qetch Documentation

Release 0.0.0

Stephen Bunn

Feb 12, 2018

Contents

CHAPTER 1

Documentation

1.1 Getting Started

> This framework is my attempt at modernizing the type of content extraction that youtube-dl performs. It's called "Qetch" because I couldn't think of anything better...

I started this because I needed a way of extracting and downloading raw content from just a user dropping in a url. The issue with current solutions is that they have an unintuitive API and an overcomplicated implementation (no offense intended, I really appreciate the work that went into the current solutions).

But I'm a stickler and wanted a cleaner more modular way of building extractors and quicker downloaders; also something that doesn't strive to be "Pure Python" **because pure Python isn't real Python**.

Note: Qetch requires Python 3.6+. Because of support dropping for Python 2.7 and so many various improvments from 3.5, it was decided unanimously (meaning just me) that this project will only support 3.6+.

1.1.1 Installation

Since Qetch is in pre-development/proof-of-concept stages, it is not yet on PyPi. You can install Qetch by cloning the repository at stephen-bunn/qetch and installing the dependencies.

```
git clone https://github.com/stephen-bunn/qetch.git
cd ./qetch
pip install -r ./requirements.txt
```

Pipenv is also an option! *If you don't yet know about Pipenv, you should definitely start using it!*

1.1.2 Basic Usage

The quickest way to utilize Qetch is to just allow Qetch to discover what extractors/downloaders are required for a URL you give it.

```
import os
import qetch

# discover what extractor can handle a URL and initialize it
extractor = qetch.get_extractor(URL, init=True)

# extract the first discovered content
content = next(extractor.extract(URL))[0]

# discover what downloader can handle the extracted content and initialize it
downloader = qetch.get_downloader(content, init=True)

# download the content to a given filepath
downloader.download(content, os.path.expanduser('~/Downloads/downloaded_file'))
```

As shown in the example above, there are several objects that make up Qetch. You can learn more about them in the *Project Structure* documentation and the *Qetch Package* reference.

1.2 Project Structure

I like pictures, so bear with me while I use a couple that some of you might *roll* your eyes at.

Qetch mainly consists of 4 separate components. These are listed in the following sections with a quick and simple description of each one and what it's purpose is.

1.2.1 Content

The `Content` is a simple object which stores all the required information needed to download something.

Most of the attributes in this object is sugar used for better representing the content. The only three that really matter are the `uid`, `extractor`, and `fragments`.

The `uid` is simply a unique identifier for the content. The `extractor` is just a reference to the `BaseExtractor` subclass that was used to extract the content.

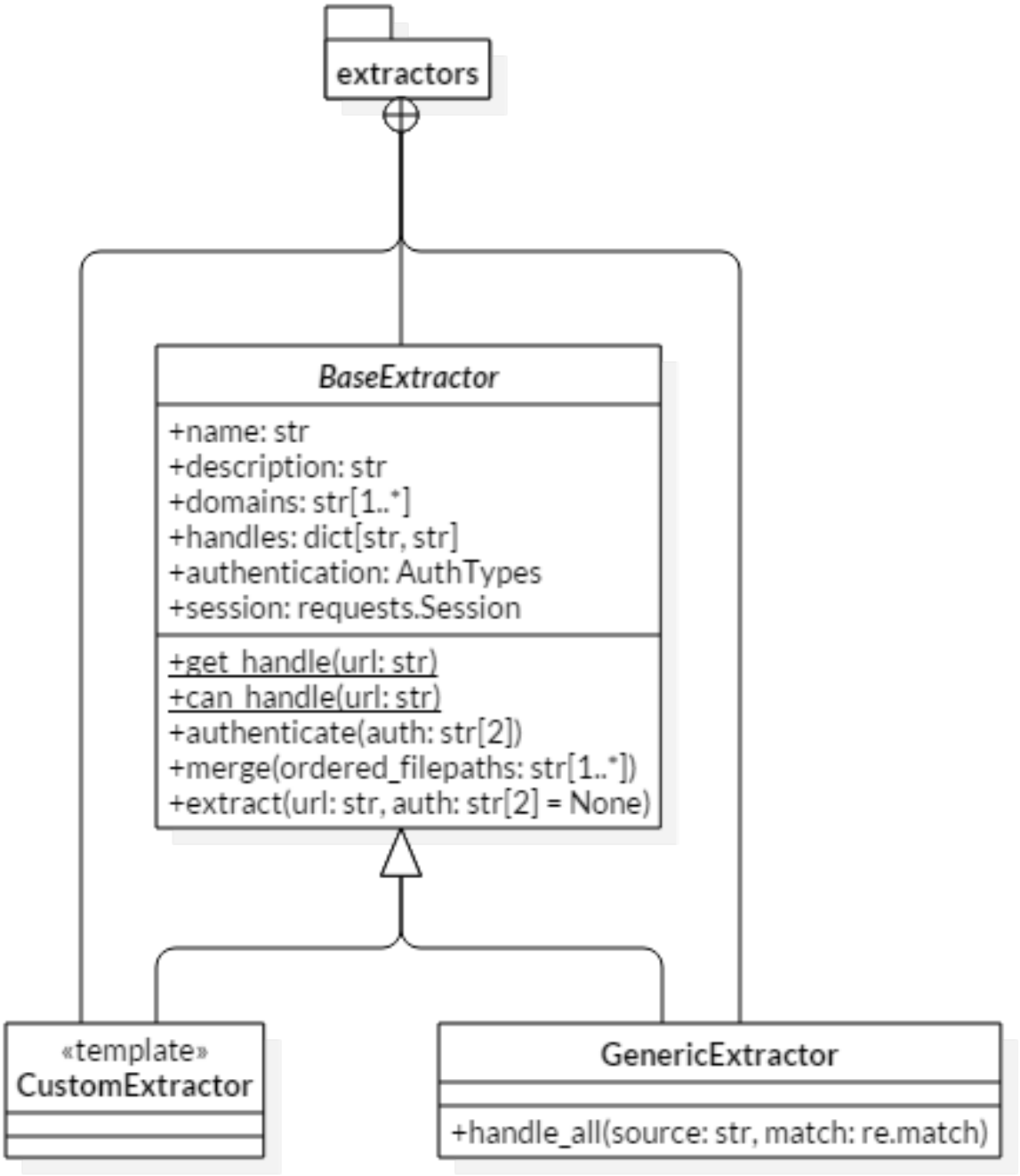
The actual urls which need to be downloaded to form the full content are items in the `fragments` list. In most cases the length of this list is 1 (because the raw content is not hosted as segments). *However*, for sites that do stream segments of media, it most likely means that the length of the `fragments` list will be more than one.

