
darc

Release 0.6.9

Jarry Shaw

Nov 21, 2020

CONTENTS

1	Darkweb Crawler Project	1
1.1	Main Processing	1
1.2	Web Crawlers	3
1.3	URL Utilities	5
1.4	Source Parsing	8
1.5	Source Saving	10
1.6	Link Database	13
1.7	Data Submission	21
1.8	Requests Wrapper	29
1.9	Selenium Wrapper	30
1.10	Proxy Utilities	32
1.11	Sites Customisation	48
1.12	Module Constants	55
1.13	Custom Exceptions	60
1.14	Data Models	62
2	Docker Integration	75
3	Web Backend Demo	85
4	Data Models Demo	91
5	Submission Data Schema	95
5.1	New Host Submission	95
5.2	Requests Submission	99
5.3	Selenium Submission	104
5.4	Model Definitions	107
6	Auxiliary Scripts	113
6.1	Health Check	113
6.2	Upload API Submission Files	113
6.3	Remove Repeated Lines	114
7	Installation	117
8	Usage	119
9	Configuration	121
9.1	General Configurations	121
9.2	Data Storage	122
9.3	Web Crawlers	124

9.4	White / Black Lists	125
9.5	Data Submission	127
9.6	Tor Proxy Configuration	128
9.7	I2P Proxy Configuration	129
9.8	ZeroNet Proxy Configuration	130
9.9	Freenet Proxy Configuration	131
10	Indices and tables	133
	Python Module Index	135
	Index	137

DARKWEB CRAWLER PROJECT

`darc` is designed as a swiss army 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.

1.1 Main Processing

The `darc.process` module contains the main processing logic of the `darc` module.

`darc.process._process(worker)`

Wrapper function to start the worker process.

Parameters `worker` (`Union[darc.process.process_crawler, darc.process.process_loader]`) –

`darc.process._signal_handler(signum=None, frame=None)`

Signal handler.

If the current process is not the main process, the function shall do nothing.

Parameters

- **signum** (`Optional[Union[int, signal.Signals]]`) – The signal to handle.
- **frame** (`types.FrameType`) – The traceback frame from the signal.

See also:

- `darc.const.getpid()`

`darc.process.process(worker)`

Main process.

The function will register `_signal_handler()` for SIGTERM, and start the main process of the `darc` darkweb crawlers.

Parameters `worker` (`Literal[crawler, loader]`) – Worker process type.

Raises `ValueError` – If `worker` is not a valid value.

Before starting the workers, the function will start proxies through

- `darc.proxy.tor.tor_proxy()`
- `darc.proxy.i2p.i2p_proxy()`
- `darc.proxy.zeronet.zeronet_proxy()`

- `darc.proxy.freenet.freenet_proxy()`

The general process can be described as following for *workers* of crawler type:

1. `process_crawler()`: obtain URLs from the `requests` link database (c.f. `load_requests()`), and feed such URLs to `crawler()`.

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.

2. `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`), `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()`).

The general process can be described as following for *workers* of loader type:

1. `process_loader()`: in the meanwhile, *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.

2. `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 Chrome object.

If successful, the rendered source HTML document will be saved, 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()*).

If in reboot mode, i.e. *REBOOT* is *True*, the function will exit after first round. If not, it will renew the Tor connections (if bootstrapped), c.f. *renew_tor_session()*, and start another round.

darc.process.process_crawler()
A worker to run the *crawler()* process.

darc.process.process_loader()
A worker to run the *loader()* process.

darc.process._WORKER_POOL = None
List of active child processes and/or threads.

Type List[Union[multiprocessing.Process, threading.Thread]]

1.2 Web Crawlers

The *darc.crawl* module provides two types of crawlers.

- *crawler()* – crawler powered by *requests*
- *loader()* – crawler powered by *selenium*

darc.crawl.crawler(link)
Single *requests* crawler for a entry link.

Parameters *link* (*darc.link.Link*) – URL to be crawled by *requests*.

The function will first 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()*).

Note: A host is new if *have_hostname()* returns *True*.

If *darc.proxy.null.fetch_sitemap()* and/or *darc.proxy.i2p.fetch_hosts()* failed when fetching such documents, the host will be removed from the hostname database through *drop_hostname()*, and considered as new when next encounter.

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, and the link will be removed from the `requests` database through `drop_requests()`.

If `LinkNoReturn` is raised, the link will be removed from the `requests` database through `drop_requests()`.

If the content type of response document is not ignored (c.f. `MIME_WHITE_LIST` and `MIME_BLACK_LIST`), `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()`).

`darc.crawl.loader(link)`

Single selenium loader for a entry link.

Parameters

- **Link** – URL to be crawled by selenium.
- **link** (`darc.link.Link`) –

The function will first 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.

Note: If `LinkNoReturn` is raised, the link will be removed from the selenium database through `drop_selenium()`.

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

Note: When taking full-page screenshot, `loader()` will use `document.body.scrollHeight` to get the total height of web page. If the page height is *less than 1,000 pixels*, then `darc` will by default set the height as **1,000 pixels**.

Later `darc` will tell selenium to resize the window (in *headless* mode) to **1,024 pixels** in width and **110%** of the page height in height, and take a *PNG* screenshot.

