

Onyx | CLI & Python API

CLIMB-TRE

Table of contents

1. CLI & Python API	3
1.1 CLI & Python API for Onyx	3
1.2 Installation	5
1.3 Accessibility	6
1.4 Command-line Interface	7
1.5 Python API	20

1. CLI & Python API

1.1 CLI & Python API for Onyx

1.1.1 Introduction

This is the documentation for [Onyx-client](#), a program that provides a command-line interface and Python API for interacting with the [Onyx](#) database.

Onyx is being developed as part of the [CLIMB-TRE](#) project.

A PDF of this documentation can be found [here](#).

```
Usage: onyx [OPTIONS] COMMAND [ARGS]...

API for Pathogen Metadata.
For documentation, see: https://climb-tre.github.io/onyx-client/

Options
--domain -d TEXT Domain name for connecting to Onyx. [env var: ONYX_DOMAIN] [default: None]
--token -t TEXT Token for authenticating with Onyx. [env var: ONYX_TOKEN] [default: None]
--username -u TEXT Username for authenticating with Onyx. [env var: ONYX_USERNAME] [default: None]
--password -p TEXT Password for authenticating with Onyx. [env var: ONYX_PASSWORD] [default: None]
--version -v Show the client version number and exit.
--help -h Show this message and exit.

Commands
auth Authentication commands.
admin Admin commands.

Info
projects View available projects.
types View available field types.
lookups View available lookups.

Records
fields View the field specification for a project.
choices View options for a choice field in a project.
get Get a record from a project.
filter Filter multiple records from a project.
history View the history of a record in a project.
analyses View analyses of a record in a project.
identify Get the anonymised identifier for a value on a field.
create Create a record in a project.
update Update a record in a project.
delete Delete a record in a project.

Analyses
analysis-fields View the analysis field specification for a project.
analysis-choices View options for an analysis choice field.
get-analysis Get an analysis from a project.
filter-analysis Filter multiple analyses from a project.
analysis-history View the history of an analysis in a project.
analysis-records View records involved in an analysis in a project.
create-analysis Create an analysis in a project.
update-analysis Update an analysis in a project.
delete-analysis Delete an analysis in a project.

Accounts
profile View profile information.
activity View latest profile activity.
siteusers View users from the same site.
```

1.1.2 Contents

Installation

Learn how to install the client, or build it manually for development.

Accessibility

Learn how to enable/disable colours in the CLI.

Command-line Interface

Getting Started

Get started with filtering data on the command-line with Onyx.

Documentation

Documentation on all command-line functionality.

Python API

OnyxClient

Documentation on the `onyxClient` class, used for interacting with Onyx.

[OnyxConfig](#)

Documentation on the `OnyxConfig` class, used to provide credentials to `OnyxClient`.

[OnyxEnv](#)

Documentation on the `OnyxEnv` class, used as a shortcut to environment variable credentials.

[OnyxField](#)

Documentation on the `OnyxField` class, used to represent fields in an `OnyxClient` query.

[Exceptions](#)

Documentation on the possible exceptions raised by the `OnyxClient`.

1.2 Installation

Guidance for installing the Onyx client, or building it manually for development.

Usage within CLIMB JupyterLab Servers

If you are running a CLIMB JupyterLab server, you **do not** need to install the client, as it comes pre-configured in your environment.

If you cannot see the most up-to-date version of the Onyx client, this is because you will have previously installed your own version manually.

To revert your Onyx client to the managed up-to-date version, navigate to your terminal and run:

```
$ pip uninstall climb-onyx-client
```

And restart your JupyterLab server.

1.2.1 Install from conda-forge

```
$ conda create --name onyx --channel conda-forge climb-onyx-client
```

This installs the latest version of the client from [conda-forge](#).

1.2.2 Install from PyPI

```
$ pip install climb-onyx-client
```

This installs the latest version of the client from [PyPI](#).

1.2.3 Build from source

Clone the source code from GitHub:

```
$ git clone https://github.com/CLIMB-TRE/onyx-client.git
```

Run installation from within the source code directory:

```
$ cd onyx-client/  
$ pip install .
```

Developing the Client

If you wish to develop the client, ensure you have followed the above steps to build it.

From there, you can simply modify the client code and dependencies, and rebuild by executing:

```
$ pip install .
```

1.3 Accessibility

1.3.1 Enable/disable colours in the command-line interface

Colours are enabled by default in the output of the command-line interface:

```
Usage: onyx [OPTIONS] COMMAND [ARGS]...

API for Pathogen Metadata.
For documentation, see: https://climb-tre.github.io/onyx-client/

Options
--domain -d TEXT Domain name for connecting to Onyx. [env var: ONYX_DOMAIN] [default: None]
--token -t TEXT Token for authenticating with Onyx. [env var: ONYX_TOKEN] [default: None]
--username -u TEXT Username for authenticating with Onyx. [env var: ONYX_USERNAME] [default: None]
--password -p TEXT Password for authenticating with Onyx. [env var: ONYX_PASSWORD] [default: None]
--version -v Show the client version number and exit.
--help -h Show this message and exit.

Commands
auth Authentication commands.
admin Admin commands.

Info
projects View available projects.
types View available field types.
lookups View available lookups.

Records
fields View the field specification for a project.
choices View options for a choice field in a project.
get Get a record from a project.
filter Filter multiple records from a project.
history View the history of a record in a project.
analyses View analyses of a record in a project.
identify Get the anonymised identifier for a value on a field.
create Create a record in a project.
update Update a record in a project.
delete Delete a record in a project.

Analyses
analysis-fields View the analysis field specification for a project.
analysis-choices View options for an analysis choice field.
get-analysis Get an analysis from a project.
filter-analysis Filter multiple analyses from a project.
analysis-history View the history of an analysis in a project.
analysis-records View records involved in an analysis in a project.
create-analysis Create an analysis in a project.
update-analysis Update an analysis in a project.
delete-analysis Delete an analysis in a project.

Accounts
profile View profile information.
activity View latest profile activity.
siteusers View users from the same site.
```

To disable them, create an environment variable `ONYX_COLOURS` with the value `NONE`:

```
$ export ONYX_COLOURS=NONE
```

```
Usage: onyx [OPTIONS] COMMAND [ARGS]...

API for Pathogen Metadata.
For documentation, see: https://climb-tre.github.io/onyx-client/

Options
--domain -d TEXT Domain name for connecting to Onyx. [env var: ONYX_DOMAIN] [default: None]
--token -t TEXT Token for authenticating with Onyx. [env var: ONYX_TOKEN] [default: None]
--username -u TEXT Username for authenticating with Onyx. [env var: ONYX_USERNAME] [default: None]
--password -p TEXT Password for authenticating with Onyx. [env var: ONYX_PASSWORD] [default: None]
--version -v Show the client version number and exit.
--help -h Show this message and exit.

Commands
auth Authentication commands.
admin Admin commands.

Info
projects View available projects.
types View available field types.
lookups View available lookups.

Records
fields View the field specification for a project.
choices View options for a choice field in a project.
get Get a record from a project.
filter Filter multiple records from a project.
history View the history of a record in a project.
analyses View analyses of a record in a project.
identify Get the anonymised identifier for a value on a field.
create Create a record in a project.
update Update a record in a project.
delete Delete a record in a project.

Analyses
analysis-fields View the analysis field specification for a project.
analysis-choices View options for an analysis choice field.
get-analysis Get an analysis from a project.
filter-analysis Filter multiple analyses from a project.
analysis-history View the history of an analysis in a project.
analysis-records View records involved in an analysis in a project.
create-analysis Create an analysis in a project.
update-analysis Update an analysis in a project.
delete-analysis Delete an analysis in a project.

Accounts
profile View profile information.
activity View latest profile activity.
siteusers View users from the same site.
```

To re-enable colours, unset the environment variable:

```
$ unset ONYX_COLOURS
```

1.4 Command-line Interface

1.4.1 Getting Started

This guide walks through getting started with the Onyx-client command-line interface.

The guide assumes an environment where authentication credentials are pre-configured.

```
Usage: onyx [OPTIONS] COMMAND [ARGS]...

API for Pathogen Metadata.
For documentation, see: https://climb-tre.github.io/onyx-client/

Options
  --domain -d TEXT Domain name for connecting to Onyx. [env var: ONYX_DOMAIN] [default: None]
  --token -t TEXT Token for authenticating with Onyx. [env var: ONYX_TOKEN] [default: None]
  --username -u TEXT Username for authenticating with Onyx. [env var: ONYX_USERNAME] [default: None]
  --password -p TEXT Password for authenticating with Onyx. [env var: ONYX_PASSWORD] [default: None]
  --version -v Show the client version number and exit.
  --help -h Show this message and exit.

Commands
  auth Authentication commands.
  admin Admin commands.

Info
  projects View available projects.
  types View available field types.
  lookups View available lookups.

Records
  fields View the field specification for a project.
  choices View options for a choice field in a project.
  get Get a record from a project.
  filter Filter multiple records from a project.
  history View the history of a record in a project.
  analyses View analyses of a record in a project.
  identify Get the anonymised identifier for a value on a field.
  create Create a record in a project.
  update Update a record in a project.
  delete Delete a record in a project.

Analyses
  analysis-fields View the analysis field specification for a project.
  analysis-choices View options for an analysis choice field.
  get-analysis Get an analysis from a project.
  filter-analysis Filter multiple analyses from a project.
  analysis-history View the history of an analysis in a project.
  analysis-records View records involved in an analysis in a project.
  create-analysis Create an analysis in a project.
  update-analysis Update an analysis in a project.
  delete-analysis Delete an analysis in a project.

Accounts
  profile View profile information.
  activity View latest profile activity.
  siteusers View users from the same site.
```

Profile information

```
$ onyx profile
```

Available projects

```
$ onyx projects
```

If you cannot see the project(s) that you require access to, contact an admin.

Project fields

```
$ onyx fields PROJECT
```

This returns the fields specification for the given `PROJECT`.

Project data

```
$ onyx filter PROJECT
```

This returns all records from the given `PROJECT`.

