radiant_mlhub

Release 0.2.2

Radiant Earth Foundation

Aug 03, 2021

CONTENTS:

The Python client for the Radiant MLHub API.

CHAPTER

ONE

GETTING STARTED

This guide will walk you through the basic usage of the `radiant_mlhub` library, including:

- Installing & configuring the library
- Discovering & fetching datasets
- Discovering & fetching collections
- Downloading assets

1.1 Installation

1.1.1 Install with `pip`

```
$ pip install radiant_mlhub
```

1.1.2 Install with `conda`

```
$ conda install -c conda-forge radiant-mlhub
```

1.2 Configuration

If you have not done so already, you will need to register for an MLHub API key here.

Once you have your API key, you will need to create a default profile by setting up a `.mlhub/profiles` file in your home directory. You can use the *mlhub configure* command line tool to do this:

```
$ mlhub configure
API Key: Enter your API key here...
Wrote profile to /Users/youruser/.mlhub/profiles
```

Hint: If you do not have write access to the home directory on your machine, you can change the location of the `profiles` file using the `MLHUB_HOME` environment variables. For instance, setting `MLHUB_HOME=/tmp/some-directory/.mlhub` will cause the client to look for your profiles in a `/tmp/some-directory/.mlhub/profiles` file. You may want to permanently set this environment variable to ensure the client continues to look in the correct place for your profiles.

1.3 List Datasets

Once you have your profile configured, you can get a list of the available datasets from the Radiant MLHub API using the *Dataset.list* method. This method is a generator that yields *Dataset* instances. You can use the `id` and `title` properties to get more information about a dataset.

```
>>> from radiant_mlhub import Dataset
>>> for dataset in Dataset.list():
...     print(f'{dataset.id}: {dataset.title}')
'bigearthnet_v1: BigEarthNet V1'
```

1.4 Fetch a Dataset

You can also fetch a dataset by ID using the *Dataset.fetch* method. This method returns a *Dataset* instance.

```
>>> dataset = Dataset.fetch('bigearthnet_v1')
>>> print(f'{dataset.id}: {dataset.title}')
'bigearthnet_v1: BigEarthNet V1'
```

1.5 Work with Dataset Collections

Datasets have 1 or more collections associated with them. Collections fall into 2 types:

- `source_imagery`: Collections of source imagery associated with the dataset
- `labels`: Collections of labeled data associated with the dataset (these collections implement the STAC Label Extension)

To list all the collections associated with a dataset use the *collections* attribute.

```
>>> dataset.collections
[<Collection id=bigearthnet_v1_source>, <Collection id=bigearthnet_v1_labels>]
>>> type(dataset.collections[0])
<class 'radiant_mlhub.models.Collection'>
```

You can also list the collections by type using the `collections.source_imagery` and `collections.labels` properties

```
>>> from pprint import pprint
>>> len(dataset.collections.source_imagery)
1
>>> source_collection = dataset.collections.source_imagery[0]
>>> pprint(source_collection.to_dict())
{'description': 'BigEarthNet v1.0',
 'extent': {'spatial': {'bbox': [[-9.00023345437725,
                                  1.7542686833884724,
                                  83.44558248555553,
                                  68.02168200047284]]},
            'temporal': {'interval': [['2017-06-13T10:10:31Z',
                                       '2018-05-29T11:54:01Z']]}},
 'id': 'bigearthnet_v1_source',
 'keywords': [],
 'license': 'CDLA-Permissive-1.0',
```

(continues on next page)

(continued from previous page)

```
 'links': [{'href': 'https://api.radiant.earth/mlhub/v1/collections/bigearthnet_v1_
↪source',
            'rel': 'self',
            'type': 'application/json'},
           {'href': 'https://api.radiant.earth/mlhub/v1',
            'rel': 'root',
            'type': 'application/json'}],
 'properties': {},
 'providers': [{'name': 'BigEarthNet',
                'roles': ['processor', 'licensor'],
                'url': 'https://api.radiant.earth/mlhub/v1/download/dummy-download-key
↪'}],
 'sci:citation': 'G. Sumbul, M. Charfuelan, B. Demir, V. Markl, "BigEarthNet: '
                 'A Large-Scale Benchmark Archive for Remote Sensing Image '
                 'Understanding", IEEE International Geoscience and Remote '
                 'Sensing Symposium, pp. 5901-5904, Yokohama, Japan, 2019.',
 'stac_extensions': ['eo', 'sci'],
 'stac_version': '1.0.0-beta.2',
 'summaries': {},
 'title': None}
```

1.6 Download a Collection Archive

You can download all the assets associated with a collection using the *Collection.download* method. This method takes a path to a directory on the local file system where the archive should be saved.

If a file of the same name already exists, the client will check whether the downloaded file is complete by comparing its size against the size of the remote file. If they are the same size, the download is skipped, otherwise the download will be resumed from the point where it stopped. You can control this behavior using the `if_exists` argument. Setting this to `"skip"` will skip the download for existing files *without* checking for completeness (a bit faster since it doesn't require a network request), and setting this to `"overwrite"` will overwrite any existing file.

```
>>> source_collection.download('~/Downloads')
28%|       | 985.0/3496.9 [00:35<00:51, 48.31M/s]
```

Collection archives are gzipped tarballs. You can read more about the structure of these archives in this Medium post.