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()`).

See also:

- `darc.const.SE_EMPTY`
- `darc.const.SE_WAIT`

1.3 URL Utilities

The `Link` class is the key data structure of the `darc` project, it contains all information required to identify a URL's proxy type, hostname, path prefix when saving, etc.

The `link` module also provides several wrapper function to the `urllib.parse` module.

class `darc.link.Link` (*url, proxy, url_parse, host, base, name*)

Bases: `object`

Parsed link.

Parameters

- **url** – original link
- **proxy** – proxy type
- **host** – URL's hostname
- **base** – base folder for saving files
- **name** – hashed link for saving files
- **url_parse** – parsed URL from `urllib.parse.urlparse()`

Returns Parsed link object.

Return type `Link`

Note: `Link` is a `dataclass` object. It is safely *hashable*, through `hash(url)`.

__hash__ ()

Provide hash support to the `Link` object.

base: `str`

base folder for saving files

host: `str`

URL's hostname

name: `str`

hashed link for saving files

proxy: `str`

proxy type

url: `str`

original link

url_parse: `urllib.parse.ParseResult`

parsed URL from `urllib.parse.urlparse()`

`darc.link.parse_link` (*link, host=None*)

Parse link.

Parameters

- **link** (*str*) – link to be parsed
- **host** (*Optional[str]*) – hostname of the link

Returns The parsed link object.

Return type *darc.link.Link*

Note: If `host` is provided, it will override the hostname of the original link.

The parsing process of proxy type is as follows:

0. If `host` is `None` and the parse result from `urllib.parse.urlparse()` has no `netloc` (or `hostname`) specified, then set `hostname` as `(null)`; else set it as is.
1. If the scheme is `data`, then the link is a data URI, set `hostname` as `data` and `proxy` as `data`.
2. If the scheme is `javascript`, then the link is some JavaScript codes, set `proxy` as `script`.
3. If the scheme is `bitcoin`, then the link is a Bitcoin address, set `proxy` as `bitcoin`.
4. If the scheme is `ed2k`, then the link is an ED2K magnet link, set `proxy` as `ed2k`.
5. If the scheme is `magnet`, then the link is a magnet link, set `proxy` as `magnet`.
6. If the scheme is `mailto`, then the link is an email address, set `proxy` as `mail`.
7. If the scheme is `irc`, then the link is an IRC link, set `proxy` as `irc`.
8. If the scheme is **NOT** any of `http` or `https`, then set `proxy` to the scheme.
9. If the `host` is `None`, set `hostname` to `(null)`, set `proxy` to `null`.
10. If the `host` is an onion (`.onion`) address, set `proxy` to `tor`.
11. If the `host` is an I2P (`.i2p`) address, or any of `localhost:7657` and `localhost:7658`, set `proxy` to `i2p`.
12. If the `host` is `localhost` on `ZERONET_PORT`, and the path is not `/`, i.e. **NOT** root path, set `proxy` to `zeronet`; and set the first part of its path as `hostname`.

Example:

For a ZeroNet address, e.g. `http://127.0.0.1:43110/1HeLLo4uzjaLetFx6NH3PMwFP3qbRbTf3D`, `parse_link()` will parse the `hostname` as `1HeLLo4uzjaLetFx6NH3PMwFP3qbRbTf3D`.

13. If the `host` is `localhost` on `FREENET_PORT`, and the path is not `/`, i.e. **NOT** root path, set `proxy` to `freenet`; and set the first part of its path as `hostname`.

Example:

For a Freenet address, e.g. `http://127.0.0.1:8888/USK@nwa8lHa271k2QvJ8aa0Ov7IHAV-DFOCFgmDt3X6BpCI,DuQSUZiI~agF8c-6tjsFFGuZ8eICrzWCILB60nT8KKo,AQACAAE/sone/77/`, `parse_link()` will parse the `hostname` as `USK@nwa8lHa271k2QvJ8aa0Ov7IHAV-DFOCFgmDt3X6BpCI,DuQSUZiI~agF8c-6tjsFFGuZ8eICrzWCILB60nT8KKo,AQACAAE`.

14. If the `host` is a proxied onion (`.onion.sh`) address, set `proxy` to `tor2web`.
15. If none of the cases above satisfied, the `proxy` will be set as `null`, marking it a plain normal link.

The base for parsed link *Link* object is defined as

```
<root>/<proxy>/<scheme>/<hostname>/
```

where root is `PATH_DB`.

The name for parsed link `Link` object is the sha256 hash (c.f. `hashlib.sha256()`) of the original link.

`darc.link.quote(string, safe='/', encoding=None, errors=None)`

Wrapper function for `urllib.parse.quote()`.

Parameters

- **string** (*AnyStr*) – string to be quoted
- **safe** (*AnyStr*) – characters not to escape
- **encoding** (*Optional[str]*) – string encoding
- **errors** (*Optional[str]*) – encoding error handler

Returns The quoted string.

Return type `str`

Note: The function suppressed possible errors when calling `urllib.parse.quote()`. If any, it will return the original string.

`darc.link.unquote(string, encoding='utf-8', errors='replace')`

Wrapper function for `urllib.parse.unquote()`.