The data can be exported to various file formats:

```
$ onyx filter PROJECT --format json > data.json
$ onyx filter PROJECT --format csv > data.csv
$ onyx filter PROJECT --format tsv > data.tsv
```

FILTERING

A project's data can be filtered to return records that match certain conditions.

Filtering on the CLI uses a `field=value` syntax, where `field` is the name of a field in the project, and `value` is the value you want to match.

Multiple filters can be provided, and only the records that satisfy *all* these filters will be returned.

ADVANCED FILTERING USING LOOKUPS

The data can be filtered in more complex ways using [lookups](#). These use a `field.lookup=value` syntax (or alternatively, `field__lookup=value`), and different ones are available depending on a field's [data type](#) (e.g. [text](#), [integer](#)). There are lookups for searching between a range of values on a field ([range](#)), whether a field's value is empty ([isnull](#)), whether a field case-insensitively contains some text ([icontains](#)), and [more](#).

EXAMPLES

To filter for all records in a `PROJECT` published on a specific date (e.g. `2023-09-18`):

```
$ onyx filter PROJECT --field published_date=2023-09-18
```

To filter for all records in a `PROJECT` published on the current date, a special `today` keyword can be used:

```
$ onyx filter PROJECT --field published_date=today
```

To filter for all records in a `PROJECT` with a `published_date` from `2023-09-01` to `2023-09-18`, the `range` lookup can be used:

```
$ onyx filter PROJECT --field published_date.range=2023-09-01,2023-09-18
```

Assuming that `PROJECT` has a `sample_type` field, then all records with `sample_type = "swab"` that were published from `2023-09-01` to `2023-09-18` can be obtained with:

```
$ onyx filter PROJECT --field published_date.range=2023-09-01,2023-09-18 --field sample_type=swab
```

Further guidance

For further guidance using Onyx-client, use the `--help` option.

```
$ onyx --help
$ onyx profile --help
$ onyx projects --help
$ onyx fields --help
$ onyx filter --help
```


1.4.2 onyx

API for Pathogen Metadata.

For documentation, see: <https://climb-tre.github.io/onyx-client/>

Usage:

```
$ onyx [OPTIONS] COMMAND [ARGS]...
```

Options:

- `-d, --domain TEXT` : Domain name for connecting to Onyx. [env var: ONYX_DOMAIN]
- `-t, --token TEXT` : Token for authenticating with Onyx. [env var: ONYX_TOKEN]
- `-u, --username TEXT` : Username for authenticating with Onyx. [env var: ONYX_USERNAME]
- `-p, --password TEXT` : Password for authenticating with Onyx. [env var: ONYX_PASSWORD]
- `-v, --version` : Show the client version number and exit.
- `--help` : Show this message and exit.

Commands:

- `projects` : View available projects.
- `types` : View available field types.
- `lookups` : View available lookups.
- `fields` : View the field specification for a project.
- `choices` : View options for a choice field in a project.
- `get` : Get a record from a project.
- `filter` : Filter multiple records from a project.
- `history` : View the history of a record in a project.
- `analyses` : View analyses of a record in a project.
- `identify` : Get the anonymised identifier for a value...
- `create` : Create a record in a project.
- `update` : Update a record in a project.
- `delete` : Delete a record in a project.
- `analysis-fields` : View the analysis field specification for...
- `analysis-choices` : View options for an analysis choice field.
- `get-analysis` : Get an analysis from a project.
- `filter-analysis` : Filter multiple analyses from a project.
- `analysis-history` : View the history of an analysis in a project.
- `analysis-records` : View records involved in an analysis in a...
- `create-analysis` : Create an analysis in a project.
- `update-analysis` : Update an analysis in a project.
- `delete-analysis` : Delete an analysis in a project.
- `profile` : View profile information.
- `activity` : View latest profile activity.
- `siteusers` : View users from the same site.
- `auth` : Authentication commands.
- `admin` : Admin commands.

onyx projects

View available projects.

Usage:

```
$ onyx projects [OPTIONS]
```

Options:

- `-F, --format [table|json]` : Set the file format of the returned data. [default: table]
- `--help` : Show this message and exit.

onyx types

View available field types.

Usage:

```
$ onyx types [OPTIONS]
```

Options:

- `-F, --format [table|json]` : Set the file format of the returned data. [default: table]
- `--help` : Show this message and exit.

onyx lookups

View available lookups.

Usage:

```
$ onyx lookups [OPTIONS]
```

Options:

- `-F, --format [table|json]` : Set the file format of the returned data. [default: table]
- `--help` : Show this message and exit.

onyx fields

View the field specification for a project.

Usage:

```
$ onyx fields [OPTIONS] PROJECT
```

Arguments:

- `PROJECT` : [required]

Options:

- `-F, --format [table|json|csv|tsv]` : Set the file format of the returned data. [default: table]
- `--help` : Show this message and exit.

onyx choices

View options for a choice field in a project.

Usage:

```
$ onyx choices [OPTIONS] PROJECT FIELD
```

Arguments:

- `PROJECT` : [required]
- `FIELD` : [required]

Options:

- `-F, --format [table|json]` : Set the file format of the returned data. [default: table]
- `--help` : Show this message and exit.

onyx get

Get a record from a project.

Usage:

```
$ onyx get [OPTIONS] PROJECT [CLIMB_ID]
```

Arguments:

- `PROJECT` : [required]
- `[CLIMB_ID]`

Options:

- `-f, --field TEXT` : Filter the data by providing conditions that the fields must match. Uses a `name=value` syntax.
- `-i, --include TEXT` : Specify which fields to include in the output.
- `-e, --exclude TEXT` : Specify which fields to exclude from the output.
- `--help` : Show this message and exit.

onyx filter

Filter multiple records from a project.

Usage:

```
$ onyx filter [OPTIONS] PROJECT
```

Arguments:

- `PROJECT` : [required]

Options:

- `-f, --field TEXT` : Filter the data by providing conditions that the fields must match. Uses a `name=value` syntax.
- `-i, --include TEXT` : Specify which fields to include in the output.
- `-e, --exclude TEXT` : Specify which fields to exclude from the output.
- `-s, --summarise TEXT` : For a given field (or group of fields), return the frequency of each unique value (or unique group of values).
- `-F, --format [json|csv|tsv]` : Set the file format of the returned data. [default: json]
- `--help` : Show this message and exit.

onyx history

View the history of a record in a project.

Usage:

```
$ onyx history [OPTIONS] PROJECT CLIMB_ID
```

Arguments:

- `PROJECT` : [required]
- `CLIMB_ID` : [required]

Options:

- `-F, --format [table|json]` : Set the file format of the returned data. [default: table]
- `--help` : Show this message and exit.

onyx analyses

View analyses of a record in a project.

Usage:

```
$ onyx analyses [OPTIONS] PROJECT CLIMB_ID
```

Arguments:

- `PROJECT` : [required]
- `CLIMB_ID` : [required]

Options:

- `-F, --format [json|csv|tsv]` : Set the file format of the returned data. [default: json]
- `--help` : Show this message and exit.

onyx identify

Get the anonymised identifier for a value on a field.

Usage:

```
$ onyx identify [OPTIONS] PROJECT FIELD VALUE
```

Arguments:

- `PROJECT` : [required]
- `FIELD` : [required]
- `VALUE` : [required]

Options:

- `-s, --site TEXT` : Site code for the value. If not provided, defaults to the user's site.
- `-F, --format [table|json]` : Set the file format of the returned data. [default: table]
- `--help` : Show this message and exit.

onyx create

Create a record in a project.

Usage:

```
$ onyx create [OPTIONS] PROJECT
```

Arguments:

- `PROJECT` : [required]

Options:

- `-f, --field TEXT` : Field and value to be created. Uses a `name=value` syntax.
- `-t, --test` : Run the command as a test. [default: (False)]
- `--help` : Show this message and exit.

onyx update

Update a record in a project.

Usage:

```
$ onyx update [OPTIONS] PROJECT CLIMB_ID
```

Arguments:

- `PROJECT` : [required]
- `CLIMB_ID` : [required]

Options:

- `-f, --field TEXT` : Field and value to be updated. Uses a `name=value` syntax.
- `-t, --test` : Run the command as a test. [default: (False)]
- `-c, --clear TEXT` : Specify fields to clear. Resets nested fields to empty.
- `--help` : Show this message and exit.

onyx delete

Delete a record in a project.

Usage:

```
$ onyx delete [OPTIONS] PROJECT CLIMB_ID
```

Arguments:

- `PROJECT` : [required]
- `CLIMB_ID` : [required]

Options:

- `--force` : Run the command without confirmation. [default: (False)]
- `--help` : Show this message and exit.

onyx analysis-fields

View the analysis field specification for a project.

Usage:

```
$ onyx analysis-fields [OPTIONS] PROJECT
```

Arguments:

- `PROJECT` : [required]

Options:

- `-F, --format [table|json|csv|tsv]` : Set the file format of the returned data. [default: table]
- `--help` : Show this message and exit.

onyx analysis-choices

View options for an analysis choice field.

Usage:

```
$ onyx analysis-choices [OPTIONS] PROJECT FIELD
```

Arguments:

- `PROJECT` : [required]
- `FIELD` : [required]

Options:

- `-F, --format [table|json]` : Set the file format of the returned data. [default: table]
- `--help` : Show this message and exit.

onyx get-analysis

Get an analysis from a project.

Usage:

```
$ onyx get-analysis [OPTIONS] PROJECT [ANALYSIS_ID]
```

Arguments:

- `PROJECT` : [required]
- `[ANALYSIS_ID]`

Options:

- `-f, --field TEXT` : Filter the data by providing conditions that the fields must match. Uses a `name=value` syntax.
- `-i, --include TEXT` : Specify which fields to include in the output.
- `-e, --exclude TEXT` : Specify which fields to exclude from the output.
- `--help` : Show this message and exit.

onyx filter-analysis

Filter multiple analyses from a project.