Because of these fragments, it is necessary to calculate the size of the full content. This is performed through the `get_size()` method.

1.2.2 Extractors

All extractors are subclasses of `BaseExtractor`, and provide special logic to handle the extraction of certain URLs. This usually means that a handled domain will have an extractor to deal with that domain's URLs.

This is essentially the core of the project since it requires contributions from the community to grow and include the ability for difference domains to have their content extracted. *If you have the logic to create an extractor for a domain that is not yet handled*, please make a pull request following *our guidelines*.



The overall purpose of extractors is to yield one or more list of `Content` instances that can be downloaded from a given URL.

The reason extractors yeild **lists** is because a site might host various levels of quality for some content that is essentially the same. This allows the user to choose which quality of content they want from the available qualities found at the given URL.

Authentication

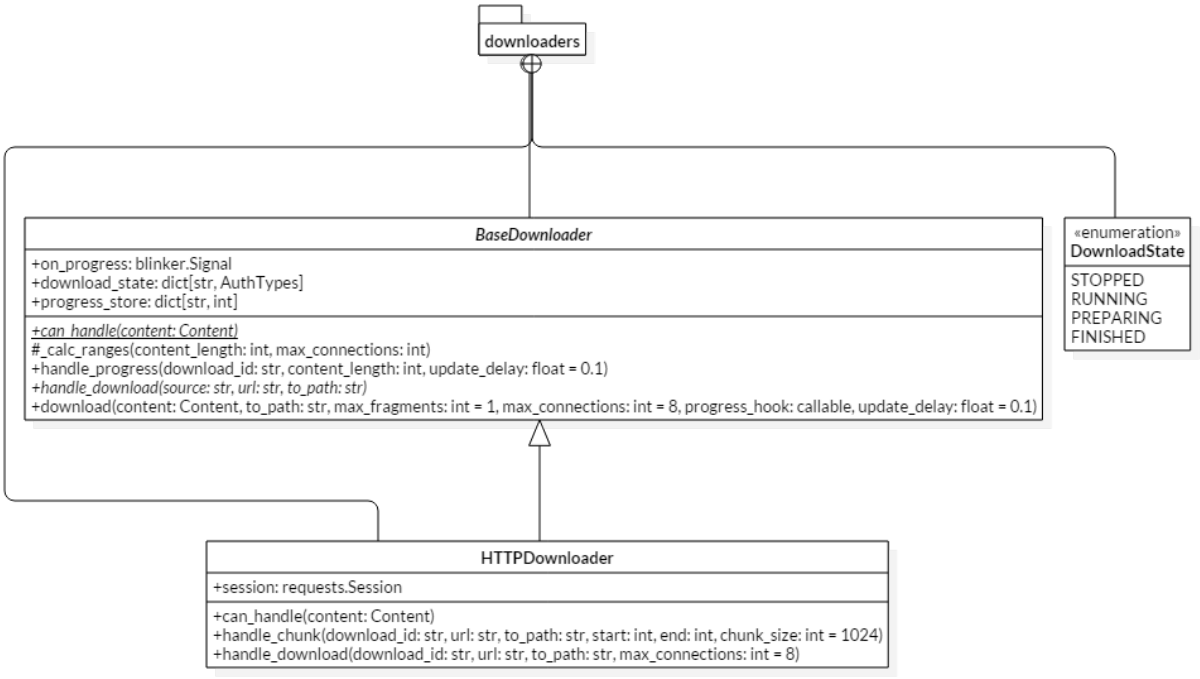
Sometimes there is no **good** way to retrieve the necessary information for a certain URL due to authentication requirements by the site itself. In order to handle this, the *AuthRegistry* was created to help extractors say what kind of authentication is required before they can extract content.