Parameters

- **string** (*AnyStr*) – string to be unquoted
- **encoding** (*str*) – string encoding
- **errors** (*str*) – encoding error handler

Returns The quoted string.

Return type `str`

Note: The function suppressed possible errors when calling `urllib.parse.unquote()`. If any, it will return the original string.

`darc.link.urljoin(base, url, allow_fragments=True)`

Wrapper function for `urllib.parse.urljoin()`.

Parameters

- **base** (*AnyStr*) – base URL
- **url** (*AnyStr*) – URL to be joined
- **allow_fragments** (*bool*) – if allow fragments

Returns The joined URL.

Return type `str`

Note: The function suppressed possible errors when calling `urllib.parse.urljoin()`. If any, it will return `base/url` directly.

`darc.link.urlparse(url, scheme="", allow_fragments=True)`
Wrapper function for `urllib.parse.urlparse()`.

Parameters

- **url** (*str*) – URL to be parsed
- **scheme** (*str*) – URL scheme
- **allow_fragments** (*bool*) – if allow fragments

Returns The parse result.

Return type `urllib.parse.ParseResult`

Note: The function suppressed possible errors when calling `urllib.parse.urlparse()`. If any, it will return `urllib.parse.ParseResult(scheme=scheme, netloc='', path=url, params='', query='', fragment='')` directly.

1.4 Source Parsing

The `darc.parse` module provides auxiliary functions to read `robots.txt`, sitemaps and HTML documents. It also contains utility functions to check if the proxy type, hostname and content type if in any of the black and white lists.

`darc.parse._check(temp_list)`
Check hostname and proxy type of links.

Parameters **temp_list** (*List[darc.link.Link]*) – List of links to be checked.

Returns List of links matches the requirements.

Return type *List[darc.link.Link]*

Note: If `CHECK_NG` is `True`, the function will directly call `_check_ng()` instead.

See also:

- `darc.parse.match_host()`
- `darc.parse.match_proxy()`

`darc.parse._check_ng(temp_list)`
Check content type of links through HEAD requests.

Parameters **temp_list** (*List[darc.link.Link]*) – List of links to be checked.

Returns List of links matches the requirements.

Return type *List[darc.link.Link]*

See also:

- `darc.parse.match_host()`
- `darc.parse.match_proxy()`
- `darc.parse.match_mime()`

`darc.parse.check_robots(link)`

Check if link is allowed in robots.txt.

Parameters `link` (`darc.link.Link`) – The link object to be checked.

Returns If link is allowed in robots.txt.

Return type `bool`

Note: The root path of a URL will always return `True`.

`darc.parse.extract_links(link, html, check=False)`

Extract links from HTML document.

Parameters

- `link` (`darc.link.Link`) – Original link of the HTML document.
- `html` (`Union[str, bytes]`) – Content of the HTML document.
- `check` (`bool`) – If perform checks on extracted links, default to `CHECK`.

Returns List of extracted links.

Return type `List[darc.link.Link]`

See also:

- `darc.parse._check()`
- `darc.parse._check_ng()`

`darc.parse.get_content_type(response)`

Get content type from response.

Parameters `response` (`requests.Response`) – Response object.

Returns The content type from response.

Return type `str`

Note: If the Content-Type header is not defined in response, the function will utilise `magic` to detect its content type.

`darc.parse.match_host(host)`

Check if hostname in black list.

Parameters `host` (`str`) – Hostname to be checked.

Returns If host in black list.

Return type `bool`

Note: If host is `None`, then it will always return `True`.

See also:

- `darc.const.LINK_WHITE_LIST`
- `darc.const.LINK_BLACK_LIST`
- `darc.const.LINK_FALLBACK`

`darc.parse.match_mime(mime)`

Check if content type in black list.

Parameters `mime` (*str*) – Content type to be checked.

Returns If `mime` in black list.

Return type `bool`

See also:

- `darc.const.MIME_WHITE_LIST`
- `darc.const.MIME_BLACK_LIST`
- `darc.const.MIME_FALLBACK`

`darc.parse.match_proxy(proxy)`

Check if proxy type in black list.

Parameters `proxy` (*str*) – Proxy type to be checked.

Returns If `proxy` in black list.

Return type `bool`

Note: If `proxy` is `script`, then it will always return `True`.

See also:

- `darc.const.PROXY_WHITE_LIST`
- `darc.const.PROXY_BLACK_LIST`
- `darc.const.PROXY_FALLBACK`

1.5 Source Saving

The `darc.save` module contains the core utilities for managing fetched files and documents.

The data storage under the root path (`PATH_DB`) is typically as following:

```
data
├── api
│   └── <proxy>
│       └── <scheme>
│           └── <hostname>
│               ├── new_host
│               │   └── <hash>_<timestamp>.json
│               └── requests
```

(continues on next page)

(continued from previous page)



`darc.save.sanitise` (*link*, *time=None*, *raw=False*, *data=False*, *headers=False*, *screenshot=False*)
 Sanitise link to path.