Usage:

```
$ onyx filter-analysis [OPTIONS] PROJECT
```

Arguments:

- `PROJECT` : [required]

Options:

- `-f, --field TEXT` : Filter the data by providing conditions that the fields must match. Uses a `name=value` syntax.
- `-i, --include TEXT` : Specify which fields to include in the output.
- `-e, --exclude TEXT` : Specify which fields to exclude from the output.
- `-s, --summarise TEXT` : For a given field (or group of fields), return the frequency of each unique value (or unique group of values).
- `-F, --format [json|csv|tsv]` : Set the file format of the returned data. [default: json]
- `--help` : Show this message and exit.

onyx analysis-history

View the history of an analysis in a project.

Usage:

```
$ onyx analysis-history [OPTIONS] PROJECT ANALYSIS_ID
```

Arguments:

- `PROJECT` : [required]
- `ANALYSIS_ID` : [required]

Options:

- `-F, --format [table|json]` : Set the file format of the returned data. [default: table]
- `--help` : Show this message and exit.

onyx analysis-records

View records involved in an analysis in a project.

Usage:

```
$ onyx analysis-records [OPTIONS] PROJECT ANALYSIS_ID
```

Arguments:

- `PROJECT` : [required]
- `ANALYSIS_ID` : [required]

Options:

- `-F, --format [json|csv|tsv]` : Set the file format of the returned data. [default: json]
- `--help` : Show this message and exit.

onyx create-analysis

Create an analysis in a project.

Usage:

```
$ onyx create-analysis [OPTIONS] PROJECT
```

Arguments:

- `PROJECT` : [required]

Options:

- `-f, --field TEXT` : Field and value to be created. Uses a `name=value` syntax.
- `-t, --test` : Run the command as a test. [default: (False)]
- `--help` : Show this message and exit.

onyx update-analysis

Update an analysis in a project.

Usage:

```
$ onyx update-analysis [OPTIONS] PROJECT ANALYSIS_ID
```

Arguments:

- `PROJECT` : [required]
- `ANALYSIS_ID` : [required]

Options:

- `-f, --field TEXT` : Field and value to be updated. Uses a `name=value` syntax.
- `-t, --test` : Run the command as a test. [default: (False)]
- `-c, --clear TEXT` : Specify fields to clear. Resets nested fields to empty.
- `--help` : Show this message and exit.

onyx delete-analysis

Delete an analysis in a project.

Usage:

```
$ onyx delete-analysis [OPTIONS] PROJECT ANALYSIS_ID
```

Arguments:

- `PROJECT` : [required]
- `ANALYSIS_ID` : [required]

Options:

- `--force` : Run the command without confirmation. [default: (False)]
- `--help` : Show this message and exit.

onyx profile

View profile information.

Usage:

```
$ onyx profile [OPTIONS]
```

Options:

- `-F, --format [table|json]` : Set the file format of the returned data. [default: table]
- `--help` : Show this message and exit.

onyx activity

View latest profile activity.

Usage:

```
$ onyx activity [OPTIONS]
```

Options:

- `-F, --format [table|json]` : Set the file format of the returned data. [default: table]
- `--help` : Show this message and exit.

onyx siteusers

View users from the same site.

Usage:

```
$ onyx siteusers [OPTIONS]
```

Options:

- `-F, --format [table|json]` : Set the file format of the returned data. [default: table]
- `--help` : Show this message and exit.

onyx auth

Authentication commands.

Usage:

```
$ onyx auth [OPTIONS] COMMAND [ARGS]...
```

Options:

- `--help` : Show this message and exit.

Commands:

- `register` : Create a new user.
- `login` : Log in.
- `logout` : Log out.
- `logoutall` : Log out across all clients.

onyx auth register

Create a new user.

Usage:

```
$ onyx auth register [OPTIONS]
```

Options:

- `--help` : Show this message and exit.

onyx auth login

Log in.

Usage:

```
$ onyx auth login [OPTIONS]
```

Options:

- `--help`: Show this message and exit.

```
onyx auth logout
```

Log out.

Usage:

```
$ onyx auth logout [OPTIONS]
```

Options:

- `--help`: Show this message and exit.

```
onyx auth logoutall
```

Log out across all clients.

Usage:

```
$ onyx auth logoutall [OPTIONS]
```

Options:

- `--help`: Show this message and exit.

```
onyx admin
```

Admin commands.

Usage:

```
$ onyx admin [OPTIONS] COMMAND [ARGS]...
```

Options:

- `--help`: Show this message and exit.

Commands:

- `waiting`: View users waiting for approval.
- `approve`: Approve a user.
- `allusers`: View users across all sites.

```
onyx admin waiting
```

View users waiting for approval.

Usage:

```
$ onyx admin waiting [OPTIONS]
```

Options:

- `-F, --format [table|json]`: Set the file format of the returned data. [default: table]
- `--help`: Show this message and exit.

```
onyx admin approve
```

Approve a user.

Usage:

```
$ onyx admin approve [OPTIONS] USERNAME
```

Arguments:

- `USERNAME` : Name of the user being approved. [required]

Options:

- `--help` : Show this message and exit.

```
onyx admin allusers
```

View users across all sites.

Usage:

```
$ onyx admin allusers [OPTIONS]
```

Options:

- `-F, --format [table|json]` : Set the file format of the returned data. [default: table]
- `--help` : Show this message and exit.

1.5 Python API

1.5.1 OnyxClient

Class for querying and manipulating data within Onyx.

`__init__(config)`

Initialise a client.

PARAMETER	DESCRIPTION
<code>config</code>	<code>OnyxConfig</code> object that stores information for connecting and authenticating with Onyx. TYPE: <code>OnyxConfig</code>

Examples:

The recommended way to initialise a client (as a context manager):

```
import os
from onyx import OnyxConfig, OnyxEnv, OnyxClient

config = OnyxConfig(
    domain=os.environ[OnyxEnv.DOMAIN],
    token=os.environ[OnyxEnv.TOKEN],
)

with OnyxClient(config) as client:
    pass # Do something with the client here
```

Alternatively, the client can be initialised as follows:

```
import os
from onyx import OnyxConfig, OnyxEnv, OnyxClient

config = OnyxConfig(
    domain=os.environ[OnyxEnv.DOMAIN],
    token=os.environ[OnyxEnv.TOKEN],
)

client = OnyxClient(config)
# Do something with the client here
```

os ▾

- When making multiple requests, using the client as a context manager can improve performance.
- This is due to the fact that the client will re-use the same session for all requests, rather than creating a new session for each request.
- For more information, see: <https://requests.readthedocs.io/en/master/user/advanced/#session-objects>

`projects()`

View available projects.

RETURNS	DESCRIPTION
<code>List[Dict[str, str]]</code>	List of projects.

Examples:

```
import os
from onyx import OnyxConfig, OnyxEnv, OnyxClient
```

```

config = OnyxConfig(
    domain=os.environ[OnyxEnv.DOMAIN],
    token=os.environ[OnyxEnv.TOKEN],
)

with OnyxClient(config) as client:
    projects = client.projects()

```

```

>>> projects
[
  {
    "project": "project_1",
    "scope": "admin",
    "actions": [
      "get",
      "list",
      "filter",
      "add",
      "change",
      "delete",
    ],
  },
  {
    "project": "project_2",
    "scope": "analyst",
    "actions": [
      "get",
      "list",
      "filter",
    ],
  },
]

```

types()

View available field types.

RETURNS	DESCRIPTION
List[Dict[str, Any]]	List of field types.

Examples:

```

import os
from onyx import OnyxConfig, OnyxEnv, OnyxClient

config = OnyxConfig(
    domain=os.environ[OnyxEnv.DOMAIN],
    token=os.environ[OnyxEnv.TOKEN],
)

with OnyxClient(config) as client:
    field_types = client.types()

```

```

>>> field_types
[
  {
    "type": "text",
    "description": "A string of characters.",
    "lookups": [
      "exact",
      "ne",
      "in",
      "notin",
      "contains",
      "startswith",
      "endswith",
      "iexact",
      "icontains",
      "istartswith",
      "iendswith",
      "length",
      "length__in",
      "length__range",
      "isnull",
    ],
  },
  {
    "type": "choice",
    "description": "A restricted set of options.",
    "lookups": [
      "exact",
      "ne",
      "in",
      "notin",
      "isnull",
    ],
  },
]

```

```
    },
  },
]
```

lookups()

View available lookups.

RETURNS	DESCRIPTION
List[Dict[str, Any]]	List of lookups.

Examples:

```
import os

from onyx import OnyxConfig, OnyxEnv, OnyxClient

config = OnyxConfig(
    domain=os.environ[OnyxEnv.DOMAIN],
    token=os.environ[OnyxEnv.TOKEN],
)

with OnyxClient(config) as client:
    lookups = client.lookups()
```

```
>>> lookups
[
  {
    "lookup": "exact",
    "description": "The field's value must be equal to the query value.",
    "types": [
      "text",
      "choice",
      "integer",
      "decimal",
      "date",
      "datetime",
      "bool",
    ],
  },
  {
    "lookup": "ne",
    "description": "The field's value must not be equal to the query value.",
    "types": [
      "text",
      "choice",
      "integer",
      "decimal",
      "date",
      "datetime",
      "bool",
    ],
  },
]
```

fields(project)

View fields for a project.

PARAMETER	DESCRIPTION
project	Name of the project.
TYPE: str	

RETURNS	DESCRIPTION
Dict[str, Any]	Dict of fields.

Examples:

```
import os
from onyx import OnyxConfig, OnyxEnv, OnyxClient

config = OnyxConfig(
    domain=os.environ[OnyxEnv.DOMAIN],
    token=os.environ[OnyxEnv.TOKEN],
```

```
)

with OnyxClient(config) as client:
    fields = client.fields("project")
```

```
>>> fields
{
  "name": "Project Name",
  "description": "Project description.",
  "object_type": "records",
  "version": "0.1.0",
  "fields": {
    "climb_id": {
      "description": "Unique identifier for a project record in Onyx.",
      "type": "text",
      "required": True,
      "actions": [
        "get",
        "list",
        "filter",
      ],
      "restrictions": [
        "Max length: 12",
      ],
    },
    "is_published": {
      "description": "Indicator for whether a project record has been published.",
      "type": "bool",
      "required": False,
      "actions": [
        "get",
        "list",
        "filter",
        "add",
        "change",
      ],
      "default": True,
    },
    "published_date": {
      "description": "The date the project record was published in Onyx.",
      "type": "date (YYYY-MM-DD)",
      "required": False,
      "actions": [
        "get",
        "list",
        "filter",
      ],
    },
    "country": {
      "description": "Country of origin.",
      "type": "choice",
      "required": False,
      "actions": [
        "get",
        "list",
        "filter",
        "add",
        "change",
      ],
      "values": [
        "ENG",
        "WALES",
        "SCOT",
        "NI",
      ],
    },
  },
}
```

```
choices(project, field)
```

View choices for a field.