An extractor specifies the necessary *AuthTypes* literal in the `authentication` property. It applies any authentication in the *authenticate()* method before extraction.

The *AuthRegistry* is a borg dictionary which stores authentication information across all instances of the registry.

1.2.3 Downloaders

Downloaders are similaraly structured to extractors, but their purpose is to download a single *Content* instance to a specified filepath. They all extend *BaseDownloader* and provide progress hooks to the download process.

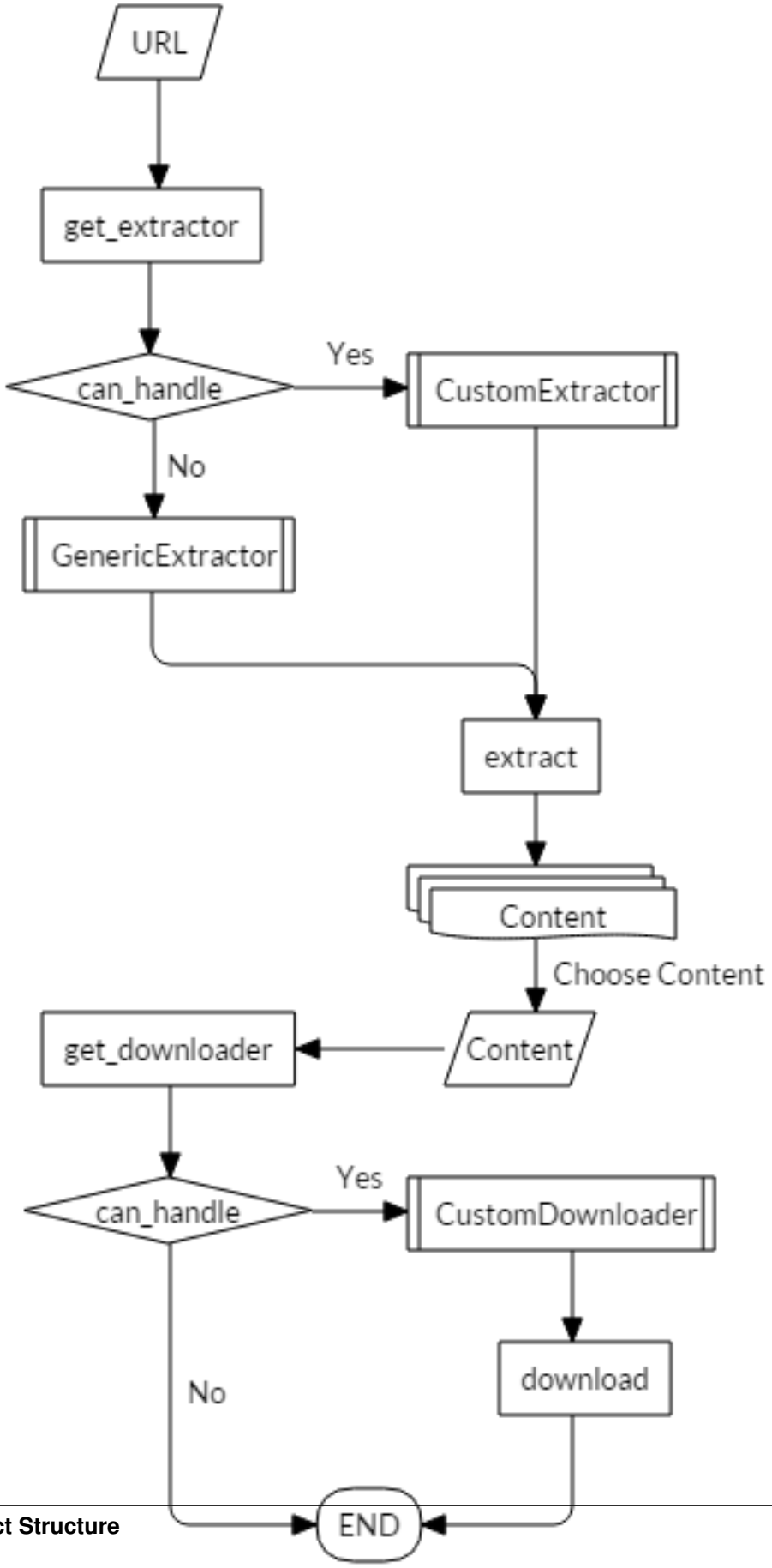


All of the downloaders *should* support multi-threaded/multi-connection downloads similar to the *HTTPDownloader*.

The optional merging of fragments is handled by the extractor itself in the *merge()* (since downloader's are abstracted away from extraction). If the extractor does require downloaded fragment merging, then it is necessary for the extractor to override that method.

1.2.4 Basic Overview

Just to visualize the overall process involved in downloading a URL from start to finish, here is a simple flow chart describing the process.


URL
get_extractor
can_handle
Yes
CustomExtractor
No
GenericExtractor
extract
Content
Choose Content
get_downloader
Content
can_handle
Yes
CustomDownloader
download
No
END

1.3 Changelog

All notable changes to qetch will be documented in this file.
The format is based on Keep a Changelog and this project adheres to Semantic Versioning.

1.3.1 *unreleased*

- added basic project structure migration from previous proof-of-concepts
- enhanced documentation to make it readable
- fixed multi-connection threaded progress reporting
- removed broken WIP extractors from previous repositories

CHAPTER 2

Reference

2.1 Contributing

When contributing to this repository, please first discuss the change you wish to make via an issue to the owners of this repository before submitting a pull request.

Important: We have an enforced style guide and a code of conduct. Please follow them in all your interactions with this project.

2.1.1 Style Guide

- We stictly follow PEP8 and utilize Sphinx docstrings on **all** classes and functions.
- **We employee flake8 as our linter with exceptions to the following rules:**
  - D203
  - F401
  - E123
- Linting and test environments are configured via `tox.ini`.
- An `.editorconfig` file is included in this repository which dictates whitespace, indentation, and file encoding rules.
- Although `requirements.txt` and `requirements_dev.txt` do exist, Pipenv is utilized as the primary virtual environment and package manager for this project.