Parameters

- **link** (`darc.link.Link`) – Link object to sanitise the path
- **time** (`datetime`) – Timestamp for the path.
- **raw** (`bool`) – If this is a raw HTML document from `requests`.
- **data** (`bool`) – If this is a generic content type document.
- **headers** (`bool`) – If this is response headers from `requests`.
- **screenshot** (`bool`) – If this is the screenshot from `selenium`.

Returns

- If `raw` is `True`, `<root>/<proxy>/<scheme>/<hostname>/<hash>_<timestamp>_raw.html`.
- If `data` is `True`, `<root>/<proxy>/<scheme>/<hostname>/<hash>_<timestamp>.dat`.
- If `headers` is `True`, `<root>/<proxy>/<scheme>/<hostname>/<hash>_<timestamp>.json`.
- If `screenshot` is `True`, `<root>/<proxy>/<scheme>/<hostname>/<hash>_<timestamp>.png`.
- If none above, `<root>/<proxy>/<scheme>/<hostname>/<hash>_<timestamp>.html`.

Return type `str`

See also:

- `darc.crawl.crawler()`
- `darc.crawl.loader()`

`darc.save.save_headers` (*time, link, response, session*)

Save HTTP response headers.

Parameters

- **time** (*datetime*) – Timestamp of response.
- **link** (`darc.link.Link`) – Link object of response.
- **response** (`requests.Response`) – Response object to be saved.
- **session** (`requests.Session`) – Session object of response.

Returns Saved path to response headers, i.e. `<root>/<proxy>/<scheme>/<hostname>/<hash>_<timestamp>.json`.

Return type `str`

The JSON data saved is as following:

```
{
  "[metadata]": {
    "url": "...",
    "proxy": "...",
    "host": "...",
    "base": "...",
    "name": "..."
  },
  "Timestamp": "...",
  "URL": "...",
  "Method": "GET",
  "Status-Code": "...",
  "Reason": "...",
  "Cookies": {
    "...": "..."
  },
  "Session": {
    "...": "..."
  },
  "Request": {
    "...": "..."
  },
  "Response": {
    "...": "..."
  },
  "History": [
    { "...": "..." }
  ]
}
```

See also:

- `darc.save.sanitise()`
- `darc.crawl.crawler()`

`darc.save.save_link` (*link*)

Save link hash database `link.csv`.

The CSV file has following fields:

- proxy type: `link.proxy`

- URL scheme: `link.url_parse.scheme`
- hostname: `link.base`
- link hash: `link.name`
- original URL: `link.url`

Parameters `link` (`darc.link.Link`) – Link object to be saved.

See also:

- `darc.const.PATH_LN`
- `darc.save._SAVE_LOCK`

`darc.save._SAVE_LOCK`: `Union[multiprocessing.Lock, threading.Lock, contextlib.nullcontext]`
I/O lock for saving link hash database `link.csv`.

See also:

- `darc.save.save_link()`
- `darc.const.get_lock()`

1.6 Link Database

The `darc` project utilises [Redis](#) based database to provide tele-process communication.

Note: In its first implementation, the `darc` project used [Queue](#) to support such communication. However, as noticed when runtime, the [Queue](#) object will be much affected by the lack of memory.

There will be three databases, all following the save naming convention with `queue_` prefix:

- the hostname database – `queue_hostname` (`HostnameQueueModel`)
- the `requests` database – `queue_requests` (`RequestsQueueModel`)
- the selenium database – `queue_selenium` (`SeleniumQueueModel`)

For `queue_hostname`, `queue_requests` and `queue_selenium`, they are all [Redis sorted set](#) data type.

If `FLAG_DB` is `True`, then the module uses the RDS storage described by the `peewee` models as backend.

`darc.db._db_operation` (`operation`, `*args`, `**kwargs`)

Retry operation on database.

Parameters

- **operation** (`Callable[[...], T]`) – Callable / method to perform.
- ***args** – Arbitrary positional arguments.

Keyword Arguments ****kwargs** – Arbitrary keyword arguments.

Returns Any return value from a successful operation call.

Return type `T`

`darc.db._drop_hostname_db(link)`

Remove link from the hostname database.

The function updates the `HostnameQueueModel` table.

Parameters `link` (`darc.link.Link`) – Link to be removed.

`darc.db._drop_hostname_redis(link)`

Remove link from the hostname database.

The function updates the `queue_hostname` database.

Parameters `link` (`darc.link.Link`) – Link to be removed.

`darc.db._drop_requests_db(link)`

Remove link from the `requests` database.

The function updates the `RequestsQueueModel` table.

Parameters `link` (`darc.link.Link`) – Link to be removed.

`darc.db._drop_requests_redis(link)`

Remove link from the `requests` database.

The function updates the `queue_requests` database.

Parameters `link` (`darc.link.Link`) – Link to be removed.

`darc.db._drop_selenium_db(link)`

Remove link from the selenium database.

The function updates the `SeleniumQueueModel` table.

Parameters `link` (`darc.link.Link`) – Link to be removed.

`darc.db._drop_selenium_redis(link)`

Remove link from the selenium database.

The function updates the `queue_selenium` database.

Parameters `link` (`darc.link.Link`) – Link to be removed.

`darc.db._gen_arg_msg(*args, **kwargs)`

Sanitise arguments representation string.