PARAMETER	DESCRIPTION
<code>project</code>	Name of the project. TYPE: <code>str</code>
<code>field</code>	Choice field on the project. TYPE: <code>str</code>

RETURNS	DESCRIPTION
<code>Dict[str, Dict[str, Any]]</code>	Dictionary mapping choices to information about the choice.

Examples:

```
import os
from onyx import OnyxConfig, OnyxEnv, OnyxClient

config = OnyxConfig(
    domain=os.environ[OnyxEnv.DOMAIN],
    token=os.environ[OnyxEnv.TOKEN],
)

with OnyxClient(config) as client:
    choices = client.choices("project", "country")
```

```
>>> choices
{
  "ENG": {
    "description": "England",
    "is_active" : True,
  },
  "WALES": {
    "description": "Wales",
    "is_active" : True,
  },
  "SCOT": {
    "description": "Scotland",
    "is_active" : True,
  },
  "NI": {
    "description": "Northern Ireland",
    "is_active" : True,
  },
}
```



```
get(project, climb_id=None, fields=None, include=None, exclude=None)
```

Get a record from a project.

PARAMETER	DESCRIPTION
<code>project</code>	Name of the project. TYPE: <code>str</code>
<code>climb_id</code>	Unique identifier for the record. TYPE: <code>Optional[str]</code> DEFAULT: <code>None</code>
<code>fields</code>	Dictionary of field filters used to uniquely identify the record. TYPE: <code>Optional[Dict[str, Any]]</code> DEFAULT: <code>None</code>
<code>include</code>	Fields to include in the output. TYPE: <code>Union[List[str], str, None]</code> DEFAULT: <code>None</code>
<code>exclude</code>	Fields to exclude from the output. TYPE: <code>Union[List[str], str, None]</code> DEFAULT: <code>None</code>

RETURNS	DESCRIPTION
<code>Dict[str, Any]</code>	Dict containing the record.

Examples:

Get a record by CLIMB ID:

```
import os
from onyx import OnyxConfig, OnyxEnv, OnyxClient

config = OnyxConfig(
    domain=os.environ[OnyxEnv.DOMAIN],
    token=os.environ[OnyxEnv.TOKEN],
)

with OnyxClient(config) as client:
    record = client.get("project", "C-1234567890")
```

```
>>> record
{
  "climb_id": "C-1234567890",
  "published_date": "2023-01-01",
  "field1": "value1",
  "field2": "value2",
}
```

Get a record by fields that uniquely identify it:

```
import os
from onyx import OnyxConfig, OnyxEnv, OnyxClient

config = OnyxConfig(
    domain=os.environ[OnyxEnv.DOMAIN],
    token=os.environ[OnyxEnv.TOKEN],
)

with OnyxClient(config) as client:
    record = client.get(
        "project",
        fields={
            "field1": "value1",
            "field2": "value2",
        },
    )
```

```
>>> record
{
  "climb_id": "C-1234567890",
  "published_date": "2023-01-01",
}
```

```

    "field1": "value1",
    "field2": "value2",
}

```

The `include` and `exclude` arguments can be used to control the fields returned:

```

import os
from onyx import OnyxConfig, OnyxEnv, OnyxClient

config = OnyxConfig(
    domain=os.environ[OnyxEnv.DOMAIN],
    token=os.environ[OnyxEnv.TOKEN],
)

with OnyxClient(config) as client:
    record_v1 = client.get(
        "project",
        climb_id="C-1234567890",
        include=["climb_id", "published_date"],
    )
    record_v2 = client.get(
        "project",
        climb_id="C-1234567890",
        exclude=["field2"],
    )

```

```

>>> record_v1
{
  "climb_id": "C-1234567890",
  "published_date": "2023-01-01",
}
>>> record_v2
{
  "climb_id": "C-1234567890",
  "published_date": "2023-01-01",
  "field1": "value1",
}

```

OS ▼

- Including/excluding fields to reduce the size of the returned data can improve performance.

```
filter(project, fields=None, include=None, exclude=None, summarise=None, **kwargs)
```

Filter records from a project.

PARAMETER	DESCRIPTION
<code>project</code>	Name of the project. TYPE: <code>str</code>
<code>fields</code>	Dictionary of field filters. TYPE: <code>Optional[Dict[str, Any]]</code> DEFAULT: <code>None</code>
<code>include</code>	Fields to include in the output. TYPE: <code>Union[List[str], str, None]</code> DEFAULT: <code>None</code>
<code>exclude</code>	Fields to exclude from the output. TYPE: <code>Union[List[str], str, None]</code> DEFAULT: <code>None</code>
<code>summarise</code>	For a given field (or group of fields), return the frequency of each unique value (or unique group of values). TYPE: <code>Union[List[str], str, None]</code> DEFAULT: <code>None</code>
<code>**kwargs</code>	Additional keyword arguments are interpreted as field filters. TYPE: <code>Any</code> DEFAULT: <code>{}</code>

RETURNS	DESCRIPTION
<code>None</code>	Generator of records. If a <code>summarise</code> argument is provided, each record will be a dict containing values of the summary fields and a count for the frequency.

tes ▾

- Field filters specify requirements that the returned data must satisfy. They can be provided as keyword arguments, or as a dictionary to the `fields` argument.
- These filters can be a simple match on a value (e.g. `"published_date" : "2023-01-01"`), or they can use a 'lookup' for more complex matching conditions (e.g. `"published_date__iso_year" : "2023"`).
- Multi-value lookups (e.g. `in`, `range`) can also be used. For keyword arguments, multiple values can be provided as a Python list. For the `fields` dictionary, multiple values must be provided as a comma-separated string (see examples below).

Examples:

Retrieve all records that match a set of field requirements:

```
import os
from onyx import OnyxConfig, OnyxEnv, OnyxClient

config = OnyxConfig(
    domain=os.environ[OnyxEnv.DOMAIN],
    token=os.environ[OnyxEnv.TOKEN],
)

# Field conditions can either be provided as keyword arguments:
with OnyxClient(config) as client:
    records = list(
        client.filter(
            project="project",
            field1="abcd",
            published_date__range=["2023-01-01", "2023-01-02"],
        )
    )

# Or as a dictionary to the 'fields' argument:
with OnyxClient(config) as client:
    records = list(
```

```

        client.filter(
            project="project",
            fields={
                "field1": "abcd",
                "published_date__range" : "2023-01-01, 2023-01-02",
            },
        )
    )
)

```

```

>>> records
[
  {
    "climb_id": "C-1234567890",
    "published_date": "2023-01-01",
    "field1": "abcd",
    "field2": 123,
  },
  {
    "climb_id": "C-1234567891",
    "published_date": "2023-01-02",
    "field1": "abcd",
    "field2": 456,
  },
]

```

The `summarise` argument can be used to return the frequency of each unique value for a given field, or the frequency of each unique set of values for a group of fields:

```

import os
from onyx import OnyxConfig, OnyxEnv, OnyxClient

config = OnyxConfig(
    domain=os.environ[OnyxEnv.DOMAIN],
    token=os.environ[OnyxEnv.TOKEN],
)

with OnyxClient(config) as client:
    records_v1 = list(
        client.filter(
            project="project",
            field1="abcd",
            published_date__range=["2023-01-01", "2023-01-02"],
            summarise="published_date",
        )
    )

    records_v2 = list(
        client.filter(
            project="project",
            field1="abcd",
            published_date__range=["2023-01-01", "2023-01-02"],
            summarise=["published_date", "field2"],
        )
    )

```

```

>>> records_v1
[
  {
    "published_date": "2023-01-01",
    "count": 1,
  },
  {
    "published_date": "2023-01-02",
    "count": 1,
  },
]
>>> records_v2
[
  {
    "published_date": "2023-01-01",
    "field2": 123,
    "count": 1,
  },
  {
    "published_date": "2023-01-02",
    "field2": 456,
    "count": 1,
  },
]

```

```
query(project, query=None, include=None, exclude=None, summarise=None)
```

Query records from a project.

This method supports more complex filtering than the `OnyxClient.filter` method. Here, filters can be combined using Python's bitwise operators, representing AND, OR, XOR and NOT operations.

PARAMETER	DESCRIPTION
<code>project</code>	Name of the project. TYPE: <code>str</code>
<code>query</code>	<code>OnyxField</code> object representing the query being made. TYPE: <code>Optional[OnyxField]</code> DEFAULT: <code>None</code>
<code>include</code>	Fields to include in the output. TYPE: <code>Union[List[str], str, None]</code> DEFAULT: <code>None</code>
<code>exclude</code>	Fields to exclude from the output. TYPE: <code>Union[List[str], str, None]</code> DEFAULT: <code>None</code>
<code>summarise</code>	For a given field (or group of fields), return the frequency of each unique value (or unique group of values). TYPE: <code>Union[List[str], str, None]</code> DEFAULT: <code>None</code>

RETURNS	DESCRIPTION
<code>None</code>	Generator of records. If a <code>summarise</code> argument is provided, each record will be a dict containing values of the summary fields and a count for the frequency.

Notes ▾

- The `query` argument must be an instance of `OnyxField`.
- `OnyxField` instances can be combined into complex expressions using Python's bitwise operators: `&` (AND), `|` (OR), `^` (XOR), and `~` (NOT).
- Multi-value lookups (e.g. `in`, `range`) support passing a Python list (see example below).