- We strictly utilize Semantic Versioning as our version specification.

2.1.2 Issues

Issues should follow the included `ISSUE_TEMPLATE` found in `.github/ISSUE_TEMPLATE.md`.

- **Issues should contain the following sections:**
  - Expected Behavior
  - Current Behavior
  - Possible Solution
  - Steps to Reproduce (for bugs)
  - Context
  - Your Environment

These sections help the developers greatly by providing a large understanding of the context of the bug or requested feature without having to launch a full fleged discussion inside of the issue.

2.1.3 Pull Requests

Pull requests should follow the included `PULL_REQUEST_TEMPLATE` found in `.github/PULL_REQUEST_TEMPLATE.md`.

- Pull requests should always be from a **topic/feature/bugfix** (left side) branch. *Pull requests from master branches will not be merged.*
- Pull requests should not fail our requested style guidelines or linting checks.

2.1.4 Code of Conduct

Our code of conduct is taken directly from the Contributor Covenant since it directly hits all of the points we find necessary to address.

Our Pledge

In the interest of fostering an open and welcoming environment, we as contributors and maintainers pledge to making participation in our project and our community a harassment-free experience for everyone, regardless of age, body size, disability, ethnicity, gender identity and expression, level of experience, education, socio-economic status, nationality, personal appearance, race, religion, or sexual identity and orientation.

Our Standards

Examples of behavior that contributes to creating a positive environment include:

- Using welcoming and inclusive language
- Being respectful of differing viewpoints and experiences
- Gracefully accepting constructive criticism
- Focusing on what is best for the community
- Showing empathy towards other community members

Examples of unacceptable behavior by participants include:

- The use of sexualized language or imagery and unwelcome sexual attention or advances
- Trolling, insulting/derogatory comments, and personal or political attacks
- Public or private harassment
- Publishing others' private information, such as a physical or electronic address, without explicit permission
- Other conduct which could reasonably be considered inappropriate in a professional setting

Our Responsibilities

Project maintainers are responsible for clarifying the standards of acceptable behavior and are expected to take appropriate and fair corrective action in response to any instances of unacceptable behavior.

