Onyx-client

CLIMB-TRE

None

Table of contents

1. Onyx-client

1.1 Introduction

This site documents 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.

1.2 Installation

1.2.1 Install from conda-forge (recommended)

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

This installs the latest version of the Onyx-Client from conda-forge.

1.2.2 Install from PyPI

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

This installs the latest version of the Onyx-Client from PyPI.

1.2.3 Build from source

Download the source code from Github:

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

Run installation from within the source code directory:

```
$ cd onyx-client/
$ 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. To disable them, create an environment variable `ONYX_COLOURS` with the value `NONE`:

```
$ export ONYX_COLOURS=NONE
```

To re-enable colours, unset the environment variable:

```
$ unset ONYX_COLOURS
```

2. Command-line interface

2.1 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.

2.1.1 Profile information

```
$ onyx profile
```

2.1.2 Available projects

```
$ onyx projects
```

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

2.1.3 Project fields

```
$ onyx fields PROJECT
```

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

2.1.4 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
```

2.1.5 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
```

2.2 `onyx`

API for pathogen metadata.

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.
- `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.
- `profile` : View profile information.
- `activity` : View latest profile activity.
- `siteusers` : View users from the same site.
- `auth` : Authentication commands.
- `admin` : Admin commands.

2.2.1 `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.

2.2.2 `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.

2.2.3 `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.

2.2.4 `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.

2.2.5 `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.

2.2.6 `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.

2.2.7 `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.

2.2.8 `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.

2.2.9 `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.

2.2.10 `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.

2.2.11 `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)]
- `--help`: Show this message and exit.

2.2.12 `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.

2.2.13 `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.

2.2.14 `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.

2.2.15 `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.

2.2.16 `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.

2.2.17 `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.

3. Python API

3.1 Documentation

3.1.1 OnyxClient

Bases: `OnyxClientBase`

Class for querying and manipulating data within Onyx.

`__init__(config)`

Initialise a client.

PARAMETER	DESCRIPTION
`config`	`OnyxConfig` object that stores information for connecting and authenticating with Onyx. **TYPE:** `OnyxConfig`

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
```

ɔs

- 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
`List[Dict[str, str]]`	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:

```python
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
{
    "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
`project`	Name of the project. **TYPE:** `str`
`field`	Choice field on the project. **TYPE:** `str`

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

RETURNS	DESCRIPTION
`Dict[str, Any]`	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",
}
```

Tips

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

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

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

tes

- 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
`csv_file`	File object for the CSV file being written to. **TYPE:** `TextIO`
`data`	The data being written to the CSV file. Must be either a list / generator of dict records. **TYPE:** `Union[List[Dict[str, Any]], Generator[Dict[str, Any], Any, 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`

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
`project`	Name of the project. **TYPE:** `str`
`climb_id`	Unique identifier for the record in the project. **TYPE:** `str`

RETURNS	DESCRIPTION
`Dict[str, Any]`	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,
                },
            ],
        },
    ],
}
```

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

Get the anonymised identifier for a value on a field.

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

RETURNS	DESCRIPTION
`Dict[str, str]`	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)`

Update a record in a project.

PARAMETER	DESCRIPTION
`project`	Name of the project. **TYPE:** `str`
`climb_id`	Unique identifier for the record in the project. **TYPE:** `str`
`fields`	Object representing the record to be updated. **TYPE:** `Optional[Dict[str, Any]]` **DEFAULT:** `None`
`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 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",
        },
    )
```

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

`delete(project, climb_id)`

Delete a record in a project.

