python-zyte-api Documentation

Release 0.5.0

Zyte Group Ltd

Apr 12, 2024

GETTING STARTED

Command-line client and Python client library for Zyte API.

CHAPTER

ONE

INSTALLATION

```
pip install zyte-api
```

Note: Python 3.8+ is required.

CHAPTER

TWO

BASIC USAGE

2.1 Set your API key

After you sign up for a Zyte API account, copy your API key.

2.2 Use the command-line client

Then you can use the zyte-api command-line client to send Zyte API requests. First create a text file with a list of URLs:

```
https://books.toscrape.com
https://quotes.toscrape.com
```

And then call `zyte-api` from your shell:

```
zyte-api url-list.txt --api-key YOUR_API_KEY --output results.jsonl
```

2.3 Use the Python sync API

For very basic Python scripts, use the sync API:

```
from zyte_api import ZyteAPI

client = ZyteAPI(api_key="YOUR_API_KEY")
response = client.get({"url": "https://toscrape.com", "httpResponseBody": True})
```

2.4 Use the Python async API

For asyncio code, use the async API:

```
import asyncio

from zyte_api import AsyncZyteAPI


async def main():
```

(continues on next page)

(continued from previous page)

```
    client = AsyncZyteAPI(api_key="YOUR_API_KEY")
    response = await client.get(
        {"url": "https://toscrape.com", "httpResponseBody": True}
    )


asyncio.run(main())
```

CHAPTER

THREE

API KEY

After you sign up for a Zyte API account, copy your API key.

It is recommended to configure your API key through an environment variable, so that it can be picked by both the *command-line client* and the *Python client library*:

- On Windows:

```
> set ZYTE_API_KEY=YOUR_API_KEY
```

- On macOS and Linux:

```
$ export ZYTE_API_KEY=YOUR_API_KEY
```

Alternatively, you may pass your API key to the clients directly:

- To pass your API key directly to the command-line client, use the `--api-key` switch:

```
zyte-api --api-key YOUR_API_KEY ...
```

- To pass your API key directly to the Python client classes, use the `api_key` parameter when creating a client object:

```
from zyte_api import ZyteAPI

client = ZyteAPI(api_key="YOUR_API_KEY")
```

```
from zyte_api import AsyncZyteAPI

client = AsyncZyteAPI(api_key="YOUR_API_KEY")
```

CHAPTER

FOUR

COMMAND-LINE CLIENT

Once you have *installed python-zyte-api* and *configured your API key*, you can use the `zyte-api` command-line client.

To use `zyte-api`, pass an *input file* as the first parameter and specify an *output file* with `--output`. For example:

```
zyte-api urls.txt --output result.jsonl
```

4.1 Input file

The input file can be either of the following:

- A plain-text file with a list of target URLs, one per line. For example:

  ```
  https://books.toscrape.com
  https://quotes.toscrape.com
  ```

  For each URL, a Zyte API request will be sent with browserHtml set to `True`.

- A JSON Lines file with a object of Zyte API request parameters per line. For example:

  ```
  {"url": "https://a.example", "browserHtml": true, "geolocation": "GB"}
  {"url": "https://b.example", "httpResponseBody": true}
  {"url": "https://books.toscrape.com", "productNavigation": true}
  ```

4.2 Output file

You can specify the path to an output file with the `--output`/`-o` switch. If not specified, the output is printed on the standard output.

> **Warning:** The output path is overwritten.

The output file is in JSON Lines format. Each line contains a JSON object with a response from Zyte API.

By default, `zyte-api` uses multiple concurrent connections for *performance reasons* and, as a result, the order of responses will probably not match the order of the source requests from the *input file*. If you need to match the output results to the input requests, the best way is to use echoData. By default, `zyte-api` fills echoData with the input URL.

4.3 Optimization

By default, `zyte-api` uses 20 concurrent connections for requests. Use the `--n-conn` switch to change that:

```
zyte-api --n-conn 40 ...
```

The `--shuffle` option can be useful if you target multiple websites and your *input file* is sorted by website, to randomize the request order and hence distribute the load somewhat evenly:

```
zyte-api urls.txt --shuffle ...
```

For guidelines on how to choose the optimal `--n-conn` value for you, and other optimization tips, see Optimizing Zyte API usage.