Project maintainers have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, or to ban temporarily or permanently any contributor for other behaviors that they deem inappropriate, threatening, offensive, or harmful.

Scope

This Code of Conduct applies both within project spaces and in public spaces when an individual is representing the project or its community. Examples of representing a project or community include using an official project e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event. Representation of a project may be further defined and clarified by project maintainers.

Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported by contacting the project team at stephen@bunn.io. All complaints will be reviewed and investigated and will result in a response that is deemed necessary and appropriate to the circumstances. The project team is obligated to maintain confidentiality with regard to the reporter of an incident. Further details of specific enforcement policies may be posted separately.

Project maintainers who do not follow or enforce the Code of Conduct in good faith may face temporary or permanent repercussions as determined by other members of the project's leadership.

Attribution

This Code of Conduct is adapted from the Contributor Covenant, version 1.4, available at https://www.contributor-covenant.org/version/1/4/code-of-conduct.html

2.2 Qetch Package

This is the base qetch package.

qetch.**get_downloader**(*content*, *init=False*, **args*, ***kwargs*)

Gets the first downloader that can handle a given content.

Parameters

- **content** (Content) – The content that needs to be downloaded
- **init** (*bool, optional*) – If True initializes the class, otherwise returns the class

Returns

The downloader that can handle the content.

Return type *downloaders._common.BaseDownloader*

Examples

Basic usage...

```
>>> import qetch
>>> content = next(qetch.get_extractor(GFYCAT_URL, init=True)
...     .extract(GFYCAT_URL))[0]
>>> downloader = qetch.get_downloader(content, init=True)
>>> print(downloader)
<HTTPDownloader at 0xABCDEF1234567890>
```

qetch.**get_extractor**(*url*, *init=False*, **args*, ***kwargs*)

Gets the first extractor that can handle a given url.

Parameters

- **url** (*str*) – The url that needs to be extracted
- **init** (*bool, optional*) – If True initializes the class, otherwise returns the class

Returns

The extractor that can handle the url.

Return type *extractors._common.BaseExtractor*

Examples

Basic usage...

```
>>> import qetch
>>> extractor = qetch.get_extractor(GFYCAT_URL, init=True)
>>> print(extractor)
<GfycatExtractor "gfycat">
```

2.2.1 qetch.auth

class qetch.auth.**AuthRegistry**(***kwargs*)

Bases: dict

Custom borg style registry dictionary.

This registry dictionary utilizes the borg design pattern and maintains the same state across multiple instances. This means that multiple instances of this object can exist, but the values between them will stay syncronized.

Examples

Basic usage...

```
>>> from qetch.auth import (AuthRegistry,)
>>> from qetch.extractors import (GfycatExtractor,)
>>> registry_1 = AuthRegistry()
>>> registry_1[GfycatExtractor.name] = ('KEY', 'SECRET',)
>>> print(registry_1[GfycatExtractor.name])
('KEY', 'SECRET')
>>> registry_2 = AuthRegistry()
>>> print(registry_2[GfycatExtractor.name])
('KEY', 'SECRET')
>>> registry_1[GfycatExtractor.name] = ('USERNAME', 'PASSWORD',)
>>> print(registry_2[GfycatExtractor.name])
('USERNAME', 'PASSWORD')
```

clear() → None. Remove all items from D.

copy() → a shallow copy of D

items() → a set-like object providing a view on D's items

keys() → a set-like object providing a view on D's keys

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised

update([*E*], ***F*) → None. Update D from dict/iterable E and F.
If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

values() → an object providing a view on D's values

class qetch.auth.**AuthTypes**
Bases: enum.Enum

An enumeration of available authentication types.

Values:

- NONE: No authentication required
- BASIC: Basic (username, password) authentication required
- OAUTH: Standard oauth (key, secret) authentication required

2.2.2 qetch.content

This is the base content instance which is used to normalize hosted media for use between `extractors` and `downloaders`. The most important attributes of this object are the following:

- ***uid*: The unique id that identifies the content** (*even unique between levels of quality*).
- *source*: The url that was given to the extractor for extracting.
- ***fragments*: A list of urls where the raw content can be retrieved from** (*is a list in case that content is fragmented/segmented*).
- *quality*: A float value between 0 and 1, 1 being the best quality format.

class qetch.content.**Content**(*uid*, *source*, *fragments*, *extractor*, *extension=None*, *title=None*, *description=None*, *quality=0.0*, *uploaded_by=None*, *uploaded_date=None*, *metadata={}*)

Bases: object

The resulting content instance yielded by extractors.

get_size()

Returns the sum of the length of the fragments.