CHAPTER

TWO

AUTHENTICATION

The Radiant MLHub API uses API keys to authenticate users. These keys must be passed as a `key` query parameter in any request made to the API. Anyone can register for an API key by going to https://dashboard.mlhub.earth and creating an account. Once you have logged into your account, go to http://dashboard.mlhub.earth/api-keys to create API keys.

2.1 Using API Keys

The best way to add your API key to requests is to create a *`Session`* instance using the *`get_session()`* helper function and making requests using this instance:

```
>>> from radiant_mlhub import get_session
>>> session = get_session()
>>> r = session.get(...)
```

You can associate an API key with a session in a number of ways:

- programmatically via an instantiation argument
- using environment variables
- using a named profile

The *`Session`* resolves an API key by trying each of the following (in this order):

1) Use an `api_key` argument provided during instantiation

```
>>> session = get_session(api_key='myapikey')
```

2) Use an `MLHUB_API_KEY` environment variable

```
>>> import os
>>> os.environ['MLHUB_API_KEY'] = 'myapikey'
>>> session = get_session()
```

3) Use a `profile` argument provided during instantiation (see *Using Profiles* section for details)

```
>>> session = get_session(profile='my-profile')
```

4) Use an `MLHUB_PROFILE` environment variable (see *Using Profiles* section for details)

```
>>> os.environ['MLHUB_PROFILE'] = 'my-profile'
>>> session = get_session()
```

5) Use the `default` profile (see *Using Profiles* section for details)

```
>>> session = get_session()
```

If none of the above strategies results in a valid API key, then an `APIKeyNotFound` exception is raised.

The *`radiant_mlhub.session.Session`* instance inherits from `requests.Session` and adds a few conveniences to a typical session:

- Adds the API key as a `key` query parameter
- Adds an `Accept: application/json` header
- Adds a `User-Agent` header that contains the package name and version, plus basic system information like the the OS name
- Prepends the MLHub root URL (`https://api.radiant.earth/mlhub/v1/`) to any request paths without a domain
- Raises a *`radiant_mlhub.exceptions.AuthenticationError`* for `401 (UNAUTHORIZED)` responses

2.2 Using Profiles

Profiles in `radiant_mlhub` are inspired by the Named Profiles used by `boto3` and `awscli`. These named profiles provide a way to store API keys (and potentially other configuration) on your local system so that you do not need to explicitly set environment variables or pass in arguments every time you create a session.

All profile configuration must be stored in a `.mlhub/profiles` file in your home directory. The `profiles` file uses the INI file structure supported by Python's `configparser` module as described here.

Hint: If you do not have write access to the home directory on your machine, you can change the location of the `profiles` file using the `MLHUB_HOME` environment variables. For instance, setting `MLHUB_HOME=/tmp/some-directory/.mlhub` will cause the client to look for your profiles in a `/tmp/some-directory/.mlhub/profiles` file. You may want to permanently set this environment variable to ensure the client continues to look in the correct place for your profiles.

The easiest way to configure a profile is using the `mlhub configure` CLI tool documented in the *CLI Tools section*:

```
$ mlhub configure
API Key: <Enter your API key when prompted>
Wrote profile to /Users/youruser/.mlhub/profiles
```

Given the following `profiles` file...

```
[default]
api_key = default_api_key

[project1]
api_key = some_other_api_key

[project2]
api_key = yet_another_api_key
```

These would be the API keys used by sessions created using the various methods described in *Using API Keys*:

```
# As long as we haven't set the MLHUB_API_KEY or MLHUB_PROFILE environment variables
#  this will pull from the default profile
>>> session = get_session()
>>> session.params['key']
'default_api_key'

# Setting the MLHUB_PROFILE environment variable overrides the default profile
>>> os.environ['MLHUB_PROFILE'] = 'project1'
>>> session = get_session()
>>> session.params['key']
'some_other_api_key'

# Passing the profile argument directly overrides the MLHUB_PROFILE environment␣
↪variable
>>> session = get_session(profile='profile2')
>>> session.params['key']
'yet_another_api_key'

# Setting the MLHUB_API_KEY environment variable overrides any profile-related␣
↪arguments
>>> os.environ['MLHUB_API_KEY'] = 'environment_direct'
>>> session = get_session()
>>> session.params['key']
'environment_direct'

# Passing the api_key argument overrides all other strategies or finding the key
>>> session = get_session(api_key='argument_direct')
>>> session.params['key']
'argument_direct'
```

2.3 Making API Requests

Once you have your `profiles` file in place, you can create a session that will be used to make authenticated requests to the API:

```
>>> from radiant_mlhub import get_session
>>> session = get_session()
```

You can use this session to make authenticated calls to the API. For example, to list all collections:

```
>>> r = session.get('/collections')  # Leading slash is optional
>>> collections = r.json()['collections']
>>> print(len(collections))
47
```

2.3.1 Relative v. Absolute URLs

Any URLs that do not include a scheme (`http://`, `https://`) are assumed to be relative to the Radiant MLHub root URL. For instance, the following code would make a request to `https://api.radiant.earth/mlhub/v1/some-endpoint`:

```
>>> session.get('some-endpoint')
```

but the following code would make a request to `https://www.google.com`:

```
>>> session.get('https://www.google.com')
```

It is not recommended to make calls to APIs other than the Radiant MLHub API using these sessions.