4.4 Errors and retries

`zyte-api` automatically handles retries for rate-limiting and unsuccessful responses, as well as network errors, following the *default retry policy*.

Use `--dont-retry-errors` to disable the retrying of error responses, and retrying only rate-limiting responses:

```
zyte-api --dont-retry-errors ...
```

By default, errors are only logged in the standard error output (`stderr`). If you want to include error responses in the output file, use `--store-errors`:

```
zyte-api --store-errors ...
```

See also:

CLI reference

CHAPTER

FIVE

PYTHON CLIENT LIBRARY

Once you have *installed python-zyte-api* and *configured your API key*, you can use one of its APIs from Python code:

- The *sync API* can be used to build simple, proof-of-concept or debugging Python scripts.
- The *async API* can be used from coroutines, and is meant for production usage, as well as for asyncio environments like Jupyter notebooks.

5.1 Sync API

Create a *ZyteAPI* object, and use its *get()* method to perform a single request:

```
from zyte_api import ZyteAPI

client = ZyteAPI()
result = client.get({"url": "https://toscrape.com", "httpResponseBody": True})
```

To perform multiple requests, use a *session()* for better performance, and use *iter()* to send multiple requests in parallel:

```
from zyte_api import ZyteAPI, RequestError

client = ZyteAPI()
with client.session() as session:
    queries = [
        {"url": "https://toscrape.com", "httpResponseBody": True},
        {"url": "https://books.toscrape.com", "httpResponseBody": True},
    ]
    for result_or_exception in session.iter(queries):
        if isinstance(result_or_exception, dict):
            ...
        elif isinstance(result_or_exception, RequestError):
            ...
        else:
            assert isinstance(result_or_exception, Exception)
            ...
```

Tip: *iter()* yields results as they come, not necessarily in their original order. Use echoData to track the source request.

5.2 Async API

Create an *AsyncZyteAPI* object, and use its *get()* method to perform a single request:

```
import asyncio

from zyte_api import AsyncZyteAPI


async def main():
    client = AsyncZyteAPI()
    result = await client.get({"url": "https://toscrape.com", "httpResponseBody": True})


asyncio.run(main())
```

To perform multiple requests, use a `session()` for better performance, and use *iter()* to send multiple requests in parallel:

```
import asyncio

from zyte_api import ZyteAPI, RequestError


async def main():
    client = ZyteAPI()
    async with client.session() as session:
        queries = [
            {"url": "https://toscrape.com", "httpResponseBody": True},
            {"url": "https://books.toscrape.com", "httpResponseBody": True},
        ]
        for future in session.iter(queries):
            try:
                result = await future
            except RequestError as e:
                ...
            except Exception as e:
                ...


asyncio.run(main())
```

Tip: *iter()* yields results as they come, not necessarily in their original order. Use echoData to track the source request.

5.3 Optimization

ZyteAPI and *AsyncZyteAPI* use 15 concurrent connections by default.

To change that, use the `n_conn` parameter when creating your client object:

```
client = ZyteAPI(n_conn=30)
```

The number of concurrent connections if enforced across all method calls, including different sessions of the same client.

For guidelines on how to choose the optimal value for you, and other optimization tips, see Optimizing Zyte API usage.

5.4 Errors and retries

Methods of *ZyteAPI* and *AsyncZyteAPI* automatically handle retries for rate-limiting and unsuccessful responses, as well as network errors.

The default retry policy, *zyte_api_retrying*, does the following:

- Retries rate-limiting responses forever.
- Retries unsuccessful responses up to 3 times.
- Retries network errors for up to 15 minutes.

All retries are done with an exponential backoff algorithm.

To customize the retry policy, create your own `AsyncRetrying` object, e.g. using a custom subclass of *RetryFactory*, and pass it when creating your client object:

```
client = ZyteAPI(retrying=custom_retry_policy)
```

When retries are exceeded for a given request, an exception is raised. Except for the *iter()* method of the *sync API*, which yields exceptions instead of raising them, to prevent exceptions from interrupting the entire iteration.

