# darc

*Release 0.1.0*

**Jarry Shaw**

**Mar 08, 2020**

# CONTENTS

CHAPTER

# ONE

# DARKWEB CRAWLER PROJECT

`darc` is designed as a swiss-knife for darkweb crawling. It integrates `requests` to collect HTTP request and response information, such as cookies, header fields, etc. It also bundles `selenium` to provide a fully rendered web page and screenshot of such view.

`darc.save.`**`_SAVE_LOCK = multiprocessing.Lock()`**
I/O lock for saving link hash database `link.csv`.

**See also:**

- `darc.save.save_link()`

`darc.submit.`**`PATH_API = '{PATH_DB}/api/'`**
Path to the API submittsion records, i.e. `api` folder under the root of data storage.

**See also:**

- *darc.const.PATH_DB*

`darc.submit.`**`API_RETRY: int`**
Retry times for API submission when failure.

**Default** `3`

**Environ** *API_RETRY*

`darc.submit.`**`API_NEW_HOST: str`**
API URL for `submit_new_host()`.

**Default** `None`

**Environ** *API_NEW_HOST*

`darc.submit.`**`API_REQUESTS: str`**
API URL for `submit_requests()`.

**Default** `None`

**Environ** *API_REQUESTS*

`darc.submit.`**`API_SELENIUM: str`**
API URL for `submit_selenium()`.

**Default** `None`

**Environ** *API_SELENIUM*

**Note:** If *API_NEW_HOST*, *API_REQUESTS* and *API_SELENIUM* is `None`, the corresponding submit function will save the JSON data in the path specified by *PATH_API*.

darc.db.**QR_LOCK = multiprocessing.Lock()**
I/O lock for the `requests` database `_queue_requests.txt`.

**See also:**

- `darc.db.save_requests()`

darc.db.**QS_LOCK = multiprocessing.Lock() | threading.Lock() | contextlib.nullcontext()**
I/O lock for the `selenium` database `_queue_selenium.txt`.

If *FLAG_MP* is `True`, it will be an instance of `multiprocessing.Lock`. If *FLAG_TH* is `True`, it will be an instance of `threading.Lock`. If none above, it will be an instance of `contextlib.nullcontext`.

**See also:**

- `darc.db.save_selenium()`
- *darc.const.FLAG_MP*
- *darc.const.FLAG_TH*

## 1.1 Proxy Utilities

### 1.1.1 No Proxy

### 1.1.2 Tor Proxy

### 1.1.3 I2P Proxy

### 1.1.4 ZeroNet Proxy

### 1.1.5 Freenet Proxy

### 1.1.6 Bitcoin Addresses

### 1.1.7 Data URI Schemes

### 1.1.8 ED2K Magnet Links

### 1.1.9 IRC Addresses

### 1.1.10 Magnet Links

### 1.1.11 Email Addresses

## 1.2 Site Specific Customisation

### 1.2.1 Default Hooks

## 1.3 Module Constants

### 1.3.1 Auxiliary Function

### 1.3.2 General Configurations

`darc.const.`**`REBOOT: bool`**
If exit the program after first round, i.e. crawled all links from the `requests` link database and loaded all links from the `selenium` link database.

**Default** `False`

**Environ** *DARC_REBOOT*

`darc.const.`**`DEBUG: bool`**
If run the program in debugging mode.

**Default** `False`

**Environ** *DARC_DEBUG*

`darc.const.`**`VERBOSE: bool`**
If run the program in verbose mode. If *DEBUG* is `True`, then the verbose mode will be always enabled.

**Default** `False`

**Environ** *DARC_VERBOSE*

`darc.const.`**FORCE: bool**
If ignore `robots.txt` rules when crawling (c.f. `crawler()`).

**Default** `False`

**Environ** *DARC_FORCE*

`darc.const.`**CHECK: bool**
If check proxy and hostname before crawling (when calling `extract_links()`, `read_sitemap()` and `read_hosts()`).

If *CHECK_NG* is `True`, then this environment variable will be always set as `True`.

**Default** `False`

**Environ** *DARC_CHECK*

`darc.const.`**CHECK_NG: bool**
If check content type through `HEAD` requests before crawling (when calling `extract_links()`, `read_sitemap()` and `read_hosts()`).

