

---

# Labelbox Python API reference

*Release 3.11.0*

**Labelbox**

**Dec 18, 2021**



**CONTENTS:**

<b>1</b>	<b>Client</b>	<b>1</b>
<b>2</b>	<b>AssetAttachment</b>	<b>7</b>
<b>3</b>	<b>Benchmark</b>	<b>9</b>
<b>4</b>	<b>BulkImportRequest</b>	<b>11</b>
<b>5</b>	<b>DataRow</b>	<b>13</b>
<b>6</b>	<b>Dataset</b>	<b>15</b>
<b>7</b>	<b>Label</b>	<b>19</b>
<b>8</b>	<b>LabelingFrontend</b>	<b>21</b>
<b>9</b>	<b>LabelingFrontendOptions</b>	<b>23</b>
<b>10</b>	<b>LabelingParameterOverride</b>	<b>25</b>
<b>11</b>	<b>Ontology</b>	<b>27</b>
<b>12</b>	<b>Organization</b>	<b>29</b>
<b>13</b>	<b>Project</b>	<b>31</b>
<b>14</b>	<b>Review</b>	<b>37</b>
<b>15</b>	<b>Task</b>	<b>39</b>
<b>16</b>	<b>User</b>	<b>41</b>
<b>17</b>	<b>Webhook</b>	<b>43</b>
<b>18</b>	<b>Exceptions</b>	<b>45</b>
<b>19</b>	<b>Pagination</b>	<b>47</b>
<b>20</b>	<b>Enums</b>	<b>49</b>
<b>21</b>	<b>ModelRun</b>	<b>51</b>
<b>22</b>	<b>Model</b>	<b>53</b>

<b>23 DataRowMetadata</b>	<b>55</b>
<b>Python Module Index</b>	<b>57</b>
<b>Index</b>	<b>59</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.

```
create_dataset (iam_integration='DEFAULT', **kwargs)
```

Creates a Dataset object on the server.

Attribute values are passed as keyword arguments.

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

### Parameters

- **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*)

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*)

**Creates an ontology from normalized data**

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

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

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

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

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

**Parameters**

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

**Returns** The created Ontology

**create\_ontology\_from\_feature\_schemas** (*name, feature\_schema\_ids*)

Creates an ontology from a list of feature schema ids

**Parameters**

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

**Returns** The created Ontology

**create\_project** (*\*\*kwargs*)

Creates a Project object on the server.

Attribute values are passed as keyword arguments.

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

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

**Returns** A new Project object.

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

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

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

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

**Parameters**

- **query** (*str*) – The query to execute.
- **params** (*dict*) – Query parameters referenced within the query.

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

**Returns** dict, parsed JSON response.

**Raises**

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

**get\_data\_row** (*data\_row\_id*)

**Returns** returns a single data row given the data row id

**Return type** *DataRow*

**get\_data\_row\_ids\_for\_external\_ids** (*external\_ids: List[str]*) → Dict[str, List[str]]

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

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

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

**get\_data\_row\_metadata\_ontology** ()

**Returns** The ontology for Data Row Metadata for an organization

**Return type** *DataRowMetadataOntology*

**get\_dataset** (*dataset\_id*)

Gets a single Dataset with the given ID.

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

**Parameters** **dataset\_id** (*str*) – Unique ID of the Dataset.

**Returns** The sought Dataset.

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

**get\_datasets** (*where=None*)

Fetches one or more datasets.



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

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

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

**get\_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*)

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*)

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*)

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\_models** (*where=None*)

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*)

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*)

Fetches an Ontology by id.

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

**Returns** Ontology

**get\_organization** ()

Gets the Organization DB object of the current user.

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

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

Gets a single Project with the given ID.

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

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

**Returns** The sought Project.

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

**get\_projects** (*where=None*)

Fetches all the projects the user has access to.

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

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

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

**get\_roles** ()

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

**Return type** Roles

**get\_user** ()

Gets the current User database object.

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

## **ASSETATTACHMENT**

**class** `labelbox.schema.asset_attachment.AssetAttachment` (*client, field\_values*)

Bases: `labelbox.orm.db_object.DbObject`

Asset attachment provides extra context about an asset while labeling.

**attachment\_type**

IMAGE, VIDEO, TEXT, IMAGE\_OVERLAY, or HTML

**Type** `str`

**attachment\_value**

URL to an external file or a string of text

**Type** `str`

**class** `AttachmentType` (*value*)

Bases: `enum.Enum`

An enumeration.



## 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** () → None

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

**Returns** None

**property errors**

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

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

- This information will expire after 24 hours.

#### **property inputs**

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

**Returns** Uploaded ndjson.

- This information will expire after 24 hours.

**refresh()** → None

Synchronizes values of all fields with the database.

#### **property statuses**

Status for each individual annotation uploaded.

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

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

- This information will expire after 24 hours.

**wait\_until\_done** (*sleep\_time\_seconds: int = 5*) → None

Blocks import job until certain conditions are met.

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

**Parameters** *sleep\_time\_seconds* (*str*) – a time to block between subsequent API calls

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

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

**Parameters** *ontology* (*Ontology*) –

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

**Return type** Dict

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

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

**Parameters** *tool* (*dict*) –

**Returns** dict



## DATAROW

```
class labelbox.schema.data_row.DataRow (*args, **kwargs)
    Bases:    labelbox.orm.db_object.DbObject,    labelbox.orm.db_object.Updateable,
    labelbox.orm.db_object.BulkDeletable

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

    external_id
        User-generated file name or identifier
        Type str

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

    updated_at
        Type datetime

    created_at
        Type datetime

    media_attributes
        generated media attributes for the datarow
        Type dict

    dataset
        ToOne relationship to Dataset
        Type Relationship

    created_by
        ToOne relationship to User
        Type Relationship

    organization
        ToOne relationship to Organization
        Type Relationship

    labels
        ToMany relationship to Label
        Type Relationship

    attachments
```

#### Type Relationship

**static bulk\_delete** (*data\_rows*)

Deletes all the given DataRows.