The type of exception depends on the issue that caused the final request attempt to fail. Unsuccessful responses trigger a *RequestError* and network errors trigger aiohttp exceptions. Other exceptions could be raised; for example, from a custom retry policy.

See also:

API reference

CHAPTER

SIX

CLI REFERENCE

6.1 zyte-api

Send Zyte API requests.

```
usage: zyte-api [-h] [--intype {txt,jl}] [--limit LIMIT] [--output OUTPUT]
                [--n-conn N_CONN] [--api-key API_KEY] [--api-url API_URL]
                [--loglevel {DEBUG,INFO,WARNING,ERROR}] [--shuffle]
                [--dont-retry-errors] [--store-errors]
                INPUT
```

6.1.1 Positional Arguments

INPUT Path to an input file (see 'Command-line client > Input file' in the docs for details).

6.1.2 Named Arguments

--intype Possible choices: txt, jl

Type of the input file, either 'txt' (plain text) or 'jl' (JSON Lines).

If not specified, the input type is guessed based on the input file extension ('.jl', '.jsonl', or '.txt'), or in its content, with 'txt' as fallback.

--limit Maximum number of requests to send.

--output, -o Path for the output file. Results are written into the output file in JSON Lines format.

If not specified, results are printed to the standard output.

--n-conn Number of concurrent connections to use (default: 20).

--api-key Zyte API key.

--api-url Zyte API endpoint (default: "https://api.zyte.com/v1/").

--loglevel, -L Possible choices: DEBUG, INFO, WARNING, ERROR

Log level (default: "INFO").

--shuffle Shuffle request order.

--dont-retry-errors Do not retry unsuccessful responses and network errors, only rate-limiting responses.

--store-errors Store error responses in the output file.

If omitted, only successful responses are stored.

CHAPTER

SEVEN

API REFERENCE

7.1 Sync API

class ZyteAPI(*, *api_key=None*, *api_url='https://api.zyte.com/v1/'*, *n_conn=15*, *retrying: AsyncRetrying | None = None*, *user_agent: str | None = None*)

Synchronous Zyte API client.

api_key is your Zyte API key. If not specified, it is read from the `ZYTE_API_KEY` environment variable. See *API key*.

api_url is the Zyte API base URL.

n_conn is the maximum number of concurrent requests to use. See *Optimization*.

retrying is the retry policy for requests. Defaults to *zyte_api_retrying*.

user_agent is the user agent string reported to Zyte API. Defaults to `python-zyte-api/<VERSION>`.

Tip: To change the `User-Agent` header sent to a target website, use customHttpRequestHeaders instead.

---

get(*query: dict*, *, *endpoint: str = 'extract'*, *session: ClientSession | None = None*, *handle_retries: bool = True*, *retrying: AsyncRetrying | None = None*) → dict

Send *query* to Zyte API and return the result.

endpoint is the Zyte API endpoint path relative to the client object *api_url*.

session is the network session to use. Consider using *session()* instead of this parameter.

handle_retries determines whether or not a *retry policy* should be used.

retrying is the *retry policy* to use, provided *handle_retries* is `True`. If not specified, the *default retry policy* is used.

iter(*queries: List[dict]*, *, *endpoint: str = 'extract'*, *session: ClientSession | None = None*, *handle_retries: bool = True*, *retrying: AsyncRetrying | None = None*) → Generator[dict | Exception, None, None]

Send multiple *queries* to Zyte API in parallel and iterate over their results as they come.

The number of *queries* can exceed the *n_conn* parameter set on the client object. Extra queries will be queued, there will be only up to *n_conn* requests being processed in parallel at a time.

Results may come an a different order from the original list of *queries*. You can use echoData to attach metadata to queries, and later use that metadata to restore their original order.

When exceptions occur, they are yielded, not raised.

The remaining parameters work the same as in *get()*.

session(***kwargs*)

Context manager to create a contextual session.

A contextual session is an object that has the same API as the client object, except:

- `get()` and `iter()` do not have a *session* parameter, the contextual session creates an `aiohttp.ClientSession` object and passes it to `get()` and `iter()` automatically.
- It does not have a `session()` method.

Using the same `aiohttp.ClientSession` object for all Zyte API requests improves performance by keeping a pool of reusable connections to Zyte API.

