModelRun* method), 57

`delete_schema()` (*labelbox.schema.data\_row\_metadata.DataRowMetadataOntology* method), 64

`deleted` (*labelbox.schema.batch.Batch* attribute), 73

`description` (*labelbox.schema.dataset.Dataset* attribute), 21

`description` (*labelbox.schema.labeling\_frontend.LabelingFrontend* attribute), 27

`description` (*labelbox.schema.ontology.Ontology* attribute), 33

`description` (*labelbox.schema.project.Project* attribute), 37

`description` (*labelbox.schema.slice.Slice* attribute), 77

## E

`email` (*labelbox.schema.user.User* attribute), 47

`enable_model_assisted_labeling()` (*labelbox.schema.project.Project* method), 38

`error_file_url` (*labelbox.schema.bulk\_import\_request.BulkImportRequest* attribute), 15

`errors` (*labelbox.schema.annotation\_import.AnnotationImport* property), 67

`errors` (*labelbox.schema.bulk\_import\_request.BulkImportRequest* property), 15

`errors` (*labelbox.schema.task.Task* property), 45

`execute()` (*labelbox.client.Client* method), 5

`export_data_rows()` (*labelbox.schema.batch.Batch* method), 73

`export_data_rows()` (*labelbox.schema.dataset.Dataset* method), 23

`export_issues()` (*labelbox.schema.project.Project* method), 39

`export_labels()` (*labelbox.schema.model\_run.ModelRun* method), 57

`export_labels()` (*labelbox.schema.project.Project* method), 39

`export_queued_data_rows()` (*labelbox.schema.project.Project* method), 39

`extend_reservations()` (*labelbox.schema.project.Project* method), 39

`external_id` (*labelbox.schema.data\_row.DataRow* attribute), 17

## F

`failed_data_rows` (*labelbox.schema.task.Task* property), 45

`FeatureSchema` (class in *labelbox.schema.ontology*), 33

`FeatureSchemaId` (in module *labelbox.schema.ontology*), 33

`filter` (*labelbox.schema.slice.Slice* attribute), 77

`from_name()` (*labelbox.schema.annotation\_import.LabelImport* class method), 68

`from_name()` (*labelbox.schema.annotation\_import.MALPredictionImport* class method), 69

`from_name()` (*labelbox.schema.annotation\_import.MEAPredictionImport* class method), 70

`from_name()` (*labelbox.schema.annotation\_import.MEAToMALPredictionImport* class method), 71

## G

`get_by_name()` (*labelbox.schema.data\_row\_metadata.DataRowMetadataOntology* method), 64

[get\\_catalog\\_slice\(\)](#) (*labelbox.client.Client* method), 6  
[get\\_config\(\)](#) (*labelbox.schema.model\_run.ModelRun* method), 57  
[get\\_data\\_row\(\)](#) (*labelbox.client.Client* method), 6  
[get\\_data\\_row\\_ids\(\)](#) (*labelbox.schema.slice.CatalogSlice* method), 79  
[get\\_data\\_row\\_ids\\_for\\_external\\_ids\(\)](#) (*labelbox.client.Client* method), 6  
[get\\_data\\_row\\_ids\\_for\\_global\\_keys\(\)](#) (*labelbox.client.Client* method), 6  
[get\\_data\\_row\\_metadata\\_ontology\(\)](#) (*labelbox.client.Client* method), 6  
[get\\_dataset\(\)](#) (*labelbox.client.Client* method), 6  
[get\\_datasets\(\)](#) (*labelbox.client.Client* method), 7  
[get\\_default\\_iam\\_integration\(\)](#) (*labelbox.schema.organization.Organization* method), 35  
[get\\_feature\\_schema\(\)](#) (*labelbox.client.Client* method), 7  
[get\\_feature\\_schemas\(\)](#) (*labelbox.client.Client* method), 7  
[get\\_iam\\_integrations\(\)](#) (*labelbox.schema.organization.Organization* method), 36  
[get\\_labeling\\_frontends\(\)](#) (*labelbox.client.Client* method), 7  
[get\\_mal\\_schemas\(\)](#) (in module *labelbox.schema.bulk\_import\_request*), 16  
[get\\_many\(\)](#) (*labelbox.pagination.PaginatedCollection* method), 53  
[get\\_model\(\)](#) (*labelbox.client.Client* method), 7  
[get\\_model\\_run\(\)](#) (*labelbox.client.Client* method), 8  
[get\\_models\(\)](#) (*labelbox.client.Client* method), 8  
[get\\_one\(\)](#) (*labelbox.pagination.PaginatedCollection* method), 53  
[get\\_ontologies\(\)](#) (*labelbox.client.Client* method), 8  
[get\\_ontology\(\)](#) (*labelbox.client.Client* method), 8  
[get\\_organization\(\)](#) (*labelbox.client.Client* method), 8  
[get\\_project\(\)](#) (*labelbox.client.Client* method), 8  
[get\\_projects\(\)](#) (*labelbox.client.Client* method), 8  
[get\\_queue\\_mode\(\)](#) (*labelbox.schema.project.Project* method), 39  
[get\\_resource\\_tags\(\)](#) (*labelbox.schema.organization.Organization* method), 36  
[get\\_roles\(\)](#) (*labelbox.client.Client* method), 9  
[get\\_user\(\)](#) (*labelbox.client.Client* method), 9  
[get\\_winning\\_label\\_id\(\)](#) (*labelbox.schema.data\_row.DataRow* method), 18  
[global\\_key](#) (*labelbox.schema.data\_row.DataRow* attribute), 17