**Parameters** *data\_rows* (*list of DataRow*) – The DataRows to delete.

**create\_attachment** (*attachment\_type, attachment\_value*)

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

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

#### Parameters

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

**Returns** *AssetAttachment* DB object.

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

## 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 (**kwargs)
        Creates a single DataRow belonging to this dataset.
```

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

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

**Raises**

- *InvalidQueryError* – If *DataRow.row\_data* field value is not provided in *kwargs*.
- *InvalidAttributeError* – in case the DB object type does not contain any of the field names given in *kwargs*.

**create\_data\_rows** (*items*)

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*)

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*)

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

**Parameters** **external\_id** (*str*) – External ID of the sought *DataRow*.

**Returns** A single *DataRow* with the given ID.

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

**data\_rows\_for\_external\_id** (*external\_id*, *limit=10*)

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

**Parameters**

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

**Returns** A single *DataRow* with the given ID.

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

**export\_data\_rows** (*timeout\_seconds=120*)

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

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

**Parameters** **timeout\_seconds** (*float*) – Max waiting time, in seconds.

**Returns** Generator that yields *DataRow* objects belonging to this dataset.

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



**LABEL**

```
class labelbox.schema.label.Label(*args, **kwargs)
    Bases:    labelbox.orm.db_object.DbObject,    labelbox.orm.db_object.Updateable,
    labelbox.orm.db_object.BulkDeletable

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

    label
        Type str

    seconds_to_label
        Type float

    agreement
        Type float

    benchmark_agreement
        Type float

    is_benchmark_reference
        Type bool

    project
        ToOne relationship to Project
        Type Relationship

    data_row
        ToOne relationship to DataRow
        Type Relationship

    reviews
        ToMany relationship to Review
        Type Relationship

    created_by
        ToOne relationship to User
        Type Relationship

    static bulk_delete (labels)
        Deletes all the given Labels.

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

**create\_benchmark()**

Creates a Benchmark for this Label.

**Returns** The newly created Benchmark.

**create\_review(\*\*kwargs)**

Creates a Review for this label.

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



## **LABELINGFRONTEND**

**class** `labelbox.schema.labeling_frontend.LabelingFrontend` (*client, field\_values*)

Bases: `labelbox.orm.db_object.DbObject`

Label editor.

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

**name**

**Type** `str`

**description**

**Type** `str`

**iframe\_url\_path**

**Type** `str`

**projects**

*ToMany* relationship to Project

**Type** `Relationship`



## **LABELINGFRONTENDOPTIONS**

```
class labelbox.schema.labeling_frontend.LabelingFrontendOptions (client,  
                                                                field_values)  
    Bases: labelbox.orm.db_object.DbObject  
    Label interface options.  
    customization_options  
        Type str  
    project  
        ToOne relationship to Project  
        Type Relationship  
    labeling_frontend  
        ToOne relationship to LabelingFrontend  
        Type Relationship  
    organization  
        ToOne relationship to Organization  
        Type Relationship
```



## LABELINGPARAMETEROVERRIDE

**class** `labelbox.schema.project.LabelingParameterOverride` (*client, field\_values*)

Bases: `labelbox.orm.db_object.DbObject`

Customizes the order of assets in the label queue.

**priority**

A prioritization score.

**Type** `int`

**number\_of\_labels**

Number of times an asset should be labeled.

**Type** `int`



## ONTOLOGY

```
class labelbox.schema.ontology.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.





## 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

**get\_default\_iam\_integration()**

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

**get\_iam\_integrations()**

Returns all IAM Integrations 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*: *labelbox.schema.role.Role*, *project\_roles*: *Optional[List[labelbox.schema.role.ProjectRole]]* = *None*) → *labelbox.schema.invite.Invite*  
 Invite a new member to the org. This will send the user an email invite

#### Parameters

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

**Returns** Invite for the user

#### Notes

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

- In this case *role.get\_roles* is likely not getting the right ids
2. **Multiple invites can be sent for the same email. This can only be resolved in the UI for now.**
    - Future releases of the SDK will support the ability to query and revoke invites to solve this problem (and/or checking on the backend)
  3. Some server side response are unclear (e.g. if the user invites themselves *None* is returned which the SDK raises as a *LabelboxError* )

**remove\_user** (*user*: *labelbox.schema.user.User*)

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

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

## PROJECT

```
class labelbox.schema.project.Project (client, field_values)
    Bases:    labelbox.orm.db_object.DbObject,  labelbox.orm.db_object.Updateable,
    labelbox.orm.db_object.Deletable