CHAPTER

THREE

COLLECTIONS

A **collection** represents either a group of related labels or a group of related source imagery for a given time period and geographic area. All collections in the Radiant MLHub API are valid STAC Collections. For instance, the `ref_landcovernet_v1_source` collection catalogs the source imagery associated with the LandCoverNet dataset, while the `ref_landcovernet_v1_labels` collection catalogs the land cover labels associated with this imagery. These collections are considered part of a single `ref_landcovernet_v1` **dataset** (see the *Datasets* documentation for details on working with datasets).

To discover and fetch collections you can either use the low-level client methods from *`radiant_mlhub.client`* or the *`Collection`* class. Using the *`Collection`* class is the recommended approach, but both methods are described below.

3.1 Discovering Collections

The Radiant MLHub `/collections` endpoint returns a list of objects describing the available collections. You can use the low-level *`list_collections()`* function to work with these responses as native Python data types (`list` and `dict`). This function returns a list of JSON-like dictionaries representing STAC Collections.

```
>>> from radiant_mlhub.client import list_collections
>>> from pprint import pprint
>>> collections = list_collections()
>>> first_collection = collections[0]
>>> pprint(first_collection)
{'description': 'African Crops Kenya',
 'extent': {'spatial': {'bbox': [[34.18191992149459,
                                  0.4724181558451209,
                                  34.3714943155646,
                                  0.71442172068511091]]},
            'temporal': {'interval': [['2018-04-10T00:00:00Z',
                                       '2020-03-13T00:00:00Z']]}},
 'id': 'ref_african_crops_kenya_01_labels',
 'keywords': [],
 'license': 'CC-BY-SA-4.0',
 'links': [{'href': 'https://api.radiant.earth/mlhub/v1/collections/ref_african_crops_
↪kenya_01_labels',
            'rel': 'self',
            'title': None,
            'type': 'application/json'},
           {'href': 'https://api.radiant.earth/mlhub/v1',
            'rel': 'root',
            'title': None,
            'type': 'application/json'}],
```

(continues on next page)

(continued from previous page)

```
 'properties': {},
 'providers': [{'description': None,
                'name': 'Radiant Earth Foundation',
                'roles': ['licensor', 'host', 'processor'],
                'url': 'https://radiant.earth'}],
 'sci:citation': 'PlantVillage. (2019) PlantVillage Kenya Ground Reference '
                 'Crop Type Dataset, Version 1. [Indicate subset used]. '
                 'Radiant ML Hub. [Date Accessed]',
 'sci:doi': '10.34911/rdnt.u41j87',
 'stac_extensions': [],
 'stac_version': '1.0.0-beta.2',
 'summaries': {},
 'title': None}
```

You can also discover collections using the *Collection.list* method. This is the recommended way of listing datasets. This method returns a list of *Collection* instances.

```
>>> from radiant_mlhub import Collection
>>> collections = Collection.list()
>>> first_collection = collections[0]
>>> first_collection.ref_african_crops_kenya_01_labels
'ref_african_crops_kenya_01_labels'
>>> first_collection.description
'African Crops Kenya'
```

3.2 Fetching a Collection

The Radiant MLHub `/collections/{p1}` endpoint returns an object representing a single collection. You can use the low-level *get_collection()* function to work with this response as a `dict`.

```
>>> from radiant_mlhub.client import get_collection
>>> collection = get_collection('ref_african_crops_kenya_01_labels')
>>> pprint(collection)
{'description': 'African Crops Kenya',
 'extent': {'spatial': {'bbox': [[34.18191992149459,
                                  0.4724181558451209,
                                  34.3714943155646,
                                  0.7144217206851109]]},
            'temporal': {'interval': [['2018-04-10T00:00:00Z',
                                       '2020-03-13T00:00:00Z']]}},
 'id': 'ref_african_crops_kenya_01_labels',
 ...
 }
```

You can also fetch a collection from the Radiant MLHub API based on the collection ID using the *Collection.fetch* method. This is the recommended way of fetching a collection. This method returns a *Collection* instance.

```
>>> collection = Collection.fetch('ref_african_crops_kenya_01_labels')
>>> collection.id
'ref_african_crops_kenya_01_labels'
>>> collection.description
'African Crops Kenya'
```

For more information on a Collection, you can check out the MLHub Registry page:

```
>>> collection.registry_url
https://registry.mlhub.earth/10.14279/depositonce-10149/
```

3.3 Downloading a Collection

The Radiant MLHub `/archive/{archive_id}` endpoint allows you to download an archive of all assets associated with a given collection. You can use the low-level *`download_archive()`* function to download the archive to your local file system.

```
>>> from radiant_mlhub.client import download_archive
>>> archive_path = download_archive('sn1_AOI_1_RIO')
28%|       | 985.0/3496.9 [00:35<00:51, 48.31M/s]
>>> archive_path
PosixPath('/path/to/current/directory/sn1_AOI_1_RIO.tar.gz')
```

You can also download a collection archive using the *`Collection.download`* method. This is the recommended way of downloading an archive.

```
>>> collection = Collection.fetch('sn1_AOI_1_RIO')
>>> archive_path = collection.download('~/Downloads', exist_okay=False)  # Will raise␣
↪exception if the file already exists
28%|       | 985.0/3496.9 [00:35<00:51, 48.31M/s]
>>> archive_path
PosixPath('/Users/someuser/Downloads/sn1_AOI_1_RIO.tar.gz')
```

If a file of the same name already exists, these methods will check whether the downloaded file is complete by comparing its size against the size of the remote file. If they are the same size, the download is skipped, otherwise the download will be resumed from the point where it stopped. You can control this behavior using the `if_exists` argument. Setting this to `"skip"` will skip the download for existing files *without* checking for completeness (a bit faster since it doesn't require a network request), and setting this to `"overwrite"` will overwrite any existing file.