PARAMETER	DESCRIPTION
`project`	Name of the project. **TYPE:** `str`
`climb_id`	Unique identifier for the record in the project. **TYPE:** `str`

RETURNS	DESCRIPTION
`Dict[str, Any]`	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
`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 updated record. If `multiline = True`, 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
`project`	Name of the project. **TYPE:** `str`
`csv_file`	File object for the CSV file being used for record upload. **TYPE:** `TextIO`
`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`

RETURNS	DESCRIPTION
`Union[Dict[str, Any], List[Dict[str, Any]]]`	Dict containing the CLIMB ID of the deleted record. If `multiline = True`, 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"},
]
```

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

Create a new user.

PARAMETER	DESCRIPTION
`domain`	Name of the domain. **TYPE:** `str`
`first_name`	First name of the user. **TYPE:** `str`
`last_name`	Last name of the user. **TYPE:** `str`
`email`	Email address of the user. **TYPE:** `str`
`site`	Name of the site. **TYPE:** `str`
`password`	Password for the user. **TYPE:** `str`

RETURNS	DESCRIPTION
`Dict[str, Any]`	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
`Dict[str, Any]`	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",
    },
]
```

3.1.2 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],
)
```

3.1.3 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],
)
```

3.1.4 OnyxField

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

`__init__(**kwargs)`

Initialise a field.

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

tes

- 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"},
    ]
}
```

3.1.5 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.

`OnyxConnectionError`

Bases: `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

`OnyxHTTPError`

Bases: `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())
```

`OnyxRequestError`

Bases: `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`).

Warning

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

4. Types & Lookups

4.1 Types

Types in Onyx define the various categories of data which can be stored.

Each field belongs to a certain type. This dictates what kind of data the field can store (e.g. text, numbers, dates, etc.), as well as what filter operations (i.e. lookups) can be carried out on values of the field.

4.1.1 `text`

`[exact]` `[ne]` `[in]` `[notin]` `[contains]` `[startswith]` `[endswith]` `[iexact]` `[icontains]` `[istartswith]` `[iendswith]` `[length]` `[length__in]` `[length__range]` `[isnull]`

A string of characters.

Examples: `"C-1234567890"`, `"Details about something"`

4.1.2 `choice`

`[exact]` `[ne]` `[in]` `[notin]` `[isnull]`

A restricted set of options.

Examples: `"ENG"`, `"WALES"`, `"SCOT"`, `"NI"`

4.1.3 `integer`

`[exact]` `[ne]` `[in]` `[notin]` `[lt]` `[lte]` `[gt]` `[gte]` `[range]` `[isnull]`

A whole number.

Examples: `1`, `-1`, `123`

4.1.4 `decimal`

`[exact]` `[ne]` `[in]` `[notin]` `[lt]` `[lte]` `[gt]` `[gte]` `[range]` `[isnull]`

A decimal number.

Examples: `1.234`, `1.0`, `23.456`

4.1.5 `date`

`[exact]` `[ne]` `[in]` `[notin]` `[lt]` `[lte]` `[gt]` `[gte]` `[range]` `[iso_year]` `[iso_year__in]` `[iso_year__range]` `[week]` `[week__in]` `[week__range]` `[isnull]`

A date.

Examples: `"2023-03"`, `"2023-04-05"`, `"2024-01-01"`

4.1.6 `datetime`

`[exact]` `[ne]` `[in]` `[notin]` `[lt]` `[lte]` `[gt]` `[gte]` `[range]` `[iso_year]` `[iso_year__in]` `[iso_year__range]` `[week]` `[week__in]` `[week__range]` `[isnull]`

A date and time.

Examples: `"2023-01-01 15:30:03"`, `"2024-01-01 09:30:17"`

4.1.7 `bool`

`[exact]` `[ne]` `[in]` `[notin]` `[isnull]`

A true or false value.

Examples: `True`, `False`

4.1.8 `relation`

`[isnull]`

A link to a row, or multiple rows, in another table.

4.1.9 `array`

`[exact]` `[contains]` `[contained_by]` `[overlap]` `[length]` `[length__in]` `[length__range]` `[isnull]`

A list of values.

Examples: `[1, 2, 3]`, `["hello", "world", "!"]`

4.1.10 `structure`

`[exact]` `[contains]` `[contained_by]` `[has_key]` `[has_keys]` `[has_any_keys]` `[isnull]`

An arbitrary JSON structure.

Examples: `{"hello" : "world", "goodbye" : "!"}`, `{"numbers" : [1, 2, {"more_numbers" : [3, 4, 5]}]}`

4.2 Lookups

Lookups can be used to specify more complex conditions that fields must match when filtering.

Different types have different lookups available to them.