**I**  
[iframe\\_url\\_path](#) (*labelbox.schema.labeling\_frontend.LabelingFrontend* attribute), 27  
[InconsistentOntologyException](#), 51  
[input\\_file\\_url](#) (*labelbox.schema.bulk\_import\_request.BulkImportRequest* attribute), 15  
[inputs](#) (*labelbox.schema.annotation\_import.AnnotationImport* property), 67  
[inputs](#) (*labelbox.schema.bulk\_import\_request.BulkImportRequest* property), 16  
[intercom\\_hash](#) (*labelbox.schema.user.User* attribute), 47  
[InternalServerError](#), 51  
[InvalidAttributeError](#), 51  
[InvalidQueryError](#), 51  
[invite\\_limit\(\)](#) (*labelbox.schema.organization.Organization* method), 36  
[invite\\_user\(\)](#) (*labelbox.schema.organization.Organization* method), 36  
[is\\_benchmark\\_reference](#) (*labelbox.schema.label.Label* attribute), 25  
[is\\_external\\_viewer](#) (*labelbox.schema.user.User* attribute), 47  
[is\\_viewer](#) (*labelbox.schema.user.User* attribute), 47

**L**  
[Label](#) (class in *labelbox.schema.label*), 25  
[label](#) (*labelbox.schema.label.Label* attribute), 25  
[label](#) (*labelbox.schema.review.Review* attribute), 43  
[label\\_generator\(\)](#) (*labelbox.schema.project.Project* method), 39  
[labelbox.client](#) module, 1  
[labelbox.exceptions](#) module, 51  
[labelbox.pagination](#) module, 53  
[labelbox.schema.annotation\\_import](#) module, 67  
[labelbox.schema.asset\\_attachment](#) module, 11  
[labelbox.schema.batch](#) module, 73  
[labelbox.schema.benchmark](#) module, 13  
[labelbox.schema.bulk\\_import\\_request](#) module, 15  
[labelbox.schema.data\\_row](#) module, 17  
[labelbox.schema.data\\_row\\_metadata](#)

module, 63  
 labelbox.schema.dataset  
   module, 21  
 labelbox.schema.enums  
   module, 55  
 labelbox.schema.label  
   module, 25  
 labelbox.schema.labeling\_frontend  
   module, 27  
 labelbox.schema.model  
   module, 61  
 labelbox.schema.model\_run  
   module, 57  
 labelbox.schema.ontology  
   module, 33  
 labelbox.schema.organization  
   module, 35  
 labelbox.schema.project  
   module, 37  
 labelbox.schema.resource\_tag  
   module, 75  
 labelbox.schema.review  
   module, 43  
 labelbox.schema.slice  
   module, 77, 79  
 labelbox.schema.task  
   module, 45  
 labelbox.schema.user  
   module, 47  
 labelbox.schema.webhook  
   module, 49  
 LabelboxError, 51  
 labeler\_performance() (labelbox.schema.project.Project method), 40  
 LabelImport (class in labelbox.schema.annotation\_import), 67  
 labeling\_frontend (labelbox.schema.labeling\_frontend.LabelingFrontendOptions attribute), 29  
 labeling\_frontend (labelbox.schema.project.Project attribute), 38  
 labeling\_frontend\_options (labelbox.schema.project.Project attribute), 38  
 labeling\_parameter\_overrides (labelbox.schema.project.Project attribute), 38  
 LabelingFrontend (class in labelbox.schema.labeling\_frontend), 27  
 labels (labelbox.schema.data\_row.DataRow attribute), 18  
 labels() (labelbox.schema.project.Project method), 40  
 last\_activity (labelbox.schema.benchmark.Benchmark attribute), 13