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

    name
        Type str

    description
        Type str

    updated_at
        Type datetime

    created_at
        Type datetime

    setup_complete
        Type datetime

    last_activity_time
        Type datetime

    auto_audit_number_of_labels
        Type int

    auto_audit_percentage
        Type float

    datasets
        ToMany relationship to Dataset
        Type Relationship

    created_by
        ToOne relationship to User
        Type Relationship

    organization
        ToOne relationship to Organization
        Type Relationship
```

**reviews**

*ToMany* relationship to Review

**Type** Relationship

**labeling\_frontend**

*ToOne* relationship to LabelingFrontend

**Type** Relationship

**labeling\_frontend\_options**

*ToMany* relationship to LabelingFrontendOptions

**Type** Relationship

**labeling\_parameter\_overrides**

*ToMany* relationship to LabelingParameterOverride

**Type** Relationship

**webhooks**

*ToMany* relationship to Webhook

**Type** Relationship

**benchmarks**

*ToMany* relationship to Benchmark

**Type** Relationship

**ontology**

*ToOne* relationship to Ontology

**Type** Relationship

**bulk\_import\_requests** ()

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

**dequeue** (*data\_row\_ids: List[str]*)

Remove Data Rows from the Project queue

**enable\_model\_assisted\_labeling** (*toggle: bool = True*) → bool

Turns model assisted labeling either on or off based on input

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

**Returns** True if toggled on or False if toggled off

**export\_issues** (*status=None*)

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

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

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

**export\_labels** (*download=False, timeout\_seconds=600*)

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

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

**Parameters** **timeout\_seconds** (*float*) – Max waiting time, in seconds.

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

**export\_queued\_data\_rows** (*timeout\_seconds=120*)

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

**Parameters** *timeout\_seconds* (*float*) – Max waiting time, in seconds.

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

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

**extend\_reservations** (*queue\_type*)

Extends all the current reservations for the current user on the given queue type. :param queue\_type: Either “LabelingQueue” or “ReviewQueue” :type queue\_type: str

**Returns** int, the number of reservations that were extended.

**label\_generator** (*timeout\_seconds=600*)

Download text and image annotations

**Returns** LabelGenerator for accessing labels for each text or image

**labeler\_performance** ()

Returns the labeler performances for this Project.

**Returns** A PaginatedCollection of LabelerPerformance objects.

**labels** (*datasets=None, order\_by=None*)

Custom relationship expansion method to support limited filtering.

**Parameters**

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

**members** ()

Fetch all current members for this project

**Returns** A `PaginatedCollection of `ProjectMember`s

**queue** (*data\_row\_ids: List[str]*)

Add Data Rows to the Project queue

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

Returns this Project’s review metrics.

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

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

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

Adds labeling parameter overrides to this project.

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

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

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

**Priority:**

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

**Number of labels:**

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

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

**setup** (*labeling\_frontend*, *labeling\_frontend\_options*)

Finalizes the Project setup.

**Parameters**

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

**setup\_editor** (*ontology*)

Sets up the project using the Pictor editor.

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

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

Removes labeling parameter overrides to this project.

- This will remove unlabeled duplicates in the queue.

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

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

**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.

**upload\_annotations** (*name: str, annotations: Union[str, pathlib.Path, Iterable[Dict]], validate: bool = True*) → *labelbox.schema.bulk\_import\_request.BulkImportRequest*  
Uploads annotations to a new Editor project.

**Parameters**

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

**Returns** BulkImportRequest

**upsert\_instructions** (*instructions\_file: str*)

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

**Parameters** *instructions\_file* (*str*) – Path to a local file. \* Must be a pdf file

**Raises** **ValueError** –

- project must be setup \* instructions file must have a “.pdf” extension

**upsert\_review\_queue** (*quota\_factor*)

Sets the the proportion of total assets in a project to review.

**More information can be found here:** <https://docs.labelbox.com/en/quality-assurance/review-labels#configure-review-percentage>

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

**video\_label\_generator** (*timeout\_seconds=600*)

Download video annotations

**Returns** LabelGenerator for accessing labels for each video

**class** *labelbox.schema.project.ProjectMember* (*client, field\_values*)

Bases: *labelbox.orm.db\_object.DbObject*

**class** *labelbox.schema.project.QueueErrors* (*value*)

Bases: *enum.Enum*

An enumeration.

**class** *labelbox.schema.project.QueueMode* (*value*)

Bases: *enum.Enum*

An enumeration.





## REVIEW

```
class labelbox.schema.review.Review(client, field_values)  
    Bases:    labelbox.orm.db_object.DbObject,    labelbox.orm.db_object.Deletable,  
    labelbox.orm.db_object.Updateable
```

Reviewing labeled data is a collaborative quality assurance technique.

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

**created\_at**  
    **Type** datetime

**updated\_at**  
    **Type** datetime

**score**  
    **Type** float

**created\_by**  
    *ToOne* relationship to User  
    **Type** Relationship

**organization**  
    *ToOne* relationship to Organization  
    **Type** Relationship

**project**  
    *ToOne* relationship to Project  
    **Type** Relationship

**label**  
    *ToOne* relationship to Label  
    **Type** Relationship

```
class NetScore(value)  
    Bases: enum.Enum  
    Negative, Zero, or Positive.
```



## TASK

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

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

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

```
    updated_at
```

```
        Type datetime
```

```
    created_at
```

```
        Type datetime
```

```
    name
```

```
        Type str
```

```
    status
```

```
        Type str
```

```
    completion_percentage
```

```
        Type float
```

```
    created_by
```

```
        ToOne relationship to User
```

```
        Type Relationship
```

```
    organization
```

```
        ToOne relationship to Organization
```

```
        Type Relationship
```

```
    refresh()
```

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

```
    wait_till_done(timeout_seconds=300)
```

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

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



## USER

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

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

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

```
    updated_at  
        Type datetime
```

```
    created_at  
        Type datetime
```

```
    email  
        Type str
```

```
    name  
        Type str
```

```
    nickname  
        Type str
```

```
    intercom_hash  
        Type str
```

```
    picture  
        Type str
```

```
    is_viewer  
        Type bool
```

```
    is_external_viewer  
        Type bool
```

```
    organization  
        ToOne relationship to Organization  
        Type Relationship
```

```
    created_tasks  
        ToMany relationship to Task  
        Type Relationship
```

```
    projects  
        ToMany relationship to Project  
        Type Relationship
```

```
    remove_from_project (project: labelbox.schema.project.Project)  
        Removes a User from a project. Only used for project based users. Project based user means their org  
        role is "NONE"
```

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

**update\_org\_role** (`role`: `labelbox.schema.role.Role`)

Updated the User's organization role.

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

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

**upsert\_project\_role** (`project`: `labelbox.schema.project.Project`, `role`: `labelbox.schema.role.Role`)

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

**Parameters**

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

## WEBHOOK

```
class labelbox.schema.webhook.Webhook (client, field_values)
    Bases: labelbox.orm.db_object.DbObject, labelbox.orm.db_object.Updateable

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

    updated_at
        Type datetime

    created_at
        Type datetime

    url
        Type str

    topics
        LABEL_CREATED, LABEL_UPDATED, LABEL_DELETED REVIEW_CREATED, RE-
        VIEW_UPDATED, REVIEW_DELETED
        Type str

    status
        ACTIVE, INACTIVE, REVOKED
        Type str

class Status (value)
    Bases: enum.Enum

    An enumeration.

class Topic (value)
    Bases: enum.Enum

    An enumeration.