To check the size of the download archive without actually downloading it, you can use the `Collection.total_archive_size` property.

```
>>> collection.archive_size
3504256089
```

Collection archives are gzipped tarballs. You can read more about the structure of these archives in this Medium post.

CHAPTER

FOUR

DATASETS

A **dataset** represents a group of 1 or more related STAC Collections. They group together any source imagery Collections with the associated label Collections to provide a convenient mechanism for accessing all of these data together. For instance, the `bigearthnet_v1_source` Collection contains the source imagery for the BigEarthNet training dataset and, likewise, the `bigearthnet_v1_labels` Collection contains the annotations for that same dataset. These 2 collections are grouped together into the `bigearthnet_v1` dataset.

The Radiant MLHub Training Data Registry provides an overview of the datasets available through the Radiant MLHub API along with dataset metadata and a listing of the associated Collections.

To discover and fetch datasets you can either use the low-level client methods from *radiant_mlhub.client* or the *Dataset* class. Using the *Dataset* class is the recommended approach, but both methods are described below.

Note: The objects returned by the Radiant MLHub API dataset endpoints are not STAC-compliant objects and therefore the `Dataset` class described below is not a PySTAC object.

4.1 Discovering Datasets

The Radiant MLHub `/datasets` endpoint returns a list of objects describing the available datasets and their associated collections. You can use the low-level *list_datasets()* function to work with these responses as native Python data types (`list` and `dict`). This function is a generator that yields a `dict` for each dataset.

```
>>> from radiant_mlhub.client import list_datasets
>>> from pprint import pprint
>>> datasets = list_datasets()
>>> first_dataset = datasets[0]
>>> pprint(first_dataset)
{'collections': [{'id': 'bigearthnet_v1_source', 'types': ['source_imagery']},
                 {'id': 'bigearthnet_v1_labels', 'types': ['labels']}],
 'id': 'bigearthnet_v1',
 'title': 'BigEarthNet V1'}
```

You can also discover datasets using the *Dataset.list* method. This is the recommended way of listing datasets. This method returns a list of *Dataset* instances.

```
>>> from radiant_mlhub import Dataset
>>> datasets = Dataset.list()
>>> first_dataset = datasets[0]
>>> first_dataset.id
'bigearthnet_v1'
```

(continues on next page)

(continued from previous page)

```
>>> first_dataset.title
'BigEarthNet V1'
```

4.2 Fetching a Dataset

The Radiant MLHub `/datasets/{dataset_id}` endpoint returns an object representing a single dataset. You can use the low-level *`get_dataset()`* function to work with this response as a `dict`.

```
>>> from radiant_mlhub.client import get_dataset
>>> dataset = get_dataset('bigearthnet_v1')
>>> pprint(dataset)
{'collections': [{'id': 'bigearthnet_v1_source', 'types': ['source_imagery']},
                 {'id': 'bigearthnet_v1_labels', 'types': ['labels']}],
 'id': 'bigearthnet_v1',
 'title': 'BigEarthNet V1'}
```

You can also fetch a dataset from the Radiant MLHub API based on the dataset ID using the *`Dataset.fetch`* method. This is the recommended way of fetching a dataset. This method returns a *`Dataset`* instance.

```
>>> dataset = Dataset.fetch('bigearthnet_v1')
>>> dataset.id
'bigearthnet_v1'
```

4.3 Dataset Collections

If you are using the *`Dataset`* class, you can list the Collections associated with the dataset using the *`Dataset.collections`* property. This method returns a modified `list` that has 2 additional attributes: `source_imagery` and `labels`. You can use these attributes to list only the collections of a the associated type. All elements of these lists are instances of *`Collection`*. See the *Collections* documentation for details on how to work with these instances.

```
>>> len(first_dataset.collections)
2
>>> len(first_dataset.collections.source_imagery)
1
>>> first_dataset.collections.source_imagery[0].id
'bigearthnet_v1_source'
>>> len(first_dataset.collections.labels)
1
>>> first_dataset.collections.labels[0].id
'bigearthnet_v1_labels'
```

Warning: There are rare cases of collections that contain both `source_imagery` and `labels` items (e.g. the SpaceNet collections). In these cases, the collection will be listed in both the `dataset.collections.labels` and `dataset.collections.source_imagery` lists, but *will only appear once in the main ``dataset.collections`` list*. This may cause what appears to be a mismatch in list lengths:

```
>>> len(dataset.collections.source_imagery) + len(dataset.collections.labels) ==␣
↪len(dataset.collections)
False
```

Note: Both the low-level client functions and the class methods also accept keyword arguments that are passed directly to *get_session()* to create a session. See the *Authentication* documentation for details on how to use these arguments or configure the client to read your API key automatically.

4.4 Downloading a Dataset

The Radiant MLHub `/archive/{archive_id}` endpoint allows you to download an archive of all assets associated with a given collection. The *Dataset.download* method provides a convenient way of using this endpoint to download the archives for all collections associated with a given dataset. This method downloads the archives for all associated collections into the given output directory and returns a list of the paths to these archives.