The `aiohttp.ClientSession` object is created with sane defaults for Zyte API, but you can use *kwargs* to pass additional parameters to `aiohttp.ClientSession` and even override those sane defaults.

7.2 Async API

class AsyncZyteAPI(**, api_key=None, api_url='https://api.zyte.com/v1/', n_conn=15, retrying: AsyncRetrying | None = None, user_agent: str | None = None*)

Asynchronous Zyte API client.

Parameters work the same as for `ZyteAPI`.

async get(*query: dict, *, endpoint: str = 'extract', session=None, handle_retries=True, retrying: AsyncRetrying | None = None*) → Future

Asynchronous equivalent to `ZyteAPI.get()`.

iter(*queries: List[dict], *, endpoint: str = 'extract', session: ClientSession | None = None, handle_retries=True, retrying: AsyncRetrying | None = None*) → Iterator[Future]

Asynchronous equivalent to `ZyteAPI.iter()`.

Note: Yielded futures, when awaited, do raise their exceptions, instead of only returning them.

7.3 Retries

zyte_api_retrying

Default retry policy.

class RetryFactory

Factory class that builds the `tenacity.AsyncRetrying` object that defines the *default retry policy*.

To create a custom retry policy, you can subclass this factory class, modify it as needed, and then call `build()` on your subclass to get the corresponding `tenacity.AsyncRetrying` object.

For example, to increase the maximum number of attempts for temporary download errors from 4 (i.e. 3 retries) to 10 (i.e. 9 retries):

```
from tenacity import stop_after_attempt
from zyte_api import RetryFactory
```

(continues on next page)

(continued from previous page)

```
class CustomRetryFactory(RetryFactory):
    temporary_download_error_stop = stop_after_attempt(10)


CUSTOM_RETRY_POLICY = CustomRetryFactory().build()
```

To retry permanent download errors, treating them the same as temporary download errors:

```
from tenacity import RetryCallState, retry_if_exception, stop_after_attempt
from zyte_api import RequestError, RetryFactory


def is_permanent_download_error(exc: BaseException) -> bool:
    return isinstance(exc, RequestError) and exc.status == 521


class CustomRetryFactory(RetryFactory):

    retry_condition = RetryFactory.retry_condition | retry_if_exception(
        is_permanent_download_error
    )

    def wait(self, retry_state: RetryCallState) -> float:
        if is_permanent_download_error(retry_state.outcome.exception()):
            return self.temporary_download_error_wait(retry_state=retry_state)
        return super().wait(retry_state)

    def stop(self, retry_state: RetryCallState) -> bool:
        if is_permanent_download_error(retry_state.outcome.exception()):
            return self.temporary_download_error_stop(retry_state)
        return super().stop(retry_state)


CUSTOM_RETRY_POLICY = CustomRetryFactory().build()
```

7.4 Errors

exception RequestError(**args*, ***kwargs*)

Exception raised upon receiving a rate-limiting or unsuccessful response from Zyte API.

property parsed

Response as a *ParsedError* object.

request_id: str | None

Request ID.

response_content: bytes | None

Response body.

class ParsedError(*response_body: bytes*, *data: dict | None*, *parse_error: str | None*)

Parsed error response body from Zyte API.

data: *dict* | None

JSON-decoded response body.

If None, *parse_error* indicates the reason.

classmethod from_body(*response_body: bytes*) → *ParsedError*

Return a *ParsedError* object built out of the specified error response body.

parse_error: *str* | None

If *data* is None, this indicates whether the reason is that *response_body* is not valid JSON ("bad_json") or that it is not a JSON object ("bad_format").

response_body: *bytes*

Raw response body from Zyte API.

property type: *str* | None

ID of the error type, e.g. "/limits/over-user-limit" or "/download/temporary-error".

CHAPTER

EIGHT

CONTRIBUTING

python-zyte-api is an open-source project. Your contribution is very welcome!

8.1 Issue Tracker

If you have a bug report, a new feature proposal or simply would like to make a question, please check our issue tracker on Github: https://github.com/zytedata/python-zyte-api/issues

8.2 Source code

Our source code is hosted on Github: https://github.com/zytedata/python-zyte-api

Before opening a pull request, it might be worth checking current and previous issues. Some code changes might also require some discussion before being accepted so it might be worth opening a new issue before implementing huge or breaking changes.