static create (client, topics, url, secret, project)
    Creates a Webhook.
    Parameters
        • client (Client) – The Labelbox client used to connect to the server.
        • topics (list of str) – A list of topics this Webhook should get notifica-
          tions for. Must be one of Webhook.Topic
        • url (str) – The URL to which notifications should be sent by the Labelbox
          server.
        • secret (str) – A secret key used for signing notifications.
```

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

**Returns** A newly created Webhook.

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

**Information on configuring your server can be found here (this is where the url points to and the secret is set).**

<https://docs.labelbox.com/en/configure-editor/webhooks-setup#setup-steps>

**delete()**

Deletes the webhook

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

Updates the Webhook.

**Parameters**

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



## EXCEPTIONS

**exception** `labelbox.exceptions.ApiLimitError` (*message, cause=None*)  
Bases: `labelbox.exceptions.LabelboxError`

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

**exception** `labelbox.exceptions.AuthenticationError` (*message, cause=None*)  
Bases: `labelbox.exceptions.LabelboxError`

Raised when an API key fails authentication.

**exception** `labelbox.exceptions.AuthorizationError` (*message, cause=None*)  
Bases: `labelbox.exceptions.LabelboxError`

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

**exception** `labelbox.exceptions.InconsistentOntologyException`  
Bases: `Exception`

**exception** `labelbox.exceptions.InternalServerError` (*message, cause=None*)  
Bases: `labelbox.exceptions.LabelboxError`

Nondescript prisma or 502 related errors.

Meant to be retryable.

TODO: these errors need better messages from platform

**exception** `labelbox.exceptions.InvalidAttributeError` (*db\_object\_type, field*)  
Bases: `labelbox.exceptions.LabelboxError`

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

**exception** `labelbox.exceptions.InvalidQueryError` (*message, cause=None*)  
Bases: `labelbox.exceptions.LabelboxError`

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

**exception** `labelbox.exceptions.LabelboxError` (*message, cause=None*)  
Bases: `Exception`

Base class for exceptions.

**exception** `labelbox.exceptions.MALValidationError` (*message, cause=None*)  
Bases: `labelbox.exceptions.LabelboxError`

Raised when user input is invalid for MAL imports.

**exception** `labelbox.exceptions.MalformedQueryException`  
Bases: `Exception`

Raised when the user submits a malformed query.

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

Bases: `labelbox.exceptions.LabelboxError`

Raised when an HTTPError occurs.

**exception** `labelbox.exceptions.OperationNotAllowedException`

Bases: `Exception`

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

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

Bases: `labelbox.exceptions.LabelboxError`

Exception raised when a given resource conflicts with another.

**exception** `labelbox.exceptions.ResourceNotFoundError` (*db\_object\_type, params*)

Bases: `labelbox.exceptions.LabelboxError`

Exception raised when a given resource is not found.

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

Bases: `labelbox.exceptions.LabelboxError`

Raised when a request times-out.

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

Bases: `labelbox.exceptions.LabelboxError`

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

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

Bases: `labelbox.exceptions.LabelboxError`

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

## PAGINATION

```
class labelbox.pagination.PaginatedCollection(client: Client, query: str, params:  
Dict[str, str], dereferencing: Dict[str,  
Any], obj_class: Type[DBObject], cur-  
sor_path: Optional[Dict[str, Any]] =  
None, experimental: bool = False)
```

Bases: object

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

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

```
__init__(client: Client, query: str, params: Dict[str, str], dereferencing: Dict[str, Any], obj_class:  
Type[DBObject], cursor_path: Optional[Dict[str, Any]] = None, experimental: bool =  
False)
```

Creates a PaginatedCollection.

### Parameters

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



## ENUMS

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

Bases: `enum.Enum`

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

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



## MODEL RUN

```
class labelbox.schema.model_run.ModelRun(client, field_values)
    Bases: labelbox.orm.db_object.DbObject

    add_predictions(name: str, predictions: Union[str, pathlib.Path, Iterable[Dict]]) → label-
        box.schema.annotation_import.MEAPredictionImport
        Uploads predictions to a new Editor project. :param name: name of the AnnotationImport job :type name:
        str :param predictions: url that is publicly accessible by Labelbox containing an
            ndjson file OR local path to an ndjson file OR iterable of annotation rows
            Returns AnnotationImport

    delete()
        Deletes specified model run.
            Returns Query execution success.

    delete_model_run_data_rows(data_row_ids)
        Deletes data rows from model runs.
            Parameters data_row_ids (list) – List of data row ids to delete from the model run.
            Returns Query execution success.

    upsert_data_rows(data_row_ids, timeout_seconds=60)
        Adds data rows to a model run without any associated labels :param data_row_ids: data row ids to add
        to mea :type data_row_ids: list :param timeout_seconds: Max waiting time, in seconds. :type time-
        out_seconds: float
            Returns ID of newly generated async task

    upsert_labels(label_ids, timeout_seconds=60)
        Adds data rows and labels to a model run :param label_ids: label ids to insert :type label_ids: list :param
        timeout_seconds: Max waiting time, in seconds. :type timeout_seconds: float
            Returns ID of newly generated async task