Parameters `*args` – Arbitrary arguments.

Keyword Arguments `**kwargs` – Arbitrary keyword arguments.

Returns Sanitised arguments representation string.

Return type `str`

`darc.db._have_hostname_db(link)`

Check if current link is a new host.

The function checks the `HostnameQueueModel` table.

Parameters `link` (`darc.link.Link`) – Link to check against.

Returns A tuple of two `bool` values representing if such link is a known host and needs force refetch respectively.

Return type `Tuple[bool, bool]`

`darc.db._have_hostname_redis(link)`

Check if current link is a new host.

The function checks the `queue_hostname` database.

Parameters `link` (`darc.link.Link`) – Link to check against.

Returns A tuple of two `bool` values representing if such link is a known host and needs force refetch respectively.

Return type `Tuple[bool, bool]`

`darc.db._load_requests_db()`

Load link from the `requests` database.

The function reads the `RequestsQueueModel` table.

Returns List of loaded links from the `requests` database.

Return type `List[darc.link.Link]`

Note: At runtime, the function will load links with maximum number at `MAX_POOL` to limit the memory usage.

`darc.db._load_requests_redis()`

Load link from the `requests` database.

The function reads the `queue_requests` database.

Returns List of loaded links from the `requests` database.

Return type `List[darc.link.Link]`

Note: At runtime, the function will load links with maximum number at `MAX_POOL` to limit the memory usage.

`darc.db._load_selenium_db()`

Load link from the `selenium` database.

The function reads the `SeleniumQueueModel` table.

Returns List of loaded links from the `selenium` database.

Return type `List[darc.link.Link]`

Note: At runtime, the function will load links with maximum number at `MAX_POOL` to limit the memory usage.

`darc.db._load_selenium_redis()`

Load link from the `selenium` database.

The function reads the `queue_selenium` database.

Parameters `check` – If perform checks on loaded links, default to `CHECK`.

Returns List of loaded links from the `selenium` database.

Return type `List[darc.link.Link]`

Note: At runtime, the function will load links with maximum number at `MAX_POOL` to limit the memory usage.

`darc.db._redis_command(command, *args, **kwargs)`

Wrapper function for Redis command.

Parameters

- **command** (*str*) – Command name.
- ***args** – Arbitrary arguments for the Redis command.

Keyword Arguments ****kwargs** – Arbitrary keyword arguments for the Redis command.

Returns Values returned from the Redis command.

Warns **RedisCommandFailed** – Warns at each round when the command failed.

Return type Any

See also:

Between each retry, the function sleeps for `RETRY_INTERVAL` second(s) if such value is **NOT** `None`.

`darc.db._redis_get_lock(name, timeout=None, sleep=0.1, blocking_timeout=None, lock_class=None, thread_local=True)`

Get a lock for Redis operations.

Parameters

- **name** (*str*) – Lock name.
- **timeout** (*Optional[float]*) – Maximum life for the lock.
- **sleep** (*float*) – Amount of time to sleep per loop iteration when the lock is in blocking mode and another client is currently holding the lock.
- **blocking_timeout** (*Optional[float]*) – Maximum amount of time in seconds to spend trying to acquire the lock.
- **lock_class** (*Optional[redis.lock.Lock]*) – Lock implementation.
- **thread_local** (*bool*) – Whether the lock token is placed in thread-local storage.

Returns Return a new `redis.lock.Lock` object using key name that mimics the behavior of `threading.Lock`.

Return type Union[`redis.lock.Lock`, `contextlib.nullcontext`]

See Also: If `REDIS_LOCK` is `False`, returns a `contextlib.nullcontext` instead.

`darc.db._save_requests_db(entries, single=False, score=None, nx=False, xx=False)`

Save link to the `requests` database.

The function updates the `RequestsQueueModel` table.

Parameters

- **entries** (*Union[darc.link.Link, List[darc.link.Link]]*) – Links to be added to the `requests` database. It can be either a `list` of links, or a single link string (if `single` set as `True`).
- **single** (*bool*) – Indicate if `entries` is a `list` of links or a single link string.
- **score** – Score to for the Redis sorted set.
- **nx** – Only create new elements and not to update scores for elements that already exist.
- **xx** – Only update scores of elements that already exist. New elements will not be added.

`darc.db._save_requests_redis` (*entries*, *single=False*, *score=None*, *nx=False*, *xx=False*)

Save link to the `requests` database.

The function updates the `queue_requests` database.

Parameters

- **entries** (*Union[darc.link.Link, List[darc.link.Link]]*) – Links to be added to the `requests` database. It can be either a `list` of links, or a single link string (if `single` set as `True`).
- **single** (*bool*) – Indicate if `entries` is a `list` of links or a single link string.
- **score** – Score to for the Redis sorted set.
- **nx** – Forces `ZADD` to only create new elements and not to update scores for elements that already exist.
- **xx** – Forces `ZADD` to only update scores of elements that already exist. New elements will not be added.

`darc.db._save_selenium_db` (*entries*, *single=False*, *score=None*, *nx=False*, *xx=False*)

Save link to the `selenium` database.

The function updates the `SeleniumQueueModel` table.