If a file of the same name already exists for any of the archives, this method will check whether the downloaded file is complete by comparing its size against the size of the remote file. If they are the same size, the download is skipped, otherwise the download will be resumed from the point where it stopped. You can control this behavior using the `if_exists` argument. Setting this to `"skip"` will skip the download for existing files *without* checking for completeness (a bit faster since it doesn't require a network request), and setting this to `"overwrite"` will overwrite any existing file.

```
>>> dataset = Collection.fetch('bigearthnet_v1')
>>> archive_paths = dataset.download('~/Downloads')
>>> len(archive_paths)
2
```

To check the total size of the download archives for all collections in the dataset without actually downloading it, you can use the `Dataset.total_archive_size` property.

```
>>> dataset.total_archive_size
71311240007
```

Collection archives are gzipped tarballs. You can read more about the structure of these archives in this Medium post.

CHAPTER

FIVE

RADIANT_MLHUB PACKAGE

5.1 Submodules

5.2 radiant_mlhub.client module

Low-level functions for making requests to MLHub API endpoints.

radiant_mlhub.client.**download_archive**(*archive_id: str*, *output_dir: Optional[pathlib.Path] = None*, ***, *if_exists: str = 'resume'*, ***session_kwargs*) → pathlib.Path

Downloads the archive with the given ID to an output location (current working directory by default).

The `if_exists` argument determines how to handle an existing archive file in the output directory. The default behavior (defined by `if_exists="resume"`) is to resume the download by requesting a byte range starting at the size of the existing file. If the existing file is the same size as the file to be downloaded (as determined by the `Content-Length` header), then the download is skipped. You can automatically skip download using `if_exists="skip"` (this may be faster if you know the download was not interrupted, since no network request is made to get the archive size). You can also overwrite the existing file using `if_exists="overwrite"`.

Parameters

- **archive_id** (*str*) – The ID of the archive to download. This is the same as the Collection ID.
- **output_dir** (*Path*) – Path to which the archive will be downloaded. Defaults to the current working directory.
- **if_exists** (*str, optional*) – How to handle an existing archive at the same location. If `"skip"`, the download will be skipped. If `"overwrite"`, the existing file will be overwritten and the entire file will be re-downloaded. If `"resume"` (the default), the existing file size will be compared to the size of the download (using the `Content-Length` header). If the existing file is smaller, then only the remaining portion will be downloaded. Otherwise, the download will be skipped.
- ****session_kwargs** – Keyword arguments passed directly to *get_session()*

Returns **output_path** – The full path to the downloaded archive file.

Return type Path

Raises **ValueError** – If `if_exists` is not one of `"skip"`, `"overwrite"`, or `"resume"`.

radiant_mlhub.client.**get_archive_info**(*archive_id: str*, ***session_kwargs*) → dict

Gets info for the given archive from the `/archive/{archive_id}/info` endpoint as a JSON-like dictionary.

The JSON object returned by the API has the following properties:

- `collection`: The ID of the Collection that this archive is associated with.
- `dataset`: The ID of the dataset that this archive's Collection belongs to.
- `size`: The size of the archive (in bytes)
- `types`: The types associated with this archive's Collection. Will be one of `"source_imagery"` or `"label"`.

Parameters

- **archive_id** (*str*) – The ID of the archive. This is the same as the Collection ID.
- ****session_kwargs** – Keyword arguments passed directly to *get_session()*

Returns **archive_info** – JSON-like dictionary representing the API response.

Return type dict

radiant_mlhub.client.**get_collection**(*collection_id: str*, ***session_kwargs*) → dict

Returns a JSON-like dictionary representing the response from the Radiant MLHub `GET /collections/{p1}` endpoint.

See the MLHub API docs for details.

Parameters

- **collection_id** (*str*) – The ID of the collection to fetch
- ****session_kwargs** – Keyword arguments passed directly to *get_session()*

Returns **collection**

Return type dict

Raises

- ***EntityDoesNotExist*** – If a 404 response code is returned by the API
- ***MLHubException*** – If any other response code is returned

radiant_mlhub.client.**get_collection_item**(*collection_id: str*, *item_id: str*, ***session_kwargs*) → dict

Returns a JSON-like dictionary representing the response from the Radiant MLHub `GET /collections/{p1}/items/{p2}` endpoint.

Parameters

- **collection_id** (*str*) – The ID of the Collection to which the Item belongs.
- **item_id** (*str*) – The ID of the Item.
- ****session_kwargs** – Keyword arguments passed directly to *get_session()*

Returns **item**

Return type dict

radiant_mlhub.client.**get_dataset**(*dataset_id: str*, ***session_kwargs*) → dict

Returns a JSON-like dictionary representing the response from the Radiant MLHub `GET /datasets/{dataset_id}` endpoint.

See the MLHub API docs for details.

Parameters

- **dataset_id** (*str*) – The ID of the dataset to fetch
- ****session_kwargs** – Keyword arguments passed directly to *get_session()*

Returns dataset

Return type dict

radiant_mlhub.client.**list_collection_items**(*collection_id: str*, ***, *page_size: Optional[int] = None*, *extensions: Optional[List[str]] = None*, *limit: int = 10*, ***session_kwargs*) → Iterator[dict]

Yields JSON-like dictionaries representing STAC Item objects returned by the Radiant MLHub `GET /collections/{collection_id}/items` endpoint.