class labelbox.schema.model_run.ModelRunDataRow(client, model_id, *args, **kwargs)
    Bases: labelbox.orm.db_object.DbObject
```





## 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*)

Creates a model run belonging to this model.

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

**Returns** `ModelRun`, the created model run.

**delete** ()

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.

**parse\_metadata** (*unparsed*: *List[Dict[str, List[Union[str, Dict]]]]*) →  
*List[labelbox.schema.data\_row\_metadata.DataRowMetadata]*  
Parse metadata responses

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

**Parameters** **unparsed** – An unparsed metadata export

**Returns** List of *DataRowMetadata*

**Return type** metadata

labelbox.schema.data\_row\_metadata.**String**  
alias of labelbox.schema.data\_row\_metadata.ConstrainedStrValue

## PYTHON MODULE INDEX

### I

- `labelbox.client`, 1
- `labelbox.exceptions`, 45
- `labelbox.pagination`, 47
- `labelbox.schema.asset_attachment`, 7
- `labelbox.schema.benchmark`, 9
- `labelbox.schema.bulk_import_request`, 11
- `labelbox.schema.data_row`, 13
- `labelbox.schema.data_row_metadata`, 55
- `labelbox.schema.dataset`, 15
- `labelbox.schema.enums`, 49
- `labelbox.schema.label`, 19
- `labelbox.schema.labeling_frontend`, 21
- `labelbox.schema.model`, 53
- `labelbox.schema.model_run`, 51
- `labelbox.schema.ontology`, 27
- `labelbox.schema.organization`, 29
- `labelbox.schema.project`, 31
- `labelbox.schema.review`, 37
- `labelbox.schema.task`, 39
- `labelbox.schema.user`, 41
- `labelbox.schema.webhook`, 43



## Symbols

`__init__()` (*labelbox.client.Client* method), 1  
`__init__()` (*labelbox.pagination.PaginatedCollection* method), 47

## A

`add_predictions()` (*labelbox.schema.model\_run.ModelRun* method), 51  
`agreement` (*labelbox.schema.label.Label* attribute), 19  
`AnnotationImportState` (class in *labelbox.schema.enums*), 49  
`ApiLimitError`, 45  
`AssetAttachment` (class in *labelbox.schema.asset\_attachment*), 7  
`AssetAttachment.AttachmentType` (class in *labelbox.schema.asset\_attachment*), 7  
`attachment_type` (*labelbox.schema.asset\_attachment.AssetAttachment* attribute), 7  
`attachment_value` (*labelbox.schema.asset\_attachment.AssetAttachment* attribute), 7  
`attachments` (*labelbox.schema.data\_row.DataRow* attribute), 13  
`AuthenticationError`, 45  
`AuthorizationError`, 45  
`auto_audit_number_of_labels` (*labelbox.schema.project.Project* attribute), 31  
`auto_audit_percentage` (*labelbox.schema.project.Project* attribute), 31  
`average_agreement` (*labelbox.schema.benchmark.Benchmark* attribute), 9

## B

`Benchmark` (class in *labelbox.schema.benchmark*), 9  
`benchmark_agreement` (*labelbox.schema.label.Label* attribute), 19  
`benchmarks` (*labelbox.schema.project.Project* attribute), 32

`bulk_delete()` (*labelbox.schema.data\_row.DataRow* static method), 14  
`bulk_delete()` (*labelbox.schema.data\_row\_metadata.DataRowMetadataOntology* method), 55  
`bulk_delete()` (*labelbox.schema.label.Label* static method), 19  
`bulk_export()` (*labelbox.schema.data\_row\_metadata.DataRowMetadataOntology* method), 55  
`bulk_import_requests()` (*labelbox.schema.project.Project* method), 32  
`bulk_upsert()` (*labelbox.schema.data\_row\_metadata.DataRowMetadataOntology* method), 55  
`BulkImportRequest` (class in *labelbox.schema.bulk\_import\_request*), 11  
`BulkImportRequestState` (class in *labelbox.schema.enums*), 49

## C

`classification_schema_count` (*labelbox.schema.ontology.Ontology* attribute), 27  
`classifications` (*labelbox.schema.ontology.OntologyBuilder* attribute), 28  
`classifications()` (*labelbox.schema.ontology.Ontology* method), 27  
`Client` (class in *labelbox.client*), 1  
`completed_count` (*labelbox.schema.benchmark.Benchmark* attribute), 9  
`completion_percentage` (*labelbox.schema.task.Task* attribute), 39  
`create()` (*labelbox.schema.webhook.Webhook* static method), 43  
`create_attachment()` (*labelbox.schema.data\_row.DataRow* method), 14

[create\\_benchmark\(\)](#) ([labelbox.schema.label.Label](#) method), 19  
[create\\_data\\_row\(\)](#) ([labelbox.schema.dataset.Dataset](#) method), 15  
[create\\_data\\_rows\(\)](#) ([labelbox.schema.dataset.Dataset](#) method), 16  
[create\\_data\\_rows\\_sync\(\)](#) ([labelbox.schema.dataset.Dataset](#) method), 16  
[create\\_dataset\(\)](#) ([labelbox.client.Client](#) method), 1  
[create\\_feature\\_schema\(\)](#) ([labelbox.client.Client](#) method), 2  
[create\\_model\(\)](#) ([labelbox.client.Client](#) method), 2  
[create\\_model\\_run\(\)](#) ([labelbox.schema.model.Model](#) method), 53  
[create\\_ontology\(\)](#) ([labelbox.client.Client](#) method), 2  
[create\\_ontology\\_from\\_feature\\_schemas\(\)](#) ([labelbox.client.Client](#) method), 3  
[create\\_project\(\)](#) ([labelbox.client.Client](#) method), 3  
[create\\_review\(\)](#) ([labelbox.schema.label.Label](#) method), 20  
[created\\_at](#) ([labelbox.schema.benchmark.Benchmark](#) attribute), 9  
[created\\_at](#) ([labelbox.schema.bulk\\_import\\_request.BulkImportRequest](#) attribute), 11  
[created\\_at](#) ([labelbox.schema.data\\_row.DataRow](#) attribute), 13  
[created\\_at](#) ([labelbox.schema.dataset.Dataset](#) attribute), 15  
[created\\_at](#) ([labelbox.schema.ontology.Ontology](#) attribute), 27  