Examples:

Retrieve all records that match the query provided by an `OnyxField` object:

```
import os
from onyx import OnyxConfig, OnyxEnv, OnyxClient, OnyxField

config = OnyxConfig(
    domain=os.environ[OnyxEnv.DOMAIN],
    token=os.environ[OnyxEnv.TOKEN],
)

with OnyxClient(config) as client:
    records = list(
        client.query(
            project="project",
            query=(
                OnyxField(field1="abcd")
                & OnyxField(published_date__range=["2023-01-01", "2023-01-02"])
            ),
        )
    )
```

```
>>> records
[
  {
    "climb_id": "C-1234567890",
    "published_date": "2023-01-01",
    "field1": "abcd",
    "field2": 123,
  },
  {
    "climb_id": "C-1234567891",
```

```

        "published_date": "2023-01-02",
        "field1": "abcd",
        "field2": 456,
    },
]

```

`to_csv(csv_file, data, delimiter=None)` classmethod

Write a set of records to a CSV file.

PARAMETER	DESCRIPTION
<code>csv_file</code>	File object for the CSV file being written to. TYPE: <code>TextIO</code>
<code>data</code>	The data being written to the CSV file. Must be either a list / generator of dict records. TYPE: <code>Union[List[Dict[str, Any]], Generator[Dict[str, Any], Any, None]]</code>
<code>delimiter</code>	CSV delimiter. If not provided, defaults to <code>,</code> for CSVs. Set this to <code>"\t"</code> to work with TSV files. TYPE: <code>Optional[str]</code> DEFAULT: <code>None</code>

Examples:

```

import os
from onyx import OnyxConfig, OnyxEnv, OnyxClient

config = OnyxConfig(
    domain=os.environ[OnyxEnv.DOMAIN],
    token=os.environ[OnyxEnv.TOKEN],
)

with OnyxClient(config) as client, open("/path/to/file.csv") as csv_file:
    client.to_csv(
        csv_file=csv_file,
        data=client.filter(
            "project",
            fields={
                "field1": "value1",
                "field2": "value2",
            },
        ),
    )

```

`history(project, climb_id)`

View the history of a record in a project.

PARAMETER	DESCRIPTION
<code>project</code>	Name of the project. TYPE: <code>str</code>
<code>climb_id</code>	Unique identifier for the record. TYPE: <code>str</code>

RETURNS	DESCRIPTION
<code>Dict[str, Any]</code>	Dict containing the history of the record.

Examples:

```

import os
from onyx import OnyxConfig, OnyxEnv, OnyxClient

config = OnyxConfig(
    domain=os.environ[OnyxEnv.DOMAIN],
    token=os.environ[OnyxEnv.TOKEN],
)

```

```
with OnyxClient(config) as client:
    history = client.history("project", "C-1234567890")

>>> history
{
  "climb_id": "C-1234567890",
  "history": [
    {
      "username": "user",
      "timestamp": "2023-01-01T00:00:00Z",
      "action": "add",
    },
    {
      "username": "user",
      "timestamp": "2023-01-02T00:00:00Z",
      "action": "change",
      "changes": [
        {
          "field": "field_1",
          "type": "text",
          "from": "value1",
          "to": "value2",
        },
        {
          "field": "field_2",
          "type": "integer",
          "from": 3,
          "to": 4,
        },
        {
          "field": "nested_field",
          "type": "relation",
          "action": "add",
          "count": 3,
        },
        {
          "field": "nested_field",
          "type": "relation",
          "action": "change",
          "count": 10,
        },
      ],
    },
  ],
}
```

`analyses(project, climb_id)`

View the analyses of a record in a project.

PARAMETER	DESCRIPTION
<code>project</code>	Name of the project. TYPE: <code>str</code>
<code>climb_id</code>	Unique identifier for the record. TYPE: <code>str</code>

RETURNS	DESCRIPTION
<code>List[Dict[str, Any]]</code>	List of Dicts containing basic details of each analysis of the record.

Examples:

```
import os
from onyx import OnyxConfig, OnyxEnv, OnyxClient

config = OnyxConfig(
    domain=os.environ[OnyxEnv.DOMAIN],
    token=os.environ[OnyxEnv.TOKEN],
)

with OnyxClient(config) as client:
    analyses = client.analyses("project", "C-1234567890")

>>> analyses
[
  {
    "analysis_id": "A-1234567890",
    "published_date": "2023-02-01",
```

```
        "analysis_date": "2023-01-01",
        "site": "site",
        "name": "First Analysis",
        "report": "s3://analysis_1.html",
        "outputs": "s3://analysis_1_outputs/",
    },
    {
        "analysis_id": "A-0987654321",
        "published_date": "2024-02-01",
        "analysis_date": "2023-01-01",
        "site": "site",
        "name": "Second Analysis",
        "report": "s3://analysis_2.html",
        "outputs": "s3://analysis_2_outputs/",
    },
]
```

`identify(project, field, value, site=None)`

Get the anonymised identifier for a value on a field.

PARAMETER	DESCRIPTION
<code>project</code>	Name of the project. TYPE: <code>str</code>
<code>field</code>	Field on the project. TYPE: <code>str</code>
<code>value</code>	Value to identify. TYPE: <code>str</code>
<code>site</code>	Site to identify the value on. If not provided, defaults to the user's site. TYPE: <code>Optional[str]</code> DEFAULT: <code>None</code>

RETURNS	DESCRIPTION
<code>Dict[str, str]</code>	Dict containing the project, site, field, value and anonymised identifier.

Examples:

```
import os
from onyx import OnyxConfig, OnyxEnv, OnyxClient

config = OnyxConfig(
    domain=os.environ[OnyxEnv.DOMAIN],
    token=os.environ[OnyxEnv.TOKEN],
)

with OnyxClient(config) as client:
    identification = client.identify("project", "sample_id", "hidden-value")

>>> identification
{
    "project": "project",
    "site": "site",
    "field": "sample_id",
    "value": "hidden-value",
    "identifier": "S-1234567890",
}
```



```
create(project, fields, test=False)
```

Create a record in a project.

PARAMETER	DESCRIPTION
project	Name of the project. TYPE: str
fields	Object representing the record to be created. TYPE: Dict[str, Any]
test	If True, runs the command as a test. Default: False TYPE: bool DEFAULT: False

RETURNS	DESCRIPTION
Dict[str, Any]	Dict containing the CLIMB ID of the created record.

Examples:

```
import os
from onyx import OnyxConfig, OnyxEnv, OnyxClient

config = OnyxConfig(
    domain=os.environ[OnyxEnv.DOMAIN],
    token=os.environ[OnyxEnv.TOKEN],
)

with OnyxClient(config) as client:
    result = client.create(
        "project",
        fields={
            "field1": "value1",
            "field2": "value2",
        },
    )

>>> result
{"climb_id": "C-1234567890"}
```

```
update(project, climb_id, fields=None, test=False, clear=None)
```

Update a record in a project.

PARAMETER	DESCRIPTION
<code>project</code>	Name of the project. TYPE: <code>str</code>
<code>climb_id</code>	Unique identifier for the record. TYPE: <code>str</code>
<code>fields</code>	Object representing the record to be updated. TYPE: <code>Optional[Dict[str, Any]]</code> DEFAULT: <code>None</code>
<code>test</code>	If <code>True</code> , runs the command as a test. Default: <code>False</code> TYPE: <code>bool</code> DEFAULT: <code>False</code>
<code>clear</code>	Field or list of fields to clear. Resets nested fields to empty. TYPE: <code>Union[List[str], str, None]</code> DEFAULT: <code>None</code>

RETURNS	DESCRIPTION
<code>Dict[str, Any]</code>	Dict containing the CLIMB ID of the updated record.

Examples:

```
import os
from onyx import OnyxConfig, OnyxEnv, OnyxClient

config = OnyxConfig(
    domain=os.environ[OnyxEnv.DOMAIN],
    token=os.environ[OnyxEnv.TOKEN],
)

with OnyxClient(config) as client:
    result = client.update(
        project="project",
        climb_id="C-1234567890",
        fields={
            "field1": "value1",
            "field2": "value2",
        },
        clear=["field3", "field4"],
    )
```

```
>>> result
{"climb_id": "C-1234567890"}
```

```
delete(project, climb_id)
```

Delete a record in a project.

PARAMETER	DESCRIPTION
<code>project</code>	Name of the project. TYPE: <code>str</code>
<code>climb_id</code>	Unique identifier for the record. TYPE: <code>str</code>

RETURNS	DESCRIPTION
<code>Dict[str, Any]</code>	Dict containing the CLIMB ID of the deleted record.

Examples:

```
import os
from onyx import OnyxConfig, OnyxEnv, OnyxClient

config = OnyxConfig(
    domain=os.environ[OnyxEnv.DOMAIN],
    token=os.environ[OnyxEnv.TOKEN],
)

with OnyxClient(config) as client:
    result = client.delete(
        project="project",
        climb_id="C-1234567890",
    )
```

```
>>> result
{"climb_id": "C-1234567890"}
```

csv_create(project, csv_file, fields=None, delimiter=None, multiline=False, test=False)

Use a CSV file to create record(s) in a project.

PARAMETER	DESCRIPTION
project	Name of the project. TYPE: str
csv_file	File object for the CSV file being used for record upload. TYPE: TextIO
fields	Additional fields provided for each record being uploaded. Takes precedence over fields in the CSV. TYPE: Optional[Dict[str, Any]] DEFAULT: None
delimiter	CSV delimiter. If not provided, defaults to "," for CSVs. Set this to "\t" to work with TSV files. TYPE: Optional[str] DEFAULT: None
multiline	If True, allows processing of CSV files with more than one record. Default: False TYPE: bool DEFAULT: False
test	If True, runs the command as a test. Default: False TYPE: bool DEFAULT: False

RETURNS	DESCRIPTION
Union[Dict[str, Any], List[Dict[str, Any]]]	Dict containing the CLIMB ID of the created record. If multiline = True, returns a list of dicts containing the CLIMB ID of each created record.