Returns The sum of the length of the fragments.

Return type int

description

The description of the content.

Returns The description of the content.

Return type str

extension

The extension of the resulting content.

Returns The extension for the resulting content.

Return type str

extractor

The extractor which discovered the content.

Returns The extractor which discovered the content.

Return type *BaseExtractor*

fragments

A list of urls which represent the raw content.

Returns A list of urls which represent the raw content.

Return type list[str]

metadata

Any metadata for the current content.

Returns Any metadata for the current content.

Return type dict[str,...]

quality

The contextual quality for the current content.

Returns The contextual quality for the current content.

Return type float

source

The given source url from where the content came from.

Returns The given source url from where the content came from.

Return type furl.furl

title

The title of the content.

Returns The title of the content.

Return type str

uid
The unique id of the discovered content.

Returns The unique id of the discovered content.

Return type str

uploaded_by
A string of the uploader's name.

Returns A string of the uploader's name.

Return type str

uploaded_date
The datetime the content was uploaded.

Returns The datetime the content was uploaded.

Return type datetime.datetime

2.2.3 qetch.extractors

Below are a list of the currently included extractors which all should extend *BaseExtractor*. The purpose of extractors is to take a url and yield lists of similar content instances.

This allows content with various levels of quality to have a relationship with eachother. For example, gfycat.com hosts various levels and formats of some media (mp4, webm, webp, gif, etc. . .). When extracting the content for a gfycat url, an extractor will yield a list containing different content instances for each of these formats and different *quality* values. This allows the developer to *hopefully* correctly choose the desired content for a list of content extracted for a single resource.

BaseExtractor

class qetch.extractors._common.**BaseExtractor**
Bases: abc.ABC

The base extractor. *All extractors should extend this.*

authenticate(*auth*)
Handles authenticating the extractor if necessary.

Parameters **auth** (*tuple[str, str]*) – The authentication tuple is available.

classmethod can_handle(*url*)
Determines if an extractor can handle a url.

Parameters **url** (*str*) – The url to check

Returns True if the extractor can handle, otherwise False

Return type bool

extract(*url*, *auth=None*)
Extracts lists of content from a url.

Note: When an extractor can handle a url with a given `{handle_name: regex}` dictionary, the *extract()* method assumes that a method `handle_{handle_name}` exists to handle that specific url.

If an appropriately named method does not exist, a `NotImplementedError` is raised.

Parameters

- **url** (*str*) – The url to extract content from.
- **auth** (*tuple[str, str], optional*) – The auth tuple if available.

Raises `NotImplementedError` – If a given `handle_{handle_name}` method does not exist.

Yields *list[Content]* – A list of similar content of different qualities

Examples

Basic usage where `GFYCAT_ID` is the id determined from `GFYCAT_URL`.

```
>>> from qetch.extractors import (GfycatExtractor,)
>>> for content_list in GfycatExtractor().extract(GFYCAT_URL):
...     for content in content_list:
...         print(content)
<Content (1.0) "gfycat-GFYCAT_ID-mp4Url">
<Content (0.5) "gfycat-GFYCAT_ID-webmUrl">
<Content (0.0) "gfycat-GFYCAT_ID-webpUrl">
<Content (0.0) "gfycat-GFYCAT_ID-mobileUrl">
<Content (0.0) "gfycat-GFYCAT_ID-mobilePosterUrl">
<Content (0.0) "gfycat-GFYCAT_ID-posterUrl">
<Content (0.0) "gfycat-GFYCAT_ID-thumb360Url">
<Content (0.0) "gfycat-GFYCAT_ID-thumb360PosterUrl">
<Content (0.0) "gfycat-GFYCAT_ID-thumb100PosterUrl">
<Content (0.0) "gfycat-GFYCAT_ID-max5mbGif">
<Content (0.0) "gfycat-GFYCAT_ID-max2mbGif">
<Content (0.0) "gfycat-GFYCAT_ID-mjpgUrl">
<Content (0.0) "gfycat-GFYCAT_ID-miniUrl">
<Content (0.0) "gfycat-GFYCAT_ID-miniPosterUrl">
<Content (0.25) "gfycat-GFYCAT_ID-gifUrl">
```

Return type `Generator`[`List`[`Any`], `None`, `None`]

classmethod get_handle(*url*)

Gets the handle match for a given url.

Parameters **url** (*str*) – The url to get the handle match for.

Returns A tuple of handle and the match for the url.

Return type tuple[str, Match]

merge(*ordered_filepaths*)

Handles merging downloaded fragments into a resulting file.

Parameters **ordered_filepaths** (*list[str]*) – The list of ordered filepaths to downloaded fragments.