**Default** `False`

**Environ** *DARC_CHECK_CONTENT_TYPE*

`darc.const.`**ROOT: str**
The root folder of the project.

`darc.const.`**CWD = '.'**
The current working direcory.

`darc.const.`**DARC_CPU: int**
Number of concurrent processes. If not provided, then the number of system CPUs will be used.

**Default** `None`

**Environ** *DARC_CPU*

`darc.const.`**FLAG_MP: bool**
If enable *multiprocessing* support.

**Default** `True`

**Environ** *DARC_MULTIPROCESSING*

`darc.const.`**FLAG_TH: bool**
If enable *multithreading* support.

**Default** `False`

**Environ** *DARC_MULTITHREADING*

---

**Note:** *FLAG_MP* and *FLAG_TH* can **NOT** be toggled at the same time.

---

`darc.const.`**DARC_USER: str**
*Non-root* user for proxies.

**Default** current login user (c.f. `getpass.getuser()`)

**Environ** *DARC_USER*

### 1.3.3 Data Storage

darc.const.**PATH_DB: str**
Path to data storage.

**Default** `data`

**Environ** *PATH_DATA*

**See also:**

See `darc.save` for more information about source saving.

darc.const.**PATH_MISC = '{PATH_DB}/misc/'**
Path to miscellaneous data storage, i.e. `misc` folder under the root of data storage.

**See also:**

- *darc.const.PATH_DB*

darc.const.**PATH_LN = '{PATH_DB}/link.csv'**
Path to the link CSV file, `link.csv`.

**See also:**

- *darc.const.PATH_DB*
- `darc.save.save_link`

darc.const.**PATH_QR = '{PATH_DB}/_queue_requests.txt'**
Path to the `requests` database, `_queue_requests.txt`.

**See also:**

- *darc.const.PATH_DB*
- `darc.db.load_requests()`
- `darc.db.save_requests()`

darc.const.**PATH_QS = '{PATH_DB}/_queue_selenium.txt'**
Path to the `selenium` database, `_queue_selenium.txt`.

**See also:**

- *darc.const.PATH_DB*
- `darc.db.load_selenium()`
- `darc.db.save_selenium()`

darc.const.**PATH_ID = '{PATH_DB}/darc.pid'**
Path to the process ID file, `darc.pid`.

**See also:**

- *darc.const.PATH_DB*
- `darc.const.getpid()`

### 1.3.4 Web Crawlers

darc.const.**TIME_CACHE: float**
Time delta for caches in seconds.

The darc project supports *caching* for fetched files. *TIME_CACHE* will specify for how log the fetched files will be cached and **NOT** fetched again.

**Note:** If *TIME_CACHE* is None then caching will be marked as *forever*.

**Default** 60

**Environ** *TIME_CACHE*

darc.const.**SE_WAIT: float**
Time to wait for selenium to finish loading pages.

**Note:** Internally, selenium will wait for the browser to finish loading the pages before return (i.e. the web API event DOMContentLoaded). However, some extra scripts may take more time running after the event.

**Default** 60

**Environ** *SE_WAIT*

darc.const.**SE_EMPTY = '<html><head></head><body></body></html>'**
The empty page from selenium.

**See also:**

- darc.crawl.loader()

### 1.3.5 White / Black Lists

darc.const.**LINK_WHITE_LIST: List[str]**
White list of hostnames should be crawled.

**Default** []

**Environ** *LINK_WHITE_LIST*

**Note:** Regular expressions are supported.

darc.const.**LINK_BLACK_LIST: List[str]**
Black list of hostnames should be crawled.

**Default** []

**Environ** *LINK_BLACK_LIST*

**Note:** Regular expressions are supported.

darc.const.**MIME_WHITE_LIST: List[str]**
White list of content types should be crawled.

**Default** []

**Environ** *MIME_WHITE_LIST*

**Note:** Regular expressions are supported.

darc.const.**MIME_BLACK_LIST: List[str]**
Black list of content types should be crawled.

**Default** []

**Environ** *MIME_BLACK_LIST*

**Note:** Regular expressions are supported.

darc.const.**PROXY_WHITE_LIST: List[str]**
White list of proxy types should be crawled.