[created\\_at](#) ([labelbox.schema.organization.Organization](#) attribute), 29  
[created\\_at](#) ([labelbox.schema.project.Project](#) attribute), 31  
[created\\_at](#) ([labelbox.schema.review.Review](#) attribute), 37  
[created\\_at](#) ([labelbox.schema.task.Task](#) attribute), 39  
[created\\_at](#) ([labelbox.schema.user.User](#) attribute), 41  
[created\\_at](#) ([labelbox.schema.webhook.Webhook](#) attribute), 43  
[created\\_by](#) ([labelbox.schema.benchmark.Benchmark](#) attribute), 9  
[created\\_by](#) ([labelbox.schema.bulk\\_import\\_request.BulkImportRequest](#) attribute), 11  
[created\\_by](#) ([labelbox.schema.data\\_row.DataRow](#) attribute), 13  
[created\\_by](#) ([labelbox.schema.dataset.Dataset](#) attribute), 15  
[created\\_by](#) ([labelbox.schema.label.Label](#) attribute), 19  
[created\\_by](#) ([labelbox.schema.ontology.Ontology](#) attribute), 27  
[created\\_by](#) ([labelbox.schema.project.Project](#) attribute), 31  
[created\\_by](#) ([labelbox.schema.review.Review](#) attribute), 37  
[created\\_by](#) ([labelbox.schema.task.Task](#) attribute), 39  
[created\\_tasks](#) ([labelbox.schema.user.User](#) attribute), 41  
[customization\\_options](#) ([labelbox.schema.labeling\\_frontend.LabelingFrontendOptions](#) attribute), 23

## D

[data\\_row](#) ([labelbox.schema.label.Label](#) attribute), 19  
[data\\_row\\_for\\_external\\_id\(\)](#) ([labelbox.schema.dataset.Dataset](#) method), 16  
[data\\_rows](#) ([labelbox.schema.dataset.Dataset](#) attribute), 15  
[data\\_rows\\_for\\_external\\_id\(\)](#) ([labelbox.schema.dataset.Dataset](#) method), 17  
[DataRow](#) (class in [labelbox.schema.data\\_row](#)), 13  
[DataRowMetadataKind](#) (class in [labelbox.schema.data\\_row\\_metadata](#)), 55  
[DataRowMetadataOntology](#) (class in [labelbox.schema.data\\_row\\_metadata](#)), 55  
[dataset](#) ([labelbox.schema.data\\_row.DataRow](#) attribute), 13  
[datasets](#) ([labelbox.schema.project.Project](#) attribute), 31  
[delete\(\)](#) ([labelbox.schema.bulk\\_import\\_request.BulkImportRequest](#) method), 11  
[delete\(\)](#) ([labelbox.schema.model.Model](#) method), 53  
[delete\(\)](#) ([labelbox.schema.model\\_run.ModelRun](#) method), 51  
[delete\(\)](#) ([labelbox.schema.webhook.Webhook](#) method), 44  
[delete\\_model\\_run\\_data\\_rows\(\)](#) ([labelbox.schema.model\\_run.ModelRun](#) method), 51  
[dequeue\(\)](#) ([labelbox.schema.project.Project](#) method), 32  
[description](#) ([labelbox.schema.dataset.Dataset](#) attribute), 15  
[description](#) ([labelbox.schema.labeling\\_frontend.LabelingFrontendOptions](#) attribute), 21  
[description](#) ([labelbox.schema.ontology.Ontology](#) attribute), 27  
[description](#) ([labelbox.schema.project.Project](#) attribute), 31

## E

[email](#) ([labelbox.schema.user.User](#) attribute), 41



[enable\\_model\\_assisted\\_labeling\(\)](#) (*labelbox.schema.project.Project method*), 32  
[error\\_file\\_url](#) (*labelbox.schema.bulk\_import\_request.BulkImportRequest attribute*), 11  
[errors\(\)](#) (*labelbox.schema.bulk\_import\_request.BulkImportRequest property*), 11  
[execute\(\)](#) (*labelbox.client.Client method*), 3  
[export\\_data\\_rows\(\)](#) (*labelbox.schema.dataset.Dataset method*), 17  
[export\\_issues\(\)](#) (*labelbox.schema.project.Project method*), 32  
[export\\_labels\(\)](#) (*labelbox.schema.project.Project method*), 32  
[export\\_queued\\_data\\_rows\(\)](#) (*labelbox.schema.project.Project method*), 32  
[extend\\_reservations\(\)](#) (*labelbox.schema.project.Project method*), 33  
[external\\_id](#) (*labelbox.schema.data\_row.DataRow attribute*), 13

## F

[FeatureSchema](#) (*class in labelbox.schema.ontology*), 27  
[FeatureSchemaId](#) (*in module labelbox.schema.ontology*), 27

## G

[get\\_data\\_row\(\)](#) (*labelbox.client.Client method*), 4  
[get\\_data\\_row\\_ids\\_for\\_external\\_ids\(\)](#) (*labelbox.client.Client method*), 4  
[get\\_data\\_row\\_metadata\\_ontology\(\)](#) (*labelbox.client.Client method*), 4  
[get\\_dataset\(\)](#) (*labelbox.client.Client method*), 4  
[get\\_datasets\(\)](#) (*labelbox.client.Client method*), 4  
[get\\_default\\_iam\\_integration\(\)](#) (*labelbox.schema.organization.Organization method*), 29  
[get\\_feature\\_schema\(\)](#) (*labelbox.client.Client method*), 5  
[get\\_feature\\_schemas\(\)](#) (*labelbox.client.Client method*), 5  
[get\\_iam\\_integrations\(\)](#) (*labelbox.schema.organization.Organization method*), 29  
[get\\_labeling\\_frontends\(\)](#) (*labelbox.client.Client method*), 5  
[get\\_mal\\_schemas\(\)](#) (*in module labelbox.schema.bulk\_import\_request*), 12  
[get\\_model\(\)](#) (*labelbox.client.Client method*), 5  
[get\\_models\(\)](#) (*labelbox.client.Client method*), 5  
[get\\_ontologies\(\)](#) (*labelbox.client.Client method*), 5  
[get\\_ontology\(\)](#) (*labelbox.client.Client method*), 6  
[get\\_organization\(\)](#) (*labelbox.client.Client method*), 6  
[get\\_project\(\)](#) (*labelbox.client.Client method*), 6  
[get\\_projects\(\)](#) (*labelbox.client.Client method*), 6  
[get\\_roles\(\)](#) (*labelbox.client.Client method*), 6  
[get\\_user\(\)](#) (*labelbox.client.Client method*), 6

## I

[iframe\\_url\\_path](#) (*labelbox.schema.labeling\_frontend.LabelingFrontend attribute*), 21  
[InconsistentOntologyException](#), 45  