Returns The resulting merged file's filepath.

Return type str

session
The default session for the extractor.

Returns The default session for the extractor.

Return type requests.Session

gfycat

class qetch.extractors.gfycat.**GfycatExtractor**
Bases: *qetch.extractors._common.BaseExtractor*

The extractor for links to media from `gfycat.com`.

handle_basic(*source*, *match*)
Handles `basic` links to gfycat media.

Parameters

- **source** (*str*) – The source url
- **match** (*Match*) – The source match regex

Yields *list[Content]* – A list of various levels of quality content for the same source url

Return type Generator[List[*Content*], None, None]

handle_raw(*source*, *match*)
Handles `raw` links to gfycat media.

Parameters

- **source** (*str*) – The source url
- **match** (*Match*) – The source match regex

Yields *list[Content]* – A list of various levels of quality content for the same source url

Return type Generator[List[*Content*], None, None]

authentication = None

description = 'Site which hosts short high-quality video for sharing.'

domains = ['gfycat.com']

handles = {'basic': '^https?://(?:www\\.)?gfycat\\.com/(?:gifs/detail/)?(?P<id>[a-zA-

name = 'gfycat'

imgur

class qetch.extractors.imgur.**ImgurExtractor**
Bases: *qetch.extractors._common.BaseExtractor*

The extractor for links to media from `imgur.com`.

authenticate(*auth*)
Handles authenticating the extractor if necessary.

Parameters **auth** (*tuple[str, str]*) – The authentication tuple is available.

handle_album(*source*, *match*)
Handles `album` links to imgur media.

Parameters

- **source** (*str*) – The source url
- **match** (*Match*) – The source match regex

Yields *list[Content]* – A list of various levels of quality content for the same source url

Return type Generator[List[*Content*], None, None]

handle_basic(*source*, *match*)
Handles `basic` links to imgur media.

Parameters

- **source** (*str*) – The source url
- **match** (*Match*) – The source match regex

Yields *list[Content]* – A list of various levels of quality content for the same source url

Return type Generator[List[*Content*], None, None]

handle_raw(*source*, *match*)
Handles `raw` links to imgur media.

Parameters

- **source** (*str*) – The source url
- **match** (*Match*) – The source match regex

Yields *list[Content]* – A list of various levels of quality content for the same source url

Return type Generator[List[*Content*], None, None]

authentication = ('KEY', 'SECRET')

description = 'Dedicated image host originally built for Reddit.'

domains = ['imgur.com', 'i.imgur.com']

handles = {'album': '^https?://(?:www\\.)?imgur\\.com/(?:a|gallery)/(?P<id>[a-zA-Z0-9

name = 'imgur'

fourchan

class qetch.extractors.fourchan.**FourChanExtractor**
Bases: *qetch.extractors._common.BaseExtractor*

The extractor for links to media from `4chan.org`.

handle_raw(*source*, *match*)
Handles `raw` links to 4chan media.

Parameters

- **source** (*str*) – The source url
- **match** (*Match*) – The source match regex

Yields *list[Content]* – A list of various levels of quality content for the same source url

Return type Generator[List[*Content*], None, None]

handle_thread(*source*, *match*)
Handles `thread` links to 4chan media.

Parameters

- **source** (*str*) – The source url
- **match** (*Match*) – The source match regex

Yields *list[Content]* – A list of various levels of quality content for the same source url

Return type Generator[List[*Content*], None, None]

```
authentication = None

description = 'A no-limits and lightly categorized temporary image host.'

domains = ['4chan.org', 'i.4chan.org']

handles = {'raw':  '^https?://(?:www\\.)?i\\.4cdn\\.org/(?P<board>.*)/(?P<id>.*)\\.(?:

name = '4chan'
```

2.2.4 qetch.downloaders

Below are a list of the currently included downloaders which all should extend *BaseDownloader*. The purpose of downloaders is to take an extracted *Content* instance in order to download and merge the fragments resulting in the content being downloaded to a given local system path.

Downloaders should be built to allow parrallel fragment downloading and multiple connection downlaods for each fragment. For example, the *HTTPDownloader* allows both max_fragments and max_connections as parameters to the *download()* method. This will allow max_fragments to be processed at the same time and max_connections to be used for the download of each of those fragments. **This means that up to (max_fragments * max_connections) between your IP and the host may exist at any point during the download.**

It is best to scrutinize this to allow only 10 connections at max, since many hosts will flag/ban IPs using more than 10 connections. By default, max_fragments and max_connections are set to 1 and 8 respectively allowing a maximum of 8 connections from your IP to the host at any point, but only allows 1 fragment to be downloaded at a time.