**Default** []

**Environ** *PROXY_WHITE_LIST*

**Note:** Regular expressions are supported.

darc.const.**PROXY_BLACK_LIST: List[str]**
Black list of proxy types should be crawled.

**Default** []

**Environ** *PROXY_BLACK_LIST*

**Note:** Regular expressions are supported.

As the websites can be sometimes irritating for their anti-robots verification, login requirements, etc., the darc project also privides hooks to customise crawling behaviours around both requests and selenium.

**See also:**

Such customisation, as called in the darc project, site hooks, is site specific, user can set up your own hooks unto a certain site, c.f. darc.sites for more information.

Still, since the network is a world full of mysteries and miracles, the speed of crawling will much depend on the response speed of the target website. To boost up, as well as meet the system capacity, the darc project introduced multiprocessing, multithreading and the fallback slowest single-threaded solutions when crawling.

**Note:** When rendering the target website using selenium powered by the renown Google Chrome, it will require much memory to run. Thus, the three solutions mentioned above would only toggle the behaviour around the use of selenium.

To keep the darc project as it is a swiss-knife, only the main entrypoint function darc.process.process() is exported in global namespace (and renamed to darc.darc()), see below:

darc is designed as a swiss-knife for darkweb crawling. It integrates requests to collect HTTP request and response information, such as cookies, header fields, etc. It also bundles selenium to provide a fully rendered web page and screenshot of such view.

The general process of darc can be described as following:

0. process(): obtain URLs from the requests link database (c.f. load_requests()), and feed such URLs to crawler() with *multiprocessing* support.

1. crawler(): parse the URL using parse_link(), and check if need to crawl the URL (c.f. *PROXY_WHITE_LIST*, *PROXY_BLACK_LIST* , *LINK_WHITE_LIST* and *LINK_BLACK_LIST*); if true, then crawl the URL with requests.

   If the URL is from a brand new host, darc will first try to fetch and save robots.txt and sitemaps of the host (c.f. save_robots() and save_sitemap()), and extract then save the links from sitemaps (c.f. read_sitemap()) into link database for future crawling (c.f. save_requests()). Also, if the submission API is provided, submit_new_host() will be called and submit the documents just fetched.

   If robots.txt presented, and *FORCE* is False, darc will check if allowed to crawl the URL.

   **Note:** The root path (e.g. / in https://www.example.com/) will always be crawled ignoring robots.txt.

   At this point, darc will call the customised hook function from darc.sites to crawl and get the final response object. darc will save the session cookies and header information, using save_headers().

   **Note:** If requests.exceptions.InvalidSchema is raised, the link will be saved by save_invalid(). Further processing is dropped.

   If the content type of response document is not ignored (c.f. *MIME_WHITE_LIST* and *MIME_BLACK_LIST*), darc will save the document using save_html() or save_file() accordingly. And if the submission API is provided, submit_requests() will be called and submit the document just fetched.

   If the response document is HTML (text/html and application/xhtml+xml), extract_links() will be called then to extract all possible links from the HTML document and save such links into the database (c.f. save_requests()).

   And if the response status code is between 400 and 600, the URL will be saved back to the link database (c.f. save_requests()). If **NOT**, the URL will be saved into selenium link database to proceed next steps (c.f. save_selenium()).

2. process(): after the obtained URLs have all been crawled, darc will obtain URLs from the selenium link database (c.f. load_selenium()), and feed such URLs to loader().

   **Note:** If *FLAG_MP* is True, the function will be called with *multiprocessing* support; if *FLAG_TH* if True, the function will be called with *multithreading* support; if none, the function will be called in single-threading.

3. loader(): parse the URL using parse_link() and start loading the URL using selenium with Google Chrome.

   At this point, darc will call the customised hook function from darc.sites to load and return the original selenium.webdriver.Chrome object.

   If successful, the rendered source HTML document will be saved using save_html(), and a full-page screenshot will be taken and saved.

   If the submission API is provided, submit_selenium() will be called and submit the document just loaded.

Later, `extract_links()` will be called then to extract all possible links from the HTML document and save such links into the `requests` database (c.f. `save_requests()`).

CHAPTER

TWO

# INSTALLATION

**Note:** `darc` supports Python all versions above and includes **3.8**. Currently, it only supports and is tested on Linux (Ubuntu 18.04) and macOS (Catalina).

```
pip install darc
```

Please make sure you have Google Chrome and corresponding version of Chrome Driver installed on your system.