[input\\_file\\_url](#) (*labelbox.schema.bulk\_import\_request.BulkImportRequest attribute*), 11  
[inputs\(\)](#) (*labelbox.schema.bulk\_import\_request.BulkImportRequest property*), 12  
[intercom\\_hash](#) (*labelbox.schema.user.User attribute*), 41  
[InternalServerError](#), 45  
[InvalidAttributeError](#), 45  
[InvalidQueryError](#), 45  
[invite\\_limit\(\)](#) (*labelbox.schema.organization.Organization method*), 29  
[invite\\_user\(\)](#) (*labelbox.schema.organization.Organization method*), 29  
[is\\_benchmark\\_reference](#) (*labelbox.schema.label.Label attribute*), 19  
[is\\_external\\_viewer](#) (*labelbox.schema.user.User attribute*), 41  
[is\\_viewer](#) (*labelbox.schema.user.User attribute*), 41

## L

[Label](#) (*class in labelbox.schema.label*), 19  
[label](#) (*labelbox.schema.label.Label attribute*), 19  
[label](#) (*labelbox.schema.review.Review attribute*), 37  
[label\\_generator\(\)](#) (*labelbox.schema.project.Project method*), 33  
[labelbox.client](#) module, 1  
[labelbox.exceptions](#) module, 45  
[labelbox.pagination](#) module, 47  
[labelbox.schema.asset\\_attachment](#) module, 7  
[labelbox.schema.benchmark](#) module, 9  
[labelbox.schema.bulk\\_import\\_request](#) module, 11  
[labelbox.schema.data\\_row](#) module, 13

[labelbox.schema.data\\_row\\_metadata](#)  
     module, 55  
[labelbox.schema.dataset](#)  
     module, 15  
[labelbox.schema.enums](#)  
     module, 49  
[labelbox.schema.label](#)  
     module, 19  
[labelbox.schema.labeling\\_frontend](#)  
     module, 21  
[labelbox.schema.model](#)  
     module, 53  
[labelbox.schema.model\\_run](#)  
     module, 51  
[labelbox.schema.ontology](#)  
     module, 27  
[labelbox.schema.organization](#)  
     module, 29  
[labelbox.schema.project](#)  
     module, 31  
[labelbox.schema.review](#)  
     module, 37  
[labelbox.schema.task](#)  
     module, 39  
[labelbox.schema.user](#)  
     module, 41  
[labelbox.schema.webhook](#)  
     module, 43  
[LabelboxError](#), 45  
[labeler\\_performance\(\)](#) (*labelbox.schema.project.Project* method), 33  
[labeling\\_frontend](#) (*labelbox.schema.labeling\_frontend.LabelingFrontendOptions* attribute), 23  
[labeling\\_frontend](#) (*labelbox.schema.project.Project* attribute), 32  
[labeling\\_frontend\\_options](#) (*labelbox.schema.project.Project* attribute), 32  
[labeling\\_parameter\\_overrides](#) (*labelbox.schema.project.Project* attribute), 32  
[LabelingFrontend](#) (class in *labelbox.schema.labeling\_frontend*), 21  
[labels](#) (*labelbox.schema.data\_row.DataRow* attribute), 13  
[labels\(\)](#) (*labelbox.schema.project.Project* method), 33  
[last\\_activity](#) (*labelbox.schema.benchmark.Benchmark* attribute), 9  
[last\\_activity\\_time](#) (*labelbox.schema.project.Project* attribute), 31

## M

[MalformedQueryException](#), 45

[MALValidationError](#), 45  
[media\\_attributes](#) (*labelbox.schema.data\_row.DataRow* attribute), 13  
[members\(\)](#) (*labelbox.schema.project.Project* method), 33  
[Model](#) (class in *labelbox.schema.model*), 53  
[model\\_runs](#) (*labelbox.schema.model.Model* attribute), 53  
[ModelRun](#) (class in *labelbox.schema.model\_run*), 51  
[ModelRunDataRow](#) (class in *labelbox.schema.model\_run*), 51  
[module](#)  
     [labelbox.client](#), 1  
     [labelbox.exceptions](#), 45  
     [labelbox.pagination](#), 47  
     [labelbox.schema.asset\\_attachment](#), 7  
     [labelbox.schema.benchmark](#), 9  
     [labelbox.schema.bulk\\_import\\_request](#), 11  
     [labelbox.schema.data\\_row](#), 13  
     [labelbox.schema.data\\_row\\_metadata](#), 55  
     [labelbox.schema.dataset](#), 15  
     [labelbox.schema.enums](#), 49  
     [labelbox.schema.label](#), 19  
     [labelbox.schema.labeling\\_frontend](#), 21  
     [labelbox.schema.model](#), 53  
     [labelbox.schema.model\\_run](#), 51  
     [labelbox.schema.ontology](#), 27  
     [labelbox.schema.organization](#), 29  
     [labelbox.schema.project](#), 31  
     [labelbox.schema.review](#), 37  
     [labelbox.schema.task](#), 39  
     [labelbox.schema.user](#), 41  
     [labelbox.schema.webhook](#), 43

## N

[name](#) (*labelbox.schema.bulk\_import\_request.BulkImportRequest* attribute), 11  
[name](#) (*labelbox.schema.dataset.Dataset* attribute), 15  
[name](#) (*labelbox.schema.labeling\_frontend.LabelingFrontend* attribute), 21  
[name](#) (*labelbox.schema.organization.Organization* attribute), 29  
[name](#) (*labelbox.schema.project.Project* attribute), 31  
[name](#) (*labelbox.schema.task.Task* attribute), 39  
[name](#) (*labelbox.schema.user.User* attribute), 41  
[NetworkError](#), 46  
[nickname](#) (*labelbox.schema.user.User* attribute), 41  
[normalized](#) (*labelbox.schema.ontology.Ontology* attribute), 27

number_of_labels	(labelbox.schema.project.LabelingParameterOverride attribute), 25	projects (labelbox.schema.labeling_frontend.LabelingFrontend attribute), 21
O		projects (labelbox.schema.ontology.Ontology attribute), 27
object_schema_count	(labelbox.schema.ontology.Ontology attribute), 27	projects (labelbox.schema.organization.Organization attribute), 29
Ontology (class in labelbox.schema.ontology), 27		projects (labelbox.schema.user.User attribute), 41
ontology (labelbox.schema.project.Project attribute), 32		Q