Examples:

Create a single record:

```
import os
from onyx import OnyxConfig, OnyxEnv, OnyxClient

config = OnyxConfig(
    domain=os.environ[OnyxEnv.DOMAIN],
    token=os.environ[OnyxEnv.TOKEN],
)

with OnyxClient(config) as client, open("/path/to/file.csv") as csv_file:
    result = client.csv_create(
        project="project",
        csv_file=csv_file,
    )
```

```
>>> result
{"climb_id": "C-1234567890"}
```

Create multiple records:

```
import os
from onyx import OnyxConfig, OnyxEnv, OnyxClient

config = OnyxConfig(
    domain=os.environ[OnyxEnv.DOMAIN],
    token=os.environ[OnyxEnv.TOKEN],
)

with OnyxClient(config) as client, open("/path/to/file.csv") as csv_file:
    results = client.csv_create(
        project="project",
        csv_file=csv_file,
        multiline=True,
    )
```

```
>>> results
[
  {"climb_id": "C-1234567890"},
  {"climb_id": "C-1234567891"},
  {"climb_id": "C-1234567892"},
]
```

```
csv_update(project, csv_file, fields=None, delimiter=None, multiline=False, test=False)
```

Use a CSV file to update record(s) in a project.

PARAMETER	DESCRIPTION
<code>project</code>	Name of the project. TYPE: <code>str</code>
<code>csv_file</code>	File object for the CSV file being used for record upload. TYPE: <code>TextIO</code>
<code>fields</code>	Additional fields provided for each record being uploaded. Takes precedence over fields in the CSV. TYPE: <code>Optional[Dict[str, Any]]</code> DEFAULT: <code>None</code>
<code>delimiter</code>	CSV delimiter. If not provided, defaults to <code>","</code> for CSVs. Set this to <code>"\t"</code> to work with TSV files. TYPE: <code>Optional[str]</code> DEFAULT: <code>None</code>
<code>multiline</code>	If <code>True</code> , allows processing of CSV files with more than one record. Default: <code>False</code> TYPE: <code>bool</code> DEFAULT: <code>False</code>
<code>test</code>	If <code>True</code> , runs the command as a test. Default: <code>False</code> TYPE: <code>bool</code> DEFAULT: <code>False</code>

RETURNS	DESCRIPTION
<code>Union[Dict[str, Any], List[Dict[str, Any]]]</code>	Dict containing the CLIMB ID of the updated record. If <code>multiline = True</code> , returns a list of dicts containing the CLIMB ID of each updated record.

Examples:

Update a single record:

```
import os
from onyx import OnyxConfig, OnyxEnv, OnyxClient

config = OnyxConfig(
    domain=os.environ[OnyxEnv.DOMAIN],
    token=os.environ[OnyxEnv.TOKEN],
)

with OnyxClient(config) as client, open("/path/to/file.csv") as csv_file:
    result = client.csv_update(
        project="project",
```

```
    csv_file=csv_file,
)
```

```
>>> result
{"climb_id": "C-1234567890"}
```

Update multiple records:

```
import os
from onyx import OnyxConfig, OnyxEnv, OnyxClient

config = OnyxConfig(
    domain=os.environ[OnyxEnv.DOMAIN],
    token=os.environ[OnyxEnv.TOKEN],
)

with OnyxClient(config) as client, open("/path/to/file.csv") as csv_file:
    results = client.csv_update(
        project="project",
        csv_file=csv_file,
        multiline=True,
    )
```

```
>>> results
[
    {"climb_id": "C-1234567890"},
    {"climb_id": "C-1234567891"},
    {"climb_id": "C-1234567892"},
]
```

```
csv_delete(project, csv_file, delimiter=None, multiline=False)
```

Use a CSV file to delete record(s) in a project.

PARAMETER	DESCRIPTION
<code>project</code>	Name of the project. TYPE: <code>str</code>
<code>csv_file</code>	File object for the CSV file being used for record upload. TYPE: <code>TextIO</code>
<code>delimiter</code>	CSV delimiter. If not provided, defaults to <code>,</code> for CSVs. Set this to <code>"\t"</code> to work with TSV files. TYPE: <code>Optional[str]</code> DEFAULT: <code>None</code>
<code>multiline</code>	If <code>True</code> , allows processing of CSV files with more than one record. Default: <code>False</code> TYPE: <code>bool</code> DEFAULT: <code>False</code>

RETURNS	DESCRIPTION
<code>Union[Dict[str, Any], List[Dict[str, Any]]]</code>	Dict containing the CLIMB ID of the deleted record. If <code>multiline = True</code> , returns a list of dicts containing the CLIMB ID of each deleted record.

Examples:

Delete a single record:

```
import os
from onyx import OnyxConfig, OnyxEnv, OnyxClient

config = OnyxConfig(
    domain=os.environ[OnyxEnv.DOMAIN],
    token=os.environ[OnyxEnv.TOKEN],
)

with OnyxClient(config) as client, open("/path/to/file.csv") as csv_file:
    result = client.csv_delete(
        project="project",
        csv_file=csv_file,
    )
```

```
>>> result
{"climb_id": "C-1234567890"}
```

Delete multiple records:

```
import os
from onyx import OnyxConfig, OnyxEnv, OnyxClient

config = OnyxConfig(
    domain=os.environ[OnyxEnv.DOMAIN],
    token=os.environ[OnyxEnv.TOKEN],
)

with OnyxClient(config) as client, open("/path/to/file.csv") as csv_file:
    results = client.csv_delete(
        project="project",
        csv_file=csv_file,
        multiline=True,
    )
```

```
>>> results
[
  {"climb_id": "C-1234567890"},
  {"climb_id": "C-1234567891"},
  {"climb_id": "C-1234567892"},
]
```

`analysis_fields(project)`

View analysis fields.

PARAMETER	DESCRIPTION
<code>project</code>	Name of the project. TYPE: <code>str</code>

RETURNS	DESCRIPTION
<code>Dict[str, Any]</code>	Dict of fields.

Examples:

```
import os
from onyx import OnyxConfig, OnyxEnv, OnyxClient

config = OnyxConfig(
    domain=os.environ[OnyxEnv.DOMAIN],
    token=os.environ[OnyxEnv.TOKEN],
)

with OnyxClient(config) as client:
    fields = client.analysis_fields("project")
```

`analysis_choices(project, field)`

View choices for an analysis field.

PARAMETER	DESCRIPTION
<code>project</code>	Name of the project. TYPE: <code>str</code>
<code>field</code>	Analysis choice field. TYPE: <code>str</code>

RETURNS	DESCRIPTION
<code>Dict[str, Dict[str, Any]]</code>	Dictionary mapping choices to information about the choice.

Examples:

```
import os
from onyx import OnyxConfig, OnyxEnv, OnyxClient

config = OnyxConfig(
    domain=os.environ[OnyxEnv.DOMAIN],
    token=os.environ[OnyxEnv.TOKEN],
)

with OnyxClient(config) as client:
    choices = client.analysis_choices("project", "country")
```

```
get_analysis(project, analysis_id=None, fields=None, include=None, exclude=None)
```

Get an analysis from a project.

PARAMETER	DESCRIPTION
<code>project</code>	Name of the project. TYPE: <code>str</code>
<code>analysis_id</code>	Unique identifier for the analysis. TYPE: <code>Optional[str]</code> DEFAULT: <code>None</code>
<code>fields</code>	Dictionary of field filters used to uniquely identify the record. TYPE: <code>Optional[Dict[str, Any]]</code> DEFAULT: <code>None</code>
<code>include</code>	Fields to include in the output. TYPE: <code>Union[List[str], str, None]</code> DEFAULT: <code>None</code>
<code>exclude</code>	Fields to exclude from the output. TYPE: <code>Union[List[str], str, None]</code> DEFAULT: <code>None</code>

RETURNS	DESCRIPTION
<code>Dict[str, Any]</code>	Dict containing the record.

Examples:

Get an analysis by analysis ID:

```
import os
from onyx import OnyxConfig, OnyxEnv, OnyxClient

config = OnyxConfig(
    domain=os.environ[OnyxEnv.DOMAIN],
    token=os.environ[OnyxEnv.TOKEN],
)

with OnyxClient(config) as client:
    analysis = client.get_analysis("project", "A-1234567890")
```

```
>>> analysis
{
  "analysis_id": "A-1234567890",
  "published_date": "2023-01-01",
  "name": "Very cool analysis",
  "result": "Found very cool things",
}
```

```
filter_analysis(project, fields=None, include=None, exclude=None, summarise=None, **kwargs)
```

Filter analyses from a project.

PARAMETER	DESCRIPTION
<code>project</code>	Name of the project. TYPE: <code>str</code>
<code>fields</code>	Dictionary of field filters. TYPE: <code>Optional[Dict[str, Any]]</code> DEFAULT: <code>None</code>
<code>include</code>	Fields to include in the output. TYPE: <code>Union[List[str], str, None]</code> DEFAULT: <code>None</code>
<code>exclude</code>	Fields to exclude from the output. TYPE: <code>Union[List[str], str, None]</code> DEFAULT: <code>None</code>
<code>summarise</code>	For a given field (or group of fields), return the frequency of each unique value (or unique group of values). TYPE: <code>Union[List[str], str, None]</code> DEFAULT: <code>None</code>
<code>**kwargs</code>	Additional keyword arguments are interpreted as field filters. TYPE: <code>Any</code> DEFAULT: <code>{}</code>

RETURNS	DESCRIPTION
<code>None</code>	Generator of analyses. If a summarise argument is provided, each record will be a dict containing values of the summary fields and a count for the frequency.

Examples:

Retrieve all analyses that match a set of field requirements:

```
import os
from onyx import OnyxConfig, OnyxEnv, OnyxClient

config = OnyxConfig(
    domain=os.environ[OnyxEnv.DOMAIN],
    token=os.environ[OnyxEnv.TOKEN],
)

with OnyxClient(config) as client:
    analyses = list(
        client.filter_analysis(
            project="project",
            published_date__range=["2023-01-01", "2023-01-02"],
        )
    )
```

```
>>> analyses
[
  {
    "analysis_id": "A-1234567890",
    "published_date": "2023-01-01",
    "name": "Very cool analysis",
    "result": "Found very cool things",
  },
  {
    "analysis_id": "A-1234567891",
    "published_date": "2023-01-02",
    "name": "Not so cool analysis",
    "result": "Found not so cool things",
  },
]
```


os ▾

- See the documentation for the `filter` method for more information on filtering records, as this also applies to analyses.