However, the `darc` project is shipped with Docker and Compose support. Please see the project root for relevant files and more information.

CHAPTER

THREE

# USAGE

The `darc` project provides a simple CLI:

```
usage: darc [-h] [-f FILE] ...

darkweb swiss knife crawler

positional arguments:
  link                  links to craw

optional arguments:
  -h, --help            show this help message and exit
  -f FILE, --file FILE  read links from file
```

It can also be called through module entrypoint:

```
python -m darc ...
```

**Note:** The link files can contain **comment** lines, which should start with `#`. Empty lines and comment lines will be ignored when loading.

CHAPTER

FOUR

# CONFIGURATION

Though simple CLI, the `darc` project is more configurable by environment variables.

## 4.1 General Configurations

**DARC_REBOOT: bool (int)**
If exit the program after first round, i.e. crawled all links from the `requests` link database and loaded all links from the `selenium` link database.

**Default** `0`

**DARC_DEBUG: bool (int)**
If run the program in debugging mode.

**Default** `0`

**DARC_VERBOSE: bool (int)**
If run the program in verbose mode. If *DARC_DEBUG* is `True`, then the verbose mode will be always enabled.

**Default** `0`

**DARC_FORCE: bool (int)**
If ignore `robots.txt` rules when crawling (c.f. `crawler()`).

**Default** `0`

**DARC_CHECK: bool (int)**
If check proxy and hostname before crawling (when calling `extract_links()`, `read_sitemap()` and `read_hosts()`).

If *DARC_CHECK_CONTENT_TYPE* is `True`, then this environment variable will be always set as `True`.

**Default** `0`

**DARC_CHECK_CONTENT_TYPE: bool (int)**
If check content type through `HEAD` requests before crawling (when calling `extract_links()`, `read_sitemap()` and `read_hosts()`).

**Default** `0`

**DARC_CPU: int**
Number of concurrent processes. If not provided, then the number of system CPUs will be used.

**Default** `None`

**DARC_MULTIPROCESSING: bool (int)**
If enable *multiprocessing* support.

**Default** `1`

**DARC_MULTITHREADING: bool (int)**
If enable *multithreading* support.

**Default** `0`

**Note:** *DARC_MULTIPROCESSING* and *DARC_MULTITHREADING* can **NOT** be toggled at the same time.

**DARC_USER: str**
*Non-root* user for proxies.

**Default** current login user (c.f. `getpass.getuser()`)

## 4.2 Data Storage

**PATH_DATA: str (path)**
Path to data storage.

**Default** `data`

**See also:**

See `darc.save` for more information about source saving.

## 4.3 Web Crawlers

**TIME_CACHE: float**
Time delta for caches in seconds.

The `darc` project supports *caching* for fetched files. *TIME_CACHE* will specify for how log the fetched files will be cached and **NOT** fetched again.

**Note:** If *TIME_CACHE* is `None` then caching will be marked as *forever*.

**Default** `60`

**SE_WAIT: float**
Time to wait for `selenium` to finish loading pages.

**Note:** Internally, `selenium` will wait for the browser to finish loading the pages before return (i.e. the web API event `DOMContentLoaded`). However, some extra scripts may take more time running after the event.

**Default** `60`

## 4.4 White / Black Lists

**LINK_WHITE_LIST: List[str] (json)**
White list of hostnames should be crawled.

**Default** `[]`

**Note:** Regular expressions are supported.

**LINK_BLACK_LIST: List[str] (json)**
Black list of hostnames should be crawled.

**Default** `[]`

**Note:** Regular expressions are supported.

**MIME_WHITE_LIST: List[str] (json)**
White list of content types should be crawled.

**Default** `[]`

**Note:** Regular expressions are supported.

**MIME_BLACK_LIST: List[str] (json)**
Black list of content types should be crawled.

**Default** `[]`

**Note:** Regular expressions are supported.

**PROXY_WHITE_LIST: List[str] (json)**
White list of proxy types should be crawled.

**Default** `[]`

**Note:** Regular expressions are supported.

**PROXY_BLACK_LIST: List[str] (json)**
Black list of proxy types should be crawled.

**Default** `[]`

**Note:** Regular expressions are supported.

**Note:** If provided, *LINK_WHITE_LIST*, *LINK_BLACK_LIST*, *MIME_WHITE_LIST*, *MIME_BLACK_LIST*, *PROXY_WHITE_LIST* and *PROXY_BLACK_LIST* should all be JSON encoded strings.