last\_activity\_time (labelbox.schema.project.Project attribute), 37

## M

MalformedQueryException, 52  
 MALPredictionImport (class in labelbox.schema.annotation\_import), 68  
 MALValidationError, 51  
 MEAPredictionImport (class in labelbox.schema.annotation\_import), 69  
 MEAToMALPredictionImport (class in labelbox.schema.annotation\_import), 70  
 media\_attributes (labelbox.schema.data\_row.DataRow attribute), 17  
 members() (labelbox.schema.project.Project method), 40  
 metadata (labelbox.schema.data\_row.DataRow attribute), 17  
 metadata\_fields (labelbox.schema.data\_row.DataRow attribute), 17  
 Model (class in labelbox.schema.model), 61  
 model\_runs (labelbox.schema.model.Model attribute), 61  
 ModelRun (class in labelbox.schema.model\_run), 57  
 ModelRun.Status (class in labelbox.schema.model\_run), 57  
 ModelRunDataRow (class in labelbox.schema.model\_run), 59  
 module  
   labelbox.client, 1  
   labelbox.exceptions, 51  
   labelbox.pagination, 53  
   labelbox.schema.annotation\_import, 67  
   labelbox.schema.asset\_attachment, 11  
   labelbox.schema.batch, 73  
   labelbox.schema.benchmark, 13  
   labelbox.schema.bulk\_import\_request, 15  
   labelbox.schema.data\_row, 17  
   labelbox.schema.data\_row\_metadata, 63  
   labelbox.schema.dataset, 21  
   labelbox.schema.enums, 55  
   labelbox.schema.label, 25  
   labelbox.schema.labeling\_frontend, 27  
   labelbox.schema.model, 61  
   labelbox.schema.model\_run, 57  
   labelbox.schema.ontology, 33  
   labelbox.schema.organization, 35  
   labelbox.schema.project, 37  
   labelbox.schema.resource\_tag, 75  
   labelbox.schema.review, 43  
   labelbox.schema.slice, 77, 79  
   labelbox.schema.task, 45

`labelbox.schema.user`, 47  
`labelbox.schema.webhook`, 49

## N

`name` (`labelbox.schema.batch.Batch` attribute), 73  
`name` (`labelbox.schema.bulk_import_request.BulkImportRequest` attribute), 15  
`name` (`labelbox.schema.dataset.Dataset` attribute), 21  
`name` (`labelbox.schema.labeling_frontend.LabelingFrontend` attribute), 27  
`name` (`labelbox.schema.organization.Organization` attribute), 35  
`name` (`labelbox.schema.project.Project` attribute), 37  
`name` (`labelbox.schema.slice.Slice` attribute), 77  
`name` (`labelbox.schema.task.Task` attribute), 45  
`name` (`labelbox.schema.user.User` attribute), 47  
`NetworkError`, 52  
`nickname` (`labelbox.schema.user.User` attribute), 47  
`normalized` (`labelbox.schema.ontology.Ontology` attribute), 33  
`number_of_labels` (`labelbox.schema.project.LabelingParameterOverride` attribute), 31

## O

`object_schema_count` (`labelbox.schema.ontology.Ontology` attribute), 33  
`Ontology` (class in `labelbox.schema.ontology`), 33  
`ontology` (`labelbox.schema.project.Project` attribute), 38  
`OntologyBuilder` (class in `labelbox.schema.ontology`), 34  
`OperationNotAllowedException`, 52  
`Organization` (class in `labelbox.schema.organization`), 35  
`organization` (`labelbox.schema.data_row.DataRow` attribute), 18  
`organization` (`labelbox.schema.dataset.Dataset` attribute), 21  
`organization` (`labelbox.schema.labeling_frontend.LabelingFrontendOptions` attribute), 29  
`organization` (`labelbox.schema.project.Project` attribute), 38  
`organization` (`labelbox.schema.review.Review` attribute), 43  
`organization` (`labelbox.schema.task.Task` attribute), 45  
`organization` (`labelbox.schema.user.User` attribute), 47