Note: Because some collections may contain hundreds of thousands of items, this function limits the total number of responses to `10` by default. You can change this value by increasing the value of the `limit` keyword argument, or setting it to `None` to list all items. **Be aware that trying to list all items in a large collection may take a very long time.**

Parameters

- **collection_id** (*str*) – The ID of the collection from which to fetch items
- **page_size** (*int*) – The number of items to return in each page. If set to `None`, then this parameter will not be passed to the API and the default API value will be used (currently `30`).
- **extensions** (*list*) – If provided, then only items that support all of the extensions listed will be returned.
- **limit** (*int*) – The maximum *total* number of items to yield. Defaults to `10`.
- ****session_kwargs** – Keyword arguments passed directly to *get_session()*

Yields **item** (*dict*) – JSON-like dictionary representing a STAC Item associated with the given collection.

radiant_mlhub.client.**list_collections**(***session_kwargs*) → List[dict]

Gets a list of JSON-like dictionaries representing STAC Collection objects returned by the Radiant MLHub `GET /collections` endpoint.

See the MLHub API docs for details.

Parameters ****session_kwargs** – Keyword arguments passed directly to *get_session()*

Returns **collections** – List of JSON-like dictionaries representing STAC Collection objects.

Return type List[dict]

radiant_mlhub.client.**list_datasets**(***session_kwargs*) → List[dict]

Gets a list of JSON-like dictionaries representing dataset objects returned by the Radiant MLHub `GET /datasets` endpoint.

See the MLHub API docs for details.

Parameters ****session_kwargs** – Keyword arguments passed directly to *get_session()*

Returns **datasets**

Return type List[dict]

5.3 radiant_mlhub.exceptions module

exception radiant_mlhub.exceptions.**APIKeyNotFound**
Bases: *radiant_mlhub.exceptions.MLHubException*

Raised when an API key cannot be found using any of the strategies described in the *Authentication* docs.

exception radiant_mlhub.exceptions.**AuthenticationError**
Bases: *radiant_mlhub.exceptions.MLHubException*

Raised when the Radiant MLHub API cannot authenticate the request, either because the API key is invalid or expired, or because no API key was provided in the request.

exception radiant_mlhub.exceptions.**EntityDoesNotExist**
Bases: *radiant_mlhub.exceptions.MLHubException*

Raised when attempting to fetch a collection that does not exist in the Radiant MLHub API.

exception radiant_mlhub.exceptions.**MLHubException**
Bases: Exception

Base exception class for all Radiant MLHub exceptions

5.4 radiant_mlhub.models module

Extensions of the PySTAC classes that provide convenience methods for interacting with the Radiant MLHub API.

class radiant_mlhub.models.**Collection**(*id*, *description*, *extent*, *title*, *stac_extensions*, *href*, *extra_fields*, *catalog_type*, *license*, *keywords*, *providers*, *properties*, *summaries*, *, *api_key=None*, *profile=None*)
Bases: pystac.collection.Collection

Class inheriting from pystac.Collection that adds some convenience methods for listing and fetching from the Radiant MLHub API.

property archive_size
The size of the tarball archive for this collection in bytes (or None if the archive does not exist).

download(*output_dir: pathlib.Path*, *, *if_exists: str = 'resume'*, ***session_kwargs*) → pathlib.Path
Downloads the archive for this collection to an output location (current working directory by default). If the parent directories for output_path do not exist, they will be created.

The if_exists argument determines how to handle an existing archive file in the output directory. See the documentation for the *download_archive()* function for details. The default behavior is to resume downloading if the existing file is incomplete and skip the download if it is complete.

Note: Some collections may be very large and take a significant amount of time to download, depending on your connection speed.

Parameters

- **output_dir** (*Path*) – Path to a local directory to which the file will be downloaded. File name will be generated automatically based on the download URL.
- **if_exists** (*str, optional*) – How to handle an existing archive at the same location. If "skip", the download will be skipped. If "overwrite", the existing

file will be overwritten and the entire file will be re-downloaded. If `"resume"` (the default), the existing file size will be compared to the size of the download (using the `Content-Length` header). If the existing file is smaller, then only the remaining portion will be downloaded. Otherwise, the download will be skipped.

- **`**session_kwargs`** – Keyword arguments passed directly to *`get_session()`*

Returns **output_path** – The path to the downloaded archive file.

Return type pathlib.Path

Raises **`FileExistsError`** – If file at `output_path` already exists and both `exist_okay` and `overwrite` are `False`.

classmethod fetch(*collection_id: str*, ***session_kwargs*) → *radiant_mlhub.models.Collection*

Creates a *`Collection`* instance by fetching the collection with the given ID from the Radiant MLHub API.

Parameters

- **`collection_id`** (*`str`*) – The ID of the collection to fetch (e.g. `bigearthnet_v1_source`).
- **`**session_kwargs`** – Keyword arguments passed directly to *`get_session()`*

Returns **collection**

Return type *Collection*

fetch_item(*item_id: str*, ***session_kwargs*) → pystac.item.Item

classmethod from_dict(*d*, *href=None*, *root=None*, ***, *api_key=None*, *profile=None*)

Patches the `pystac.Collection.from_dict()` method so that it returns the calling class instead of always returning a `pystac.Collection` instance.

get_items(***session_kwargs*) → Iterator[pystac.item.Item]

Note: The `get_items` method is not implemented for Radiant MLHub *`Collection`* instances for performance reasons. Please use the *`Collection.download()`* method to download Collection assets.

Raises **`NotImplementedError`** –

classmethod list(***session_kwargs*) → List[*radiant_mlhub.models.Collection*]

Returns a list of *`Collection`* instances for all collections hosted by MLHub.