## 4.5 Data Submission

**API_RETRY: int**
Retry times for API submission when failure.

**Default** 3

**API_NEW_HOST: str**
API URL for `submit_new_host()`.

**Default** None

**API_REQUESTS: str**
API URL for `submit_requests()`.

**Default** None

**API_SELENIUM: str**
API URL for `submit_selenium()`.

**Default** None

**Note:** If *API_NEW_HOST*, *API_REQUESTS* and *API_SELENIUM* is None, the corresponding submit function will save the JSON data in the path specified by *PATH_DATA*.

## 4.6 Tor Proxy Configuration

**TOR_PORT: int**
Port for Tor proxy connection.

**Default** 9050

**TOR_CTRL: int**
Port for Tor controller connection.

**Default** 9051

**TOR_STEM: bool (int)**
If manage the Tor proxy through stem.

**Default** 1

**TOR_PASS: str**
Tor controller authentication token.

**Default** None

**Note:** If not provided, it will be requested at runtime.

**TOR_RETRY: int**
Retry times for Tor bootstrap when failure.

**Default** 3

**TOR_WAIT: float**
Time after which the attempt to start Tor is aborted.

**Default** 90

**Note:** If not provided, there will be **NO** timeouts.

**TOR_CFG: Dict[str, Any] (json)**
Tor bootstrap configuration for `stem.process.launch_tor_with_config()`.

**Default** `{}`

**Note:** If provided, it should be a JSON encoded string.

## 4.7 I2P Proxy Configuration

**I2P_PORT: int**
Port for I2P proxy connection.

**Default** `4444`

**I2P_RETRY: int**
Retry times for I2P bootstrap when failure.

**Default** `3`

**I2P_WAIT: float**
Time after which the attempt to start I2P is aborted.

**Default** `90`

**Note:** If not provided, there will be **NO** timeouts.

**I2P_ARG: str (shell)**
I2P bootstrap arguments for `i2prouter start`.

If provided, it should be parsed as command line arguments (c.f. `shlex.split`).

**Default** `''`

**Note:** The command will be run as *DARC_USER*, if current user (c.f. `getpass.getuser()`) is *root*.

## 4.8 ZeroNet Proxy Configuration

**ZERONET_PORT: int**
Port for ZeroNet proxy connection.

**Default** `4444`

**ZERONET_RETRY: int**
Retry times for ZeroNet bootstrap when failure.

**Default** `3`

**ZERONET_WAIT: float**
Time after which the attempt to start ZeroNet is aborted.

**Default** `90`

**Note:** If not provided, there will be **NO** timeouts.

**ZERONET_PATH: str (path)**
Path to the ZeroNet project.

**Default** `/usr/local/src/zeronet`

**ZERONET_ARG: str (shell)**
ZeroNet bootstrap arguments for `ZeroNet.sh main`.

**Default** `''`

**Note:** If provided, it should be parsed as command line arguments (c.f. `shlex.split`).

## 4.9 Freenet Proxy Configuration

**FREENET_PORT: int**
Port for Freenet proxy connection.

**Default** `4444`

**FREENET_RETRY: int**
Retry times for Freenet bootstrap when failure.

**Default** `3`

**FREENET_WAIT: float**
Time after which the attempt to start Freenet is aborted.

**Default** `90`

**Note:** If not provided, there will be **NO** timeouts.

**FREENET_PATH: str (path)**
Path to the Freenet project.

**Default** `/usr/local/src/freenet`

**FREENET_ARG: str (shell)**
Freenet bootstrap arguments for `run.sh start`.

If provided, it should be parsed as command line arguments (c.f. `shlex.split`).

**Default** `''`

**Note:** The command will be run as *DARC_USER*, if current user (c.f. `getpass.getuser()`) is *root*.

CHAPTER

FIVE

# INDICES AND TABLES

- genindex
- modindex
- search

# INDEX

## S



## T



## Z