## P

`PaginatedCollection` (class in `labelbox.pagination`), 53  
`parent_id` (`labelbox.schema.annotation_import.LabelImport` property), 68

`parent_id` (`labelbox.schema.annotation_import.MALPredictionImport` property), 69  
`parent_id` (`labelbox.schema.annotation_import.MEAPredictionImport` property), 70  
`parent_id` (`labelbox.schema.annotation_import.MEAToMALPredictionImport` property), 71  
`parse_classification()` (in module `labelbox.schema.bulk_import_request`), 16  
`parse_metadata()` (`labelbox.schema.data_row_metadata.DataRowMetadataOntology` method), 65  
`parse_metadata_fields()` (`labelbox.schema.data_row_metadata.DataRowMetadataOntology` method), 65  
`parse_upsert_metadata()` (`labelbox.schema.data_row_metadata.DataRowMetadataOntology` method), 65  
`picture` (`labelbox.schema.user.User` attribute), 47  
`priority` (`labelbox.schema.project.LabelingParameterOverride` attribute), 31  
`ProcessingWaitTimeout`, 52  
`Project` (class in `labelbox.schema.project`), 37  
`project` (`labelbox.schema.batch.Batch` attribute), 73  
`project` (`labelbox.schema.bulk_import_request.BulkImportRequest` attribute), 15  
`project` (`labelbox.schema.label.Label` attribute), 25  
`project` (`labelbox.schema.labeling_frontend.LabelingFrontendOptions` attribute), 29  
`project` (`labelbox.schema.review.Review` attribute), 43  
`project()` (`labelbox.schema.batch.Batch` method), 74  
`project_resource_tag` (`labelbox.schema.resource_tag.ResourceTag` attribute), 75  
`ProjectMember` (class in `labelbox.schema.project`), 42  
`projects` (`labelbox.schema.dataset.Dataset` attribute), 21  
`projects` (`labelbox.schema.labeling_frontend.LabelingFrontend` attribute), 27  
`projects` (`labelbox.schema.ontology.Ontology` attribute), 33  
`projects` (`labelbox.schema.organization.Organization` attribute), 35  
`projects` (`labelbox.schema.user.User` attribute), 47

## Q

`queue_mode` (`labelbox.schema.project.Project` attribute), 37

## R

`reference_label` (`labelbox.schema.benchmark.Benchmark` attribute), 13  
`refresh()` (`labelbox.schema.annotation_import.AnnotationImport` method), 67



[refresh\(\)](#) (*labelbox.schema.bulk\_import\_request.BulkImportRequest* method), 16  
[refresh\(\)](#) (*labelbox.schema.task.Task* method), 45  
[refresh\\_ontology\(\)](#) (*labelbox.schema.data\_row\_metadata.DataRowMetadataOntology* method), 65  
[remove\\_from\\_project\(\)](#) (*labelbox.schema.user.User* method), 47  
[remove\\_queued\\_data\\_rows\(\)](#) (*labelbox.schema.batch.Batch* method), 74  
[remove\\_user\(\)](#) (*labelbox.schema.organization.Organization* method), 36  
[reset\\_config\(\)](#) (*labelbox.schema.model\_run.ModelRun* method), 57  
[ResourceConflict](#), 52  
[ResourceCreationError](#), 52  
[ResourceNotFoundError](#), 52  
[ResourceTag](#) (class in *labelbox.schema.resource\_tag*), 75  
[result](#) (*labelbox.schema.task.Task* property), 45  
[Review](#) (class in *labelbox.schema.review*), 43  
[Review.NetScore](#) (class in *labelbox.schema.review*), 43  
[review\\_metrics\(\)](#) (*labelbox.schema.project.Project* method), 40  
[reviews](#) (*labelbox.schema.label.Label* attribute), 25  
[row\\_count](#) (*labelbox.schema.dataset.Dataset* attribute), 21  
[row\\_data](#) (*labelbox.schema.data\_row.DataRow* attribute), 17