Parameters

- **entries** (*Union[darc.link.Link, List[darc.link.Link]]*) – Links to be added to the `selenium` database. It can be either a `list` of links, or a single link string (if `single` set as `True`).
- **single** (*bool*) – Indicate if `entries` is a `list` of links or a single link string.
- **score** – Score to for the Redis sorted set.
- **nx** – Only create new elements and not to update scores for elements that already exist.
- **xx** – Only update scores of elements that already exist. New elements will not be added.

`darc.db._save_selenium_redis` (*entries*, *single=False*, *score=None*, *nx=False*, *xx=False*)

Save link to the `selenium` database.

The function updates the `queue_selenium` database.

Parameters

- **entries** (*Union[darc.link.Link, List[darc.link.Link]]*) – Links to be added to the `selenium` database. It can be either an *iterable* of links, or a single link string (if `single` set as `True`).
- **single** (*bool*) – Indicate if `entries` is an *iterable* of links or a single link string.
- **score** – Score to for the Redis sorted set.
- **nx** – Forces `ZADD` to only create new elements and not to update scores for elements that already exist.
- **xx** – Forces `ZADD` to only update scores of elements that already exist. New elements will not be added.

When `entries` is a list of `Link` instances, we tries to perform *bulk* update to easy the memory consumption. The *bulk* size is defined by `BULK_SIZE`.

Notes

The `entries` will be dumped through `pickle` so that `darc` do not need to parse them again.

`darc.db.drop_hostname(link)`

Remove link from the `hostname` database.

Parameters `link` (`darc.link.Link`) – Link to be removed.

See also:

- `darc.db._drop_hostname_db()`
- `darc.db._drop_hostname_redis()`

`darc.db.drop_requests(link)`

Remove link from the `requests` database.

Parameters `link` (`darc.link.Link`) – Link to be removed.

See also:

- `darc.db._drop_requests_db()`
- `darc.db._drop_requests_redis()`

`darc.db.drop_selenium(link)`

Remove link from the `selenium` database.

Parameters `link` (`darc.link.Link`) – Link to be removed.

See also:

- `darc.db._drop_selenium_db()`
- `darc.db._drop_selenium_redis()`

`darc.db.have_hostname(link)`

Check if current link is a new host.

Parameters `link` (`darc.link.Link`) – Link to check against.

Returns A tuple of two `bool` values representing if such link is a known host and needs force refetch respectively.

Return type `Tuple[bool, bool]`

See also:

- `darc.db._have_hostname_db()`
- `darc.db._have_hostname_redis()`

`darc.db.load_requests(check=False)`

Load link from the `requests` database.

Parameters `check` (`bool`) – If perform checks on loaded links, default to `CHECK`.

Returns List of loaded links from the `requests` database.

Return type `List[darc.link.Link]`

Note: At runtime, the function will load links with maximum number at `MAX_POOL` to limit the memory usage.

See also:

- `darc.db._load_requests_db()`
- `darc.db._load_requests_redis()`

`darc.db.load_selenium(check=False)`
Load link from the `selenium` database.

Parameters `check (bool)` – If perform checks on loaded links, default to `CHECK`.

Returns List of loaded links from the `selenium` database.

Return type List[`darc.link.Link`]

Note: At runtime, the function will load links with maximum number at `MAX_POOL` to limit the memory usage.

See also:

- `darc.db._load_selenium_db()`
- `darc.db._load_selenium_redis()`

`darc.db.save_requests(entries, single=False, score=None, nx=False, xx=False)`
Save link to the `requests` database.

The function updates the `queue_requests` database.

Parameters

- **entries** (`Union[darc.link.Link, List[darc.link.Link]]`) – Links to be added to the `requests` database. It can be either a `list` of links, or a single link string (if `single` set as `True`).
- **single** (`bool`) – Indicate if `entries` is a `list` of links or a single link string.
- **score** – Score to for the Redis sorted set.
- **nx** – Only create new elements and not to update scores for elements that already exist.
- **xx** – Only update scores of elements that already exist. New elements will not be added.

Notes

The `entries` will be dumped through `pickle` so that `darc` do not need to parse them again.

When `entries` is a list of `Link` instances, we tries to perform *bulk* update to easy the memory consumption. The *bulk* size is defined by `BULK_SIZE`.

See also:

- `darc.db._save_requests_db()`
- `darc.db._save_requests_redis()`

`darc.db.save_selenium(entries, single=False, score=None, nx=False, xx=False)`

Save link to the selenium database.

Parameters

- **entries** (`Union[darc.link.Link, List[darc.link.Link]]`) – Links to be added to the selenium database. It can be either a `list` of links, or a single link string (if `single` set as `True`).
- **single** (`bool`) – Indicate if `entries` is a `list` of links or a single link string.
- **score** – Score to for the Redis sorted set.
- **nx** – Only create new elements and not to update scores for elements that already exist.
- **xx** – Only update scores of elements that already exist. New elements will not be added.

Notes

The `entries` will be dumped through `pickle` so that `darc` do not need to parse them again.

When `entries` is a list of `Link` instances, we tries to perform *bulk* update to easy the memory consumption. The *bulk* size is defined by `BULK_SIZE`.