OntologyBuilder (class in labelbox.schema.ontology), 28		queue() (labelbox.schema.project.Project method), 33
OperationNotAllowedException, 46		QueueErrors (class in labelbox.schema.project), 35
Organization (class in labelbox.schema.organization), 29		QueueMode (class in labelbox.schema.project), 35
organization (labelbox.schema.data_row.DataRow attribute), 13		R
organization (labelbox.schema.dataset.Dataset attribute), 15		reference_label (labelbox.schema.benchmark.Benchmark attribute), 9
organization (labelbox.schema.labeling_frontend.LabelingFrontend attribute), 23		refresh() (labelbox.schema.bulk_import_request.BulkImportRequest method), 12
organization (labelbox.schema.project.Project attribute), 31		refresh() (labelbox.schema.task.Task method), 39
organization (labelbox.schema.review.Review attribute), 37		remove_from_project() (labelbox.schema.user.User method), 41
organization (labelbox.schema.task.Task attribute), 39		remove_user() (labelbox.schema.organization.Organization method), 30
organization (labelbox.schema.user.User attribute), 41		ResourceConflict, 46
P		ResourceNotFoundError, 46
PaginatedCollection (class in labelbox.pagination), 47		Review (class in labelbox.schema.review), 37
parse_classification() (in module labelbox.schema.bulk_import_request), 12		Review.NetScore (class in labelbox.schema.review), 37
parse_metadata() (labelbox.schema.data_row_metadata.DataRowMetadata attribute), 56		review_metrics() (labelbox.schema.project.Project method), 33
picture (labelbox.schema.user.User attribute), 41		reviews (labelbox.schema.label.Label attribute), 19
priority (labelbox.schema.project.LabelingParameterOverride attribute), 25		reviews (labelbox.schema.project.Project attribute), 31
Project (class in labelbox.schema.project), 31		row_count (labelbox.schema.dataset.Dataset attribute), 15
project (labelbox.schema.bulk_import_request.BulkImportRequest attribute), 11		row_data (labelbox.schema.data_row.DataRow attribute), 13
project (labelbox.schema.label.Label attribute), 19		S
project (labelbox.schema.labeling_frontend.LabelingFrontend attribute), 23		SchemaId (in module labelbox.schema.ontology), 28
project (labelbox.schema.review.Review attribute), 37		score (labelbox.schema.review.Review attribute), 37
ProjectMember (class in labelbox.schema.project), 35		seconds_to_label (labelbox.schema.label.Label attribute), 19
projects (labelbox.schema.dataset.Dataset attribute), 15		set_labeling_parameter_overrides() (labelbox.schema.project.Project method), 33
		setup() (labelbox.schema.project.Project method), 34
		setup_complete (labelbox.schema.project.Project attribute), 31
		setup_editor() (labelbox.schema.project.Project method), 34
		state (labelbox.schema.bulk_import_request.BulkImportRequest attribute), 11

status (*labelbox.schema.task.Task* attribute), 39  
status (*labelbox.schema.webhook.Webhook* attribute), 43  
status\_file\_url (*labelbox.schema.bulk\_import\_request.BulkImportRequest* attribute), 11  
statuses () (*labelbox.schema.bulk\_import\_request.BulkImportRequest* property), 12  
String (in module *labelbox.schema.data\_row\_metadata*), 56  
upsert\_labels () (*labelbox.schema.model\_run.ModelRun* method), 51  
upsert\_project\_role () (*labelbox.schema.user.User* method), 42  
upsert\_review\_queue () (*labelbox.schema.project.Project* method), 35  
url (*labelbox.schema.webhook.Webhook* attribute), 43  
User (class in *labelbox.schema.user*), 41  
users (*labelbox.schema.organization.Organization* attribute), 29  
UuidError, 46

## T

Task (class in *labelbox.schema.task*), 39  
TimeoutError, 46  
tools (*labelbox.schema.ontology.OntologyBuilder* attribute), 28  
tools () (*labelbox.schema.ontology.Ontology* method), 27  
topics (*labelbox.schema.webhook.Webhook* attribute), 43

## U

unset\_labeling\_parameter\_overrides () (*labelbox.schema.project.Project* method), 34  
update () (*labelbox.schema.project.Project* method), 34  
update () (*labelbox.schema.webhook.Webhook* method), 44  
update\_org\_role () (*labelbox.schema.user.User* method), 42  
updated\_at (*labelbox.schema.data\_row.DataRow* attribute), 13  
updated\_at (*labelbox.schema.dataset.Dataset* attribute), 15  
updated\_at (*labelbox.schema.ontology.Ontology* attribute), 27  
updated\_at (*labelbox.schema.organization.Organization* attribute), 29  
updated\_at (*labelbox.schema.project.Project* attribute), 31  
updated\_at (*labelbox.schema.review.Review* attribute), 37  
updated\_at (*labelbox.schema.task.Task* attribute), 39  
updated\_at (*labelbox.schema.user.User* attribute), 41  
updated\_at (*labelbox.schema.webhook.Webhook* attribute), 43  
upload\_annotations () (*labelbox.schema.project.Project* method), 35  
upsert\_data\_rows () (*labelbox.schema.model\_run.ModelRun* method), 51  
upsert\_instructions () (*labelbox.schema.project.Project* method), 35

## V

ValidationFailedError, 46  
video\_label\_generator () (*labelbox.schema.project.Project* method), 35

## W

wait\_till\_done () (*labelbox.schema.task.Task* method), 39  
wait\_until\_done () (*labelbox.schema.bulk\_import\_request.BulkImportRequest* method), 12  
Webhook (class in *labelbox.schema.webhook*), 43  
Webhook.Status (class in *labelbox.schema.webhook*), 43  
Webhook.Topic (class in *labelbox.schema.webhook*), 43  
webhooks (*labelbox.schema.organization.Organization* attribute), 29  
webhooks (*labelbox.schema.project.Project* attribute), 32