## S

[SchemaId](#) (in module *labelbox.schema.ontology*), 34  
[score](#) (*labelbox.schema.review.Review* attribute), 43  
[seconds\\_to\\_label](#) (*labelbox.schema.label.Label* attribute), 25  
[set\\_labeling\\_parameter\\_overrides\(\)](#) (*labelbox.schema.project.Project* method), 40  
[setup\(\)](#) (*labelbox.schema.project.Project* method), 41  
[setup\\_complete](#) (*labelbox.schema.project.Project* attribute), 37  
[setup\\_editor\(\)](#) (*labelbox.schema.project.Project* method), 41  
[Slice](#) (class in *labelbox.schema.slice*), 77  
[state](#) (*labelbox.schema.bulk\_import\_request.BulkImportRequest* attribute), 15  
[status](#) (*labelbox.schema.task.Task* attribute), 45  
[status](#) (*labelbox.schema.webhook.Webhook* attribute), 49  
[status\\_file\\_url](#) (*labelbox.schema.bulk\_import\_request.BulkImportRequest* attribute), 15

## T

[Task](#) (class in *labelbox.schema.task*), 45  
[text](#) (*labelbox.schema.resource\_tag.ResourceTag* attribute), 75  
[TimeoutError](#), 52  
[tools](#) (*labelbox.schema.ontology.OntologyBuilder* attribute), 34  
[tools\(\)](#) (*labelbox.schema.ontology.Ontology* method), 34  
[topics](#) (*labelbox.schema.webhook.Webhook* attribute), 49

## U

[unset\\_labeling\\_parameter\\_overrides\(\)](#) (*labelbox.schema.project.Project* method), 41  
[update\(\)](#) (*labelbox.schema.data\_row.DataRow* method), 18  
[update\(\)](#) (*labelbox.schema.project.Project* method), 41  
[update\(\)](#) (*labelbox.schema.webhook.Webhook* method), 50  
[update\\_config\(\)](#) (*labelbox.schema.model\_run.ModelRun* method), 58  
[update\\_enum\\_option\(\)](#) (*labelbox.schema.data\_row\_metadata.DataRowMetadataOntology* method), 65  
[update\\_org\\_role\(\)](#) (*labelbox.schema.user.User* method), 48  
[update\\_project\\_resource\\_tags\(\)](#) (*labelbox.schema.project.Project* method), 41  
[update\\_schema\(\)](#) (*labelbox.schema.data\_row\_metadata.DataRowMetadataOntology* method), 65  
[updated\\_at](#) (*labelbox.schema.batch.Batch* attribute), 73  
[updated\\_at](#) (*labelbox.schema.data\_row.DataRow* attribute), 17  
[updated\\_at](#) (*labelbox.schema.dataset.Dataset* attribute), 21  
[updated\\_at](#) (*labelbox.schema.ontology.Ontology* attribute), 33  
[updated\\_at](#) (*labelbox.schema.organization.Organization* attribute), 35  
[updated\\_at](#) (*labelbox.schema.project.Project* attribute), 37  
[updated\\_at](#) (*labelbox.schema.review.Review* attribute), 43  
[updated\\_at](#) (*labelbox.schema.slice.Slice* attribute), 77

`updated_at` (*labelbox.schema.task.Task* attribute), 45  
`updated_at` (*labelbox.schema.user.User* attribute), 47  
`updated_at` (*labelbox.schema.webhook.Webhook* attribute), 49  
`upload_annotations()` (*labelbox.schema.project.Project* method), 42  
`upsert_data_rows()` (*labelbox.schema.model\_run.ModelRun* method), 58  
`upsert_instructions()` (*labelbox.schema.project.Project* method), 42  
`upsert_labels()` (*labelbox.schema.model\_run.ModelRun* method), 58  
`upsert_predictions_and_send_to_project()` (*labelbox.schema.model\_run.ModelRun* method), 58  
`upsert_project_role()` (*labelbox.schema.user.User* method), 48  
`upsert_review_queue()` (*labelbox.schema.project.Project* method), 42  
`url` (*labelbox.schema.webhook.Webhook* attribute), 49  
`User` (class in *labelbox.schema.user*), 47  
`users` (*labelbox.schema.organization.Organization* attribute), 35  
`UuidError`, 52

## V

`ValidationFailedError`, 52

## W

`wait_till_done()` (*labelbox.schema.task.Task* method), 45  
`wait_until_done()` (*labelbox.schema.annotation\_import.AnnotationImport* method), 67  
`wait_until_done()` (*labelbox.schema.bulk\_import\_request.BulkImportRequest* method), 16  
`Webhook` (class in *labelbox.schema.webhook*), 49  
`Webhook.Status` (class in *labelbox.schema.webhook*), 49  
`Webhook.Topic` (class in *labelbox.schema.webhook*), 49  
`webhooks` (*labelbox.schema.organization.Organization* attribute), 35  
`webhooks` (*labelbox.schema.project.Project* attribute), 38