See also:

- `darc.db._save_selenium_db()`
- `darc.db._save_selenium_redis()`

`darc.db.BULK_SIZE: int`

Default 100

Environ `DARC_BULK_SIZE`

Bulk size for updating Redis databases.

See also:

- `darc.db.save_requests()`
- `darc.db.save_selenium()`

`darc.db.LOCK_TIMEOUT: Optional[float]`

Default 10

Environ `DARC_LOCK_TIMEOUT`

Lock blocking timeout.

Note: If is an infinit `inf`, no timeout will be applied.

See also:

Get a lock from `darc.db.get_lock()`.

`darc.db.MAX_POOL: int`

Default 1_000

Environ `DARC_MAX_POOL`

Maximum number of links loading from the database.

Note: If is an infinit `inf`, no limit will be applied.

`darc.db.REDIS_LOCK: bool`

Default `False`

Environ `DARC_REDIS_LOCK`

If use Redis (Lua) lock to ensure process/thread-safely operations.

See also:

Toggles the behaviour of `darc.db.get_lock()`.

`darc.db.RETRY_INTERVAL: int`

Default `10`

Environ `DARC_RETRY`

Retry interval between each Redis command failure.

Note: If is an infinit `inf`, no interval will be applied.

See also:

Toggles the behaviour of `darc.db.redis_command()`.

1.7 Data Submission

The *darc* project integrates the capability of submitting fetched data and information to a web server, to support real-time cross-analysis and status display.

There are three submission events:

1. New Host Submission – `API_NEW_HOST`

Submitted in `crawler()` function call, when the crawling URL is marked as a new host.

2. Requests Submission – `API_REQUESTS`

Submitted in `crawler()` function call, after the crawling process of the URL using `requests`.

3. Selenium Submission – `API_SELENIUM`

Submitted in `loader()` function call, after the loading process of the URL using `selenium`.

See also:

Please refer to *data schema* for more information about the submission data.

`darc.submit.get_hosts(link)`

Read `hosts.txt`.

Parameters `link` (`darc.link.Link`) – Link object to read `hosts.txt`.

Returns

- If `hosts.txt` exists, return the data from `hosts.txt`.
 - `path` – relative path from `hosts.txt` to root of data storage `PATH_DB`, `<proxy>/<scheme>/<hostname>/hosts.txt`
 - `data` – `base64` encoded content of `hosts.txt`
- If not, return `None`.

Return type `Optional[Dict[str, AnyStr]]`

See also:

- `darc.crawl.crawler()`
- `darc.proxy.i2p.save_hosts()`

`darc.submit.get_metadata(link)`

Generate metadata field.

Parameters `link` (`darc.link.Link`) – Link object to generate metadata.

Returns

The metadata from `link`.

- `url` – original URL, `link.url`
- `proxy` – proxy type, `link.proxy`
- `host` – hostname, `link.host`
- `base` – base path, `link.base`
- `name` – link hash, `link.name`

Return type `Dict[str, str]`

`darc.submit.get_robots(link)`

Read `robots.txt`.

Parameters `link` (`darc.link.Link`) – Link object to read `robots.txt`.

Returns

- If `robots.txt` exists, return the data from `robots.txt`.
 - `path` – relative path from `robots.txt` to root of data storage `PATH_DB`, `<proxy>/<scheme>/<hostname>/robots.txt`
 - `data` – `base64` encoded content of `robots.txt`
- If not, return `None`.

Return type `Optional[Dict[str, AnyStr]]`

See also:

- `darc.crawl.crawler()`
- `darc.proxy.null.save_robots()`

`darc.submit.get_sitemaps(link)`

Read sitemaps.

Parameters `link` (`darc.link.Link`) – Link object to read sitemaps.

Returns

- If sitemaps exist, return list of the data from sitemaps.
 - path – relative path from sitemap to root of data storage `PATH_DB`, `<proxy>/<scheme>/<hostname>/sitemap_<hash>.xml`
 - data – *base64* encoded content of sitemap
- If not, return `None`.

Return type `Optional[List[Dict[str, AnyStr]]]`

See also:

- `darc.crawl.crawler()`
- `darc.proxy.null.save_sitemap()`

`darc.submit.save_submit(domain, data)`
Save failed submit data.

Parameters

- **domain** (`'new_host'`, `'requests'` or `'selenium'`) – Domain of the submit data.
- **data** (`Dict[str, Any]`) – Submit data.

See also:

- `darc.submit.PATH_API`
- `darc.submit.submit()`
- `darc.submit.submit_new_host()`
- `darc.submit.submit_requests()`
- `darc.submit.submit_selenium()`

`darc.submit.submit(api, domain, data)`
Submit data.

Parameters

- **api** (`str`) – API URL.
- **domain** (`'new_host'`, `'requests'` or `'selenium'`) – Domain of the submit data.
- **data** (`Dict[str, Any]`) – Submit data.

See also:

- `darc.submit.API_RETRY`
- `darc.submit.save_submit()`
- `darc.submit.submit_new_host()`
- `darc.submit.submit_requests()`
- `darc.submit.submit_selenium()`

`darc.submit.submit_new_host (time, link, partial=False, force=False)`

Submit new host.