Downloaders should also support the usage of a progress_hook which is sent updates on the download progress every update_delay seconds. See the example in *download()* for a very simple example.

BaseDownloader

class qetch.downloaders._common.**BaseDownloader**

Bases: abc.ABC

The base abstract base downloader. *All downloaders must extend from this class.*

download(*content*, *to_path*, *max_fragments=1*, *max_connections=8*, *progress_hook=None*, *update_delay=0.1*)

The simplified download method.

Note: The max_fragments and max_connections rules imply that potentially (max_fragments * max_connections) connections from the local system's IP can exist at any time.

Many hosts will flag/ban IPs which utilize more than 10 connections for a single resource. **For this reason**, max_fragments and max_connections are set to 1 and 8 respectively by default.

Parameters

- **content** (*Content*) – The content instance to download.
- **to_path** (*str*) – The path to save the resulting download to.
- **max_fragments** (*int, optional*) – The number of fragments to process in parallel.
- **max_connections** (*int, optional*) – The number of connections to allow for downloading a single fragment.
- **progress_hook** (*callable, optional*) – A progress hook that accepts the arguments `(download_id, current_size, total_size)` for progress updates.
- **update_delay** (*float, optional*) – The frequency (in seconds) where progress updates are sent to the given `progress_hook`.

Returns The downloaded file's local path.

Return type str

Examples

Basic usage where `$HOME` is the home directory of the currently executing user.

```
>>> import os
>>> from qetch.extractors import (GfycatExtractor,)
>>> from qetch.downloaders import (HTTPDownloader,)
>>> content = next(GfycatExtractor().extract(GFYCAT_URL))[0]
>>> saved_to = HTTPDownloader().download(
...     content,
...     os.path.expanduser('~/Downloads/saved_content.mp4'))
>>> print(saved_to)
$HOME/Downloads/saved_content.mp4
```

Similar basic usage, but with a given progress hook sent updates every 0.1 seconds.

```
>>> def progress(download_id, current, total):
...     print(f'{((current / total) * 100.0):6.2f}')
>>> saved_to = HTTPDownloader().download(
...     content,
...     os.path.expanduser('~/Downloads/saved_content.mp4'),
...     progress_hook=progress,
...     update_delay=0.1)
  0.00
  0.00
 23.01
 54.32
 73.09
 90.49
 97.12
100.00
>>> print(saved_to)
$HOME/Downloads/saved_content.mp4
```

handle_progress(*download_id*, *content_length*, *update_delay=0.1*)

The progress reporting handler.

Parameters

- **download_id** (*str*) – The unique id of the download request.
- **content_length** (*int*) – The total size of the downloading content.
- **update_delay** (*float, optional*) – The frequency (in seconds) which progress updates are emitted.

download_state
dict[str,DownloadState] – The download state dictionary.

progress_store
dict[str,int] – The downloaded content size for progress reporting.

class qetch.downloaders._common.**DownloadState**
Bases: enum.Enum

An enum of allowed download states.

Values:

- STOPPED: indicates the download is stopped (error occured)
- RUNNING: indicates the download is running
- PREPARING: indicates the download is starting up
- FINISHED: indicates the download is finished (successfully)

http

class qetch.downloaders.http.**HTTPDownloader**
Bases: *qetch.downloaders._common.BaseDownloader*

The downloader for HTTP served content.

classmethod can_handle(*content*)
Determines if a given content can be handled by this downloader.

Parameters **content** (Content) – The content the check.

Returns True if the content can be handled, otherwise False.

Return type bool

handle_chunk(*download_id*, *url*, *to_path*, *start*, *end*, *chunk_size=1024*)
Handles downloading a specific range of bytes for a url.

Parameters

- **download_id** (*str*) – The unique id of the download request.
- **url** (*str*) – The url to download.
- **to_path** (*str*) – The local path to save the download.
- **start** (*int*) – The starting byte position to download.
- **end** (*int*) – The ending byte position to download.
- **chunk_size** (*int, optional*) – The size of the chunks to stream in.

handle_download(*download_id*, *url*, *to_path*, *max_connections=8*)
Handles downloading a specific url.

Note: `max_connections` defaults to 8 because many content hosting sites will typically flag/ban IPs that use over 10 connections.

Parameters

- **download_id** (*str*) – The unique id of the download request.
- **url** (*str*) – The url to download.
- **to_path** (*str*) – The local path to save the download.
- **max_connections** (*int, optional*) – The number of allowed connections for parallel downloading of the url.

session
requests.Session – The requests session to use for downloading.

Return type `Session`

2.3 Indices

- genindex
- modindex
- search

Python Module Index

q

Index

I

K

M

N

P

Q

S

T

U

V