See the *Authentication* documentation for details on how authentication is handled for this request.

Parameters **`**session_kwargs`** – Keyword arguments passed directly to *`get_session()`*

Returns **collections**

Return type List[*Collection*]

property registry_url

The URL of the registry page for this Collection. The URL is based on the DOI identifier for the collection. If the Collection does not have a `"sci:doi"` property then `registry_url` will be `None`.

class radiant_mlhub.models.**CollectionType**(*value*)

Bases: enum.Enum

Valid values for the type of a collection associated with a Radiant MLHub dataset.

LABELS = 'labels'

SOURCE = 'source_imagery'

class radiant_mlhub.models.**Dataset**(*id: str*, *collections: List[dict]*, *title: Optional[str] = None*, *registry: Optional[str] = None*, *doi: Optional[str] = None*, *citation: Optional[str] = None*, ***, *api_key: Optional[str] = None*, *profile: Optional[str] = None*, ***_*)

Bases: object

Class that brings together multiple Radiant MLHub "collections" that are all considered part of a single "dataset". For instance, the bigearthnet_v1 dataset is composed of both a source imagery collection (bigearthnet_v1_source) and a labels collection (bigearthnet_v1_labels).

id

The dataset ID.

Type str

title

The title of the dataset (or None if dataset has no title).

Type str or None

registry_url

The URL to the registry page for this dataset, or None if no registry page exists.

Type str or None

doi

The DOI identifier for this dataset, or None if there is no DOI for this dataset.

Type str or None

citation

The citation information for this dataset, or None if there is no citation information.

Type str or None

property collections

List of collections associated with this dataset. The list that is returned has 2 additional attributes (source_imagery and labels) that represent the list of collections corresponding the each type.

Note: This is a cached property, so updating self.collection_descriptions after calling self.collections the first time will have no effect on the results. See functools.cached_property() for details on clearing the cached value.

Examples

```
>>> from radiant_mlhub import Dataset
>>> dataset = Dataset.fetch('bigearthnet_v1')
>>> len(dataset.collections)
2
>>> len(dataset.collections.source_imagery)
1
>>> len(dataset.collections.labels)
1
```

To loop through all collections

```
>>> for collection in dataset.collections:
...     # Do something here
```

To loop through only the source imagery collections:

```
>>> for collection in dataset.collections.source_imagery:
...     # Do something here
```

To loop through only the label collections:

```
>>> for collection in dataset.collections.labels:
...     # Do something here
```

download(*output_dir: Union[pathlib.Path, str]*, *, *if_exists: str = 'resume'*, ***session_kwargs*) → List[pathlib.Path]

Downloads archives for all collections associated with this dataset to given directory. Each archive will be named using the collection ID (e.g. some_collection.tar.gz). If `output_dir` does not exist, it will be created.

Note: Some collections may be very large and take a significant amount of time to download, depending on your connection speed.

Parameters

- **output_dir** (*str or pathlib.Path*) – The directory into which the archives will be written.
- **if_exists** (*str, optional*) – How to handle an existing archive at the same location. If `"skip"`, the download will be skipped. If `"overwrite"`, the existing file will be overwritten and the entire file will be re-downloaded. If `"resume"` (the default), the existing file size will be compared to the size of the download (using the `Content-Length` header). If the existing file is smaller, then only the remaining portion will be downloaded. Otherwise, the download will be skipped.
- **session_kwargs** – Keyword arguments passed directly to *get_session()*

Returns **output_paths** – List of paths to the downloaded archives

Return type List[pathlib.Path]

Raises

- **IOError** – If `output_dir` exists and is not a directory.

- **FileExistsError** – If one of the archive files already exists in the `output_dir` and both `exist_okay` and `overwrite` are `False`.

classmethod fetch(*dataset_id: str*, ***session_kwargs*) → *radiant_mlhub.models.Dataset*

Creates a *Dataset* instance by fetching the dataset with the given ID from the Radiant MLHub API.

Parameters

- **dataset_id** (*str*) – The ID of the dataset to fetch (e.g. `bigearthnet_v1`).
- ****session_kwargs** – Keyword arguments passed directly to *get_session()*.

Returns dataset

Return type *Dataset*

classmethod list(***session_kwargs*) → List[*radiant_mlhub.models.Dataset*]

Returns a list of *Dataset* instances for each datasets hosted by MLHub.

See the *Authentication* documentation for details on how authentication is handled for this request.

Parameters ****session_kwargs** – Keyword arguments passed directly to *get_session()*

Yields **dataset** (*Dataset*)

property total_archive_size

Gets the total size (in bytes) of the archives for all collections associated with this dataset. If no archives exist, returns `None`.

5.5 radiant_mlhub.session module

Methods and classes to simplify constructing and authenticating requests to the MLHub API.

It is generally recommended that you use the *get_session()* function to create sessions, since this will propertly handle resolution of the API key from function arguments, environment variables, and profiles as described in *Authentication*. See the *get_session()* docs for usage examples.

class radiant_mlhub.session.**Session**(*, *api_key: Optional[str]*)

Bases: `requests.sessions.Session`

Custom class inheriting from `requests.Session` with some additional conveniences:

- Adds the API key as a `key` query parameter
- Adds an `Accept: application/json` header
- Adds a `User-Agent` header that contains the package name and version, plus basic system information like the OS name
- Prepends the MLHub root URL (`https://api.radiant.earth/mlhub/v1/`) to any request paths without a domain
- Raises a *radiant_mlhub.exceptions.AuthenticationError* for `401 (UNAUTHORIZED)` responses
- Calls `requests.Response.raise_for_status()` after all requests to raise exceptions for any status codes above 400.