When a new host is discovered, the `darc` crawler will submit the host information. Such includes `robots.txt` (if exists) and `sitemap.xml` (if any).

Parameters

- **time** (`datetime.datetime`) – Timestamp of submission.
- **link** (`darc.link.Link`) – Link object of submission.
- **partial** (`bool`) – If the data is not complete, i.e. failed when fetching `robots.txt`, `hosts.txt` and/or sitemaps.
- **force** (`bool`) – If the data is force re-fetched, i.e. cache expired when checking with `darc.db.have_hostname()`.

If `API_NEW_HOST` is `None`, the data for submission will directly be save through `save_submit()`.

The data submitted should have following format:

```
{
  // partial flag - true / false
  "$PARTIAL$": ...,
  // force flag - true / false
  "$FORCE$": ...,
  // metadata of URL
  "[metadata]": {
    // original URL - <scheme>://<netloc>/<path>;<params>?<query>#<fragment>
    "url": ...,
    // proxy type - null / tor / i2p / zeronet / freenet
    "proxy": ...,
    // hostname / netloc, c.f. ``urllib.parse.urlparse``
    "host": ...,
    // base folder, relative path (to data root path ``PATH_DATA``) in_
    ↪container - <proxy>/<scheme>/<host>
    "base": ...,
    // sha256 of URL as name for saved files (timestamp is in ISO format)
    //   JSON log as this one - <base>/<name>_<timestamp>.json
    //   HTML from requests - <base>/<name>_<timestamp>_raw.html
    //   HTML from selenium - <base>/<name>_<timestamp>.html
    //   generic data files - <base>/<name>_<timestamp>.dat
    "name": ...
  },
  // requested timestamp in ISO format as in name of saved file
  "Timestamp": ...,
  // original URL
  "URL": ...,
  // robots.txt from the host (if not exists, then ``null``)
  "Robots": {
    // path of the file, relative path (to data root path ``PATH_DATA``) in_
    ↪container
    //   - <proxy>/<scheme>/<host>/robots.txt
    "path": ...,
    // content of the file (**base64** encoded)
    "data": ...,
  },
  // sitemaps from the host (if none, then ``null``)
  "Sitemaps": [
    {
```

(continues on next page)

(continued from previous page)

```

        // path of the file, relative path (to data root path ``PATH_DATA``)
    ↪in container
        // - <proxy>/<scheme>/<host>/sitemap_<name>.xml
        "path": ...,
        // content of the file (**base64** encoded)
        "data": ...,
    },
    ...
],
// hosts.txt from the host (if proxy type is ``i2p``; if not exists, then
    ↪``null``)
    "Hosts": {
        // path of the file, relative path (to data root path ``PATH_DATA``) in
    ↪container
        // - <proxy>/<scheme>/<host>/hosts.txt
        "path": ...,
        // content of the file (**base64** encoded)
        "data": ...,
    }
}

```

See also:

- `darc.submit.API_NEW_HOST`
- `darc.submit.submit()`
- `darc.submit.save_submit()`
- `darc.submit.get_metadata()`
- `darc.submit.get_robots()`
- `darc.submit.get_sitemaps()`
- `darc.submit.get_hosts()`

`darc.submit.submit_requests` (*time, link, response, session, content, mime_type, html=True*)

Submit requests data.

When crawling, we'll first fetch the URI using `requests`, to check its availability and to save its HTTP headers information. Such information will be submitted to the web UI.

Parameters

- **time** (*datetime.datetime*) – Timestamp of submission.
- **link** (*darc.link.Link*) – Link object of submission.
- **response** (*requests.Response*) – Response object of submission.
- **session** (*requests.Session*) – Session object of submission.
- **content** (*bytes*) – Raw content of from the response.
- **mime_type** (*str*) – Content type.
- **html** (*bool*) – If current document is HTML or other files.

If `API_REQUESTS` is `None`, the data for submission will directly be save through `save_submit()`.

The data submitted should have following format:

```

{
    // metadata of URL
    "[metadata]": {
        // original URL - <scheme>://<netloc>/<path>;<params>?<query>#<fragment>
        "url": ...,
        // proxy type - null / tor / i2p / zeronet / freenet
        "proxy": ...,
        // hostname / netloc, c.f. ``urllib.parse.urlparse``
        "host": ...,
        // base folder, relative path (to data root path ``PATH_DATA``) in_
        ↪container - <proxy>/<scheme>/<host>
        "base": ...,
        // sha256 of URL as name for saved files (timestamp is in ISO format)
        // JSON log as this one - <base>/<name>_<timestamp>.json
        // HTML from requests - <base>/<name>_<timestamp>_raw.html
        // HTML from selenium - <base>/<name>_<timestamp>.html
        // generic data files - <base>/<name>_<timestamp>.dat
        "name": ...
    },
    // requested timestamp in ISO format as in name of saved file
    "Timestamp": ...,
    // original URL
    "URL": ...,
    // request method
    "Method": "GET",
    // response status code
    "Status-Code": ...,
    // response reason
    "Reason": ...,
    // response cookies (if any)
    "Cookies": {
        ...
    },
    // session cookies (if any)
    "Session": {
        ...