4.2.1 `exact`

`[text]` `[choice]` `[integer]` `[decimal]` `[date]` `[datetime]` `[bool]` `[array]` `[structure]`

Return values equal to the search value.

4.2.2 `ne`

`[text]` `[choice]` `[integer]` `[decimal]` `[date]` `[datetime]` `[bool]`

Return values not equal to the search value.

4.2.3 `in`

`[text]` `[choice]` `[integer]` `[decimal]` `[date]` `[datetime]` `[bool]`

Return values that are within the set of search values.

4.2.4 `notin`

`[text]` `[choice]` `[integer]` `[decimal]` `[date]` `[datetime]` `[bool]`

Return values that are not within the set of search values.

4.2.5 `contains`

`[text]` `[array]` `[structure]`

Return values that contain the search value.

4.2.6 `startswith`

`[text]`

Return values that start with the search value.

4.2.7 `endswith`

`[text]`

Return values that end with the search value.

4.2.8 `iexact`

`[text]`

Return values case-insensitively equal to the search value.

4.2.9 `icontains`

`[text]`

Return values that case-insensitively contain the search value.

4.2.10 `istartswith`

`[text]`

Return values that case-insensitively start with the search value.

4.2.11 `iendswith`

`[text]`

Return values that case-insensitively end with the search value.

4.2.12 `length`

`[text]` `[array]`

Return values with a length equal to the search value.

4.2.13 `length__in`

`[text]` `[array]`

Return values with a length that is within the set of search values.

4.2.14 `length__range`

`[text]` `[array]`

Return values with a length that is within an inclusive range of search values.

4.2.15 `lt`

`[integer]` `[decimal]` `[date]` `[datetime]`

Return values less than the search value.

4.2.16 `lte`

`[integer]` `[decimal]` `[date]` `[datetime]`

Return values less than or equal to the search value.

4.2.17 `gt`

`[integer]` `[decimal]` `[date]` `[datetime]`

Return values greater than the search value.

4.2.18 `gte`

`[integer]` `[decimal]` `[date]` `[datetime]`

Return values greater than or equal to the search value.

4.2.19 `range`

`[integer]` `[decimal]` `[date]` `[datetime]`

Return values within an inclusive range of search values.

4.2.20 `iso_year`

`[date]` `[datetime]`

Return values with an ISO 8601 week-numbering year equal to the search year.

4.2.21 `iso_year__in`

`[date]` `[datetime]`

Return values with an ISO 8601 week-numbering year that is within the set of search years.

4.2.22 `iso_year__range`

`[date]` `[datetime]`

Return values with an ISO 8601 week-numbering year that is within an inclusive range of search years.

4.2.23 `week`

`[date]` `[datetime]`

Return values with an ISO 8601 week number equal to the search number.

4.2.24 `week__in`

`[date]` `[datetime]`

Return values with an ISO 8601 week number that is within the set of search numbers.

4.2.25 `week__range`

`[date]` `[datetime]`

Return values with an ISO 8601 week number that is within an inclusive range of search numbers.

4.2.26 `isnull`

`[text]` `[choice]` `[integer]` `[decimal]` `[date]` `[datetime]` `[bool]` `[relation]` `[array]` `[structure]`

Return values that are empty (`isnull = True`) or non-empty (`isnull = False`).

- For `text` and `choice` types, 'empty' is defined as the empty string `""`.
- For the `relation` type, 'empty' is defined as there being zero items linked to the record being evaluated.
- For the `array` type, 'empty' is defined as the empty array `[]`.
- For the `structure` type, 'empty' is defined as the empty structure `{}`.
- For all other types, 'empty' is the SQL `null` value.

4.2.27 `contained_by`

`[array]` `[structure]`

Return values that are equal to, or a subset of, the search value.

4.2.28 `overlap`

`[array]`

Return values that overlap with the search value.

4.2.29 `has_key`

`[structure]`

Return values that have a top-level key which contains the search value.

4.2.30 `has_keys`

`[structure]`

Return values that have top-level keys which contains all of the search values.

4.2.31 `has_any_keys`

`[structure]`

Return values that have top-level keys which contains any of the search values.