`analysis_history(project, analysis_id)`

View the history of an analysis in a project.

PARAMETER	DESCRIPTION
<code>project</code>	Name of the project. TYPE: <code>str</code>
<code>analysis_id</code>	Unique identifier for the analysis. TYPE: <code>str</code>

RETURNS	DESCRIPTION
<code>Dict[str, Any]</code>	Dict containing the history of the analysis.

Examples:

```
import os
from onyx import OnyxConfig, OnyxEnv, OnyxClient

config = OnyxConfig(
    domain=os.environ[OnyxEnv.DOMAIN],
    token=os.environ[OnyxEnv.TOKEN],
)

with OnyxClient(config) as client:
    history = client.analysis_history("project", "A-1234567890")
```

```
>>> history
{
  "analysis_id": "A-1234567890",
  "history": [
    {
      "username": "user",
      "timestamp": "2023-01-01T00:00:00Z",
      "action": "add",
    },
    {
      "username": "user",
      "timestamp": "2023-01-02T00:00:00Z",
      "action": "change",
      "changes": [
        {
          "field": "name",
          "type": "text",
          "from": "Cool analysis",
          "to": "Very cool analysis",
        },
      ],
    },
  ],
}
```

analysis_records(project, analysis_id)

View the records involved in an analysis in a project.

PARAMETER	DESCRIPTION
project	Name of the project. TYPE: str
analysis_id	Unique identifier for the analysis. TYPE: str

RETURNS	DESCRIPTION
List[Dict[str, Any]]	List of Dicts containing basic details of each record involved in the analysis.

Examples:

```
import os
from onyx import OnyxConfig, OnyxEnv, OnyxClient

config = OnyxConfig(
    domain=os.environ[OnyxEnv.DOMAIN],
    token=os.environ[OnyxEnv.TOKEN],
)

with OnyxClient(config) as client:
    records = client.analysis_records("project", "A-1234567890")
```

```
>>> records
[
  {
    "climb_id": "C-1234567890",
    "published_date": "2023-01-01",
    "site": "site_1",
  },
  {
    "climb_id": "C-1234567891",
    "published_date": "2023-01-02",
    "site": "site_2",
  },
]
```

create_analysis(project, fields, test=False)

Create an analysis in a project.

PARAMETER	DESCRIPTION
project	Name of the project. TYPE: str
fields	Object representing the analysis to be created. TYPE: Dict[str, Any]
test	If True, runs the command as a test. Default: False TYPE: bool DEFAULT: False

RETURNS	DESCRIPTION
Dict[str, Any]	Dict containing the Analysis ID of the created analysis.

Examples:

```
import os
from onyx import OnyxConfig, OnyxEnv, OnyxClient
```

```
config = OnyxConfig(
    domain=os.environ[OnyxEnv.DOMAIN],
    token=os.environ[OnyxEnv.TOKEN],
)

with OnyxClient(config) as client:
    result = client.create_analysis(
        "project",
        fields={
            "name": "Absolutely incredible analysis",
            "result": "Insane results",
        },
    )

>>> result
{"analysis_id": "A-1234567890"}
```

`update_analysis(project, analysis_id, fields=None, test=False, clear=None)`

Update an analysis in a project.

PARAMETER	DESCRIPTION
<code>project</code>	Name of the project. TYPE: <code>str</code>
<code>analysis_id</code>	Unique identifier for the analysis. TYPE: <code>str</code>
<code>fields</code>	Object representing the analysis to be updated. TYPE: <code>Optional[Dict[str, Any]]</code> DEFAULT: <code>None</code>
<code>test</code>	If <code>True</code> , runs the command as a test. Default: <code>False</code> TYPE: <code>bool</code> DEFAULT: <code>False</code>
<code>clear</code>	Field or list of fields to clear. Resets nested fields to empty. TYPE: <code>Union[List[str], str, None]</code> DEFAULT: <code>None</code>

RETURNS	DESCRIPTION
<code>Dict[str, Any]</code>	Dict containing the Analysis ID of the updated analysis.

Examples:

```
import os
from onyx import OnyxConfig, OnyxEnv, OnyxClient

config = OnyxConfig(
    domain=os.environ[OnyxEnv.DOMAIN],
    token=os.environ[OnyxEnv.TOKEN],
)

with OnyxClient(config) as client:
    result = client.update_analysis(
        project="project",
        analysis_id="A-1234567890",
        fields={
            "result": "The results were even more insane",
        },
        clear=["report"],
    )

>>> result
{"analysis_id": "A-1234567890"}
```

```
delete_analysis(project, analysis_id)
```

Delete an analysis in a project.

PARAMETER	DESCRIPTION
<code>project</code>	Name of the project. TYPE: <code>str</code>
<code>analysis_id</code>	Unique identifier for the analysis. TYPE: <code>str</code>

RETURNS	DESCRIPTION
<code>Dict[str, Any]</code>	Dict containing the Analysis ID of the deleted analysis.

Examples:

```
import os
from onyx import OnyxConfig, OnyxEnv, OnyxClient

config = OnyxConfig(
    domain=os.environ[OnyxEnv.DOMAIN],
    token=os.environ[OnyxEnv.TOKEN],
)

with OnyxClient(config) as client:
    result = client.delete_analysis(
        project="project",
        analysis_id="A-1234567890",
    )
```

```
>>> result
{"analysis_id": "A-1234567890"}
```

```
register(domain, first_name, last_name, email, site, password) classmethod
```

Create a new user.

PARAMETER	DESCRIPTION
<code>domain</code>	Name of the domain. TYPE: <code>str</code>
<code>first_name</code>	First name of the user. TYPE: <code>str</code>
<code>last_name</code>	Last name of the user. TYPE: <code>str</code>
<code>email</code>	Email address of the user. TYPE: <code>str</code>
<code>site</code>	Name of the site. TYPE: <code>str</code>
<code>password</code>	Password for the user. TYPE: <code>str</code>

RETURNS	DESCRIPTION
<code>Dict[str, Any]</code>	Dict containing the user's information.

Examples:

```
import os
from onyx import OnyxClient, OnyxEnv

registration = OnyxClient.register(
    domain=os.environ[OnyxEnv.DOMAIN],
    first_name="Bill",
    last_name="Will",
    email="bill@email.com",
    site="site",
    password="pass123",
)
```

```
>>> registration
{
  "username": "onyx-willb",
  "site": "site",
  "email": "bill@email.com",
  "first_name": "Bill",
  "last_name": "Will",
}
```

login()

Log in the user.

RETURNS	DESCRIPTION
<code>Dict[str, Any]</code>	Dict containing the user's authentication token and it's expiry.

Examples:

```
import os
from onyx import OnyxConfig, OnyxEnv, OnyxClient

config = OnyxConfig(
    domain=os.environ[OnyxEnv.DOMAIN],
```

```
        username=os.environ[OnyxEnv.USERNAME],
        password=os.environ[OnyxEnv.PASSWORD],
    )

    with OnyxClient(config) as client:
        token = client.login()
```

```
>>> token
{
    "expiry": "2024-01-01T00:00:00.000000Z",
    "token": "abc123",
}
```

logout()

Log out the user.

Examples:

```
import os
from onyx import OnyxConfig, OnyxEnv, OnyxClient

config = OnyxConfig(
    domain=os.environ[OnyxEnv.DOMAIN],
    token=os.environ[OnyxEnv.TOKEN],
)

with OnyxClient(config) as client:
    client.logout()
```

logoutall()

Log out the user in all clients.

Examples:

```
import os
from onyx import OnyxConfig, OnyxEnv, OnyxClient

config = OnyxConfig(
    domain=os.environ[OnyxEnv.DOMAIN],
    token=os.environ[OnyxEnv.TOKEN],
)

with OnyxClient(config) as client:
    client.logoutall()
```

profile()

View the user's information.

RETURNS	DESCRIPTION
Dict[str, str]	Dict containing the user's information.

Examples:

```
import os
from onyx import OnyxConfig, OnyxEnv, OnyxClient

config = OnyxConfig(
    domain=os.environ[OnyxEnv.DOMAIN],
    token=os.environ[OnyxEnv.TOKEN],
)

with OnyxClient(config) as client:
    profile = client.profile()
```

```
>>> profile
{
    "username": "user",
    "site": "site",
    "email": "user@email.com",
}
```

activity()

View the user's latest activity.

RETURNS	DESCRIPTION
List[Dict[str, Any]]	List of the user's latest activity.

Examples:

```
import os
from onyx import OnyxConfig, OnyxEnv, OnyxClient

config = OnyxConfig(
    domain=os.environ[OnyxEnv.DOMAIN],
    token=os.environ[OnyxEnv.TOKEN],
)

with OnyxClient(config) as client:
    activity = client.activity()
```

```
>>> activity
[
  {
    "date": "2023-01-01T00:00:00.000000Z",
    "address": "127.0.0.1",
    "endpoint": "/projects/project/",
    "method": "POST",
    "status": 400,
    "exec_time": 29,
    "error_messages": "b'{'status':'fail','code':400,'messages':{'site':['Select a valid choice.']}'}",
  },
  {
    "timestamp": "2023-01-02T00:00:00.000000Z",
    "address": "127.0.0.1",
    "endpoint": "/accounts/activity/",
    "method": "GET",
    "status": 200,
    "exec_time": 22,
    "error_messages": "",
  },
]
```

approve(username)

Approve another user.

PARAMETER	DESCRIPTION
username	Username of the user to be approved.
TYPE: str	

RETURNS	DESCRIPTION
Dict[str, Any]	Dict confirming user approval success.