8.3 Testing

We use tox to run tests with different Python versions:

```
tox
```

The command above also runs type checks; we use mypy.

CHAPTER

NINE

CHANGES

9.1 0.5.0 (2024-04-05)

- Removed Python 3.7 support.
- Added *ZyteAPI* and *AsyncZyteAPI* to provide both sync and async Python interfaces with a cleaner API.
- Deprecated `zyte_api.aio`:
  - Replace `zyte_api.aio.client.AsyncClient` with the new *AsyncZyteAPI* class.
  - Replace `zyte_api.aio.client.create_session` with the new `AsyncZyteAPI.session` method.
  - Import `zyte_api.aio.errors.RequestError`, `zyte_api.aio.retry.RetryFactory` and `zyte_api.aio.retry.zyte_api_retrying` directly from `zyte_api` now.
- When using the command-line interface, you can now use `--store-errors` to have error responses be stored alongside successful responses.
- Improved the documentation.

9.2 0.4.8 (2023-11-02)

- Include the Zyte API request ID value in a new `.request_id` attribute in `zyte_api.aio.errors.RequestError`.

9.3 0.4.7 (2023-09-26)

- `AsyncClient` now lets you set a custom user agent to send to Zyte API.

9.4 0.4.6 (2023-09-26)

- Increased the client timeout to match the server's.
- Mentioned the `api_key` parameter of `AsyncClient` in the docs example.

9.5 0.4.5 (2023-01-03)

- w3lib >= 2.1.1 is required in install_requires, to ensure that URLs are escaped properly.
- unnecessary `requests` library is removed from install_requires
- fixed tox 4 support

9.6 0.4.4 (2022-12-01)

- Fixed an issue with submitting URLs which contain unescaped symbols
- New "retrying" argument for AsyncClient.__init__, which allows to set custom retrying policy for the client
- `--dont-retry-errors` argument in the CLI tool

9.7 0.4.3 (2022-11-10)

- Connections are no longer reused between requests. This reduces the amount of `ServerDisconnectedError` exceptions.

9.8 0.4.2 (2022-10-28)

- Bump minimum `aiohttp` version to 3.8.0, as earlier versions don't support brotli decompression of responses
- Declared Python 3.11 support

9.9 0.4.1 (2022-10-16)

- Network errors, like server timeouts or disconnections, are now retried for up to 15 minutes, instead of 5 minutes.

9.10 0.4.0 (2022-09-20)

- Require to install `Brotli` as a dependency. This changes the requests to have `Accept-Encoding: br` and automatically decompress brotli responses.

9.11 0.3.0 (2022-07-29)

Internal AggStats class is cleaned up:

- `AggStats.n_extracted_queries` attribute is removed, as it was a duplicate of `AggStats.n_results`
- `AggStats.n_results` is renamed to `AggStats.n_success`
- `AggStats.n_input_queries` is removed as redundant and misleading; AggStats got a new `AggStats.n_processed` property instead.

This change is backwards incompatible if you used stats directly.

9.12 0.2.1 (2022-07-29)

- `aiohttp.client_exceptions.ClientConnectorError` is now treated as a network error and retried accordingly.
- Removed the unused `zyte_api.sync` module.

9.13 0.2.0 (2022-07-14)

- Temporary download errors are now retried 3 times by default. They were not retried in previous releases.

9.14 0.1.4 (2022-05-21)

This release contains usability improvements to the command-line script:

- Instead of `python -m zyte_api` you can now run it as `zyte-api`;
- the type of the input file (`--intype` argument) is guessed now, based on file extension and content; .jl, .jsonl and .txt files are supported.

9.15 0.1.3 (2022-02-03)

- Minor documenation fix
- Remove support for Python 3.6
- Added support for Python 3.10

9.16 0.1.2 (2021-11-10)

- Default timeouts changed

9.17 0.1.1 (2021-11-01)

- CHANGES.rst updated properly

9.18 0.1.0 (2021-11-01)

- Initial release.

PYTHON MODULE INDEX

Z

INDEX