API_KEY_ENV_VARIABLE = 'MLHUB_API_KEY'

DEFAULT_ROOT_URL = 'https://api.radiant.earth/mlhub/v1/'

MLHUB_HOME_ENV_VARIABLE = 'MLHUB_HOME'

PROFILE_ENV_VARIABLE = 'MLHUB_PROFILE'

ROOT_URL_ENV_VARIABLE = 'MLHUB_ROOT_URL'

classmethod from_config(*profile: Optional[str] = None*) → *radiant_mlhub.session.Session*

Create a session object by reading an API key from the given profile in the `profiles` file. By default, the client will look for the `profiles` file in a `.mlhub` directory in the user's home directory (as determined by `Path.home`). However, if an `MLHUB_HOME` environment variable is present, the client will look in that directory instead.

Parameters **profile** (*str, optional*) – The name of a profile configured in the `profiles` file.

Returns session

Return type *Session*

Raises ***APIKeyNotFound*** – If the given config file does not exist, the given profile cannot be found, or there is no `api_key` property in the given profile section.

classmethod from_env() → *radiant_mlhub.session.Session*

Create a session object from an API key from the environment variable.

Returns session

Return type *Session*

Raises ***APIKeyNotFound*** – If the API key cannot be found in the environment

paginate(*url: str*, ***kwargs*) → Iterator[dict]

Makes a GET request to the given `url` and paginates through all results by looking for a link in each response with a `rel` type of `"next"`. Any additional keyword arguments are passed directly to `requests.Session.get()`.

Parameters **url** (*str*) – The URL to which the initial request will be made. Note that this may either be a full URL or a path relative to the `ROOT_URL` as described in *`Session.request()`*.

Yields **page** (*dict*) – An individual response as a dictionary.

request(*method*, *url*, *params=None*, *data=None*, *headers=None*, *cookies=None*, *files=None*, *auth=None*, *timeout=None*, *allow_redirects=True*, *proxies=None*, *hooks=None*, *stream=None*, *verify=None*, *cert=None*, *json=None*)

Overwrites the default `requests.Session.request()` method to prepend the MLHub root URL if the given `url` does not include a scheme. This will raise an *`AuthenticationError`* if a 401 response is returned by the server, and a `HTTPError` if any other status code of 400 or above is returned.

Parameters

- **method** (*str*) – The request method to use. Passed directly to the `method` argument of `requests.Session.request()`
- **url** (*str*) – Either a full URL or a path relative to the `ROOT_URL`. For example, to make a request to the Radiant MLHub API `/collections` endpoint, you could use `session.get('collections')`.
- ****kwargs** – All other keyword arguments are passed directly to `requests.Session.request()` (see that documentation for an explanation of these keyword arguments).

Raises

- ***AuthenticationError*** – If the response status code is 401

- **HTTPError** – For all other response status codes at or above 400

radiant_mlhub.session.**get_session**(*, *api_key: Optional[str] = None*, *profile: Optional[str] = None*) → *radiant_mlhub.session.Session*

Gets a *Session* object that uses the given `api_key` for all requests. If no `api_key` argument is provided then the function will try to resolve an API key by finding the following values (in order of preference):

1) An `MLHUB_API_KEY` environment variable
2) A `api_key` value found in the given `profile` section of `~/.mlhub/profiles`
3) A `api_key` value found in the given `default` section of `~/.mlhub/profiles`

Parameters

- **api_key** (*str, optional*) – The API key to use for all requests from the session. See description above for how the API key is resolved if not provided as an argument.
- **profile** (*str, optional*) – The name of a profile configured in the `.mlhub/profiles` file. This will be passed directly to *from_config()*.

Returns session

Return type *Session*

Raises ***APIKeyNotFound*** – If no API key can be resolved.

Examples

```
>>> from radiant_mlhub import get_session
# Get the API from the "default" profile
>>> session = get_session()
# Get the session from the "project1" profile
# Alternatively, you could set the MLHUB_PROFILE environment variable to "project1
↪"
>>> session = get_session(profile='project1')
# Pass an API key directly to the session
# Alternatively, you could set the MLHUB_API_KEY environment variable to "some-
↪api-key"
>>> session = get_session(api_key='some-api-key')
```

5.6 Module contents

CHAPTER

SIX

CLI TOOLS

6.1 mlhub

CLI tool for the radiant_mlhub Python client.

```
mlhub [OPTIONS] COMMAND [ARGS]...
```

Options

--version
Show the version and exit.

6.1.1 configure

Interactively set up radiant_mlhub configuration file.

This tool walks you through setting up a ~/.mlhub/profiles file and adding an API key. If you do not provide a –profile option, it will update the "default" profile. If you do not provide an –api-key option, you will be prompted to enter an API key by the tool.

If you need to change the location of the profiles file, set the MLHUB_HOME environment variable before running this command.

For details on profiles and authentication for the radiant_mlhub client, please see the official Authentication documentation:

https://radiant-mlhub.readthedocs.io

```
mlhub configure [OPTIONS]
```

Options

--profile <profile>
The name of the profile to configure.

--api-key <api_key>
The API key to use for this profile.

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

r

INDEX

Symbols

A

C

D

E

F

G

I

L

M

P

R

S

T