Examples:

```
import os
from onyx import OnyxConfig, OnyxEnv, OnyxClient

config = OnyxConfig(
    domain=os.environ[OnyxEnv.DOMAIN],
    token=os.environ[OnyxEnv.TOKEN],
)

with OnyxClient(config) as client:
    approval = client.approve("waiting_user")
```

```
>>> approval
{
  "username": "waiting_user",
  "is_approved": True,
}
```

waiting()

Get users waiting for approval.

RETURNS	DESCRIPTION
List[Dict[str, Any]]	List of users waiting for approval.

Examples:

```
import os
from onyx import OnyxConfig, OnyxEnv, OnyxClient

config = OnyxConfig(
    domain=os.environ[OnyxEnv.DOMAIN],
    token=os.environ[OnyxEnv.TOKEN],
)

with OnyxClient(config) as client:
    users = client.waiting()
```

```
>>> users
[
  {
    "username": "waiting_user",
    "site": "site",
    "email": "waiting_user@email.com",
    "date_joined": "2023-01-01T00:00:00.000000Z",
  }
]
```

site_users()

Get users within the site of the requesting user.

RETURNS	DESCRIPTION
List[Dict[str, Any]]	List of users within the site of the requesting user.

Examples:

```
import os
from onyx import OnyxConfig, OnyxEnv, OnyxClient

config = OnyxConfig(
    domain=os.environ[OnyxEnv.DOMAIN],
    token=os.environ[OnyxEnv.TOKEN],
)

with OnyxClient(config):
    users = client.site_users()
```

```
>>> users
[
  {
    "username": "user",
    "site": "site",
    "email": "user@email.com",
  }
]
```

all_users()

Get all users.

RETURNS	DESCRIPTION
List[Dict[str, Any]]	List of all users.

Examples:

```
import os
from onyx import OnyxConfig, OnyxEnv, OnyxClient
```



```
config = OnyxConfig(  
    domain=os.environ[OnyxEnv.DOMAIN],  
    token=os.environ[OnyxEnv.TOKEN],  
)  
  
with OnyxClient(config) as client:  
    users = client.all_users()
```

```
>>> users  
[  
    {  
        "username": "user",  
        "site": "site",  
        "email": "user@email.com",  
    },  
    {  
        "username": "another_user",  
        "site": "another_site",  
        "email": "another_user@email.com",  
    },  
]
```

1.5.2 OnyxConfig

Class for storing information required to connect/authenticate with Onyx.

```
__init__(domain, token=None, username=None, password=None)
```

Initialise a config.

This object stores information required to connect and authenticate with Onyx.

A domain must be provided, alongside an API token and/or the username + password.

PARAMETER	DESCRIPTION
domain	Domain for connecting to Onyx. TYPE: str
token	Token for authenticating with Onyx. TYPE: Optional[str] DEFAULT: None
username	Username for authenticating with Onyx. TYPE: Optional[str] DEFAULT: None
password	Password for authenticating with Onyx. TYPE: Optional[str] DEFAULT: None

Examples:

Create a config using environment variables for the domain and an API token:

```
import os
from onyx import OnyxConfig, OnyxEnv

config = OnyxConfig(
    domain=os.environ[OnyxEnv.DOMAIN],
    token=os.environ[OnyxEnv.TOKEN],
)
```

Or using environment variables for the domain and login credentials:

```
import os
from onyx import OnyxConfig, OnyxEnv

config = OnyxConfig(
    domain=os.environ[OnyxEnv.DOMAIN],
    username=os.environ[OnyxEnv.USERNAME],
    password=os.environ[OnyxEnv.PASSWORD],
)
```

1.5.3 OnyxEnv

Class containing recommended environment variable names for Onyx.

If environment variables are created with these recommended names, then the attributes of this class can be used to access them.

These attributes and the recommended environment variable names are:

```
OnyxEnv.DOMAIN = "ONYX_DOMAIN"
OnyxEnv.TOKEN = "ONYX_TOKEN"
OnyxEnv.USERNAME = "ONYX_USERNAME"
OnyxEnv.PASSWORD = "ONYX_PASSWORD"
```

Examples:

In the shell, create the following environment variables with your credentials:

```
$ export ONYX_DOMAIN="https://onyx.example.domain"
$ export ONYX_TOKEN="example-onyx-token"
$ export ONYX_USERNAME="example-onyx-username"
$ export ONYX_PASSWORD="example-onyx-password"
```

Then access them in Python to create an OnyxConfig object:

```
import os
from onyx import OnyxEnv, OnyxConfig

config = OnyxConfig(
    domain=os.environ[OnyxEnv.DOMAIN],
    token=os.environ[OnyxEnv.TOKEN],
    username=os.environ[OnyxEnv.USERNAME],
    password=os.environ[OnyxEnv.PASSWORD],
)
```

1.5.4 OnyxField

Class that represents a single field-value pair for use in Onyx queries.

```
__init__(**kwargs)
```

Initialise a field.

PARAMETER	DESCRIPTION
<code>**kwargs</code>	Keyword arguments containing a single key-value pair.
TYPE: Any	DEFAULT: {}

Notes ▾

- Takes a single key-value argument as input.
- The key corresponds to a field (and optional lookup) to use for filtering.
- The value corresponds to the field value(s) that are being matched against.
- `OnyxField` instances can be combined into complex expressions using Python's bitwise operators: `&` (AND), `|` (OR), `^` (XOR), and `~` (NOT).
- Multi-value lookups (e.g. `in`, `range`) support passing a Python list as the value. These are coerced into comma-separated strings internally.

Examples:

Create `OnyxField` objects and combine them using Python bitwise operators:

```
from onyx import OnyxField

field1 = OnyxField(field1="value1")
field2 = OnyxField(field2__contains="value2")

expression = (field1 | field2) & OnyxField(
    published_date__range=["2023-01-01", "2023-01-02"]
)
```

```
>>> field1
<onyx.field.OnyxField object at 0x1028eb850>
>>> field2
<onyx.field.OnyxField object at 0x1028eb850>
>>> expression
<onyx.field.OnyxField object at 0x103b6fc40>
>>> field1.query
{"field1": "value1"}
>>> field2.query
{"field2__contains": "value2"}
>>> expression.query
{
  "&": [
    {"|": [{"field1": "value1"}, {"field2__contains": "value2"}]},
    {"published_date__range": "2023-01-01,2023-01-02"},
  ]
}
```

1.5.5 Exceptions

`OnyxError`

Bases: `Exception`

Generic class for all Onyx exceptions.

`OnyxConfigError`

Bases: `OnyxError`

Config validation error.

This error occurs due to validation failures when initialising an `OnyxConfig` object.

Examples:

- A `domain` was not provided.
- Neither a `token` or valid login credentials (`username` and `password`) were provided.

`OnyxClientError`

Bases: `OnyxError`

Client validation error.

This error occurs due to validation failures within an `OnyxClient` object, and **not** due to error codes returned by the Onyx API.

Examples:

- Incorrect types were provided to `OnyxClient` methods.
- Empty strings were provided for required arguments such as the `climb_id`, creating an invalid URL.
- Empty CSV/TSV files are provided on `OnyxClient.csv_create`, `OnyxClient.csv_update`, or `OnyxClient.csv_delete`.
- CSV/TSV files with more than one record are provided to `OnyxClient.csv_create`, `OnyxClient.csv_update`, or `OnyxClient.csv_delete` when `multiline = False`.

tes ▾

- One counter-intuitive cause of this error is when an `OnyxClient.get` request using `fields` returns more than one result.
- This is not an `OnyxRequestError` because for this particular combination of parameters, an underlying call to the `OnyxClient.filter` method is made.
- The request to the Onyx API may be successful, but return more than one record. However, the `OnyxClient.get` method expects a single record, resulting in the error being raised.
- This behaviour may change in the future.

`OnyxFieldError`

Bases: `OnyxError`

Field validation error.

This error occurs due to validation failures within the `OnyxField` class.

Examples:

- The user did not provide exactly one key-value pair on initialisation.
- An attempt was made to combine an `OnyxField` instance with a different type.
- The structure of the underlying `OnyxField.query` is somehow incorrect.

OnyxConnectionErrorBases: `OnyxError`

Onyx connection error.

This error occurs due to a failure to connect to the Onyx API.

tes ▾

- This error occurs due to any subclass of `requests.RequestException` (excluding `requests.HTTPError`) being raised.
- For more information, see: <https://requests.readthedocs.io/en/latest/api/#requests.RequestException>

OnyxHTTPErrorBases: `OnyxError`

Onyx HTTP error.

This error occurs due to a request to the Onyx API either failing (code `4xx`) or causing a server error (code `5xx`).**tes** ▾

- This error occurs due to a `requests.HTTPError` being raised.
- Like the `requests.HTTPError` class, instances of this class have a `response` object containing details of the error.
- For more information on the `response` object, see: <https://requests.readthedocs.io/en/latest/api/#requests.Response>

Examples:

```
import os
from onyx import OnyxConfig, OnyxEnv, OnyxClient, OnyxField
from onyx.exceptions import OnyxHTTPError

config = OnyxConfig(
    domain=os.environ[OnyxEnv.DOMAIN],
    token=os.environ[OnyxEnv.TOKEN],
)

with OnyxClient(config) as client:
    try:
        records = list(
            client.query(
                project="project",
                query=(
                    OnyxField(field1="abcd")
                    & OnyxField(published_date__range=["2023-01-01", "2023-01-02"])
                ),
            )
        )
    except OnyxHTTPError as e:
        print(e.response.json())
```

OnyxRequestErrorBases: `OnyxHTTPError`

Onyx request error.

This error occurs due to a failed request to the Onyx API (code `4xx`).

Examples:

- Invalid field names or field values (`400 Bad Request`).
- Invalid authentication credentials (`401 Unauthorized`).
- A request was made for something which the user has insufficient permissions for (`403 Forbidden`).
- An invalid project / CLIMB ID / anonymised value was provided (`404 Not Found`).
- An invalid HTTP method was used (`405 Method Not Allowed`).

`OnyxServerError`

Bases: `OnyxHTTPError`

Onyx server error.

This error occurs due to a request to the Onyx API causing a server error (code `5xx`).



Server errors are unintended and should be reported to an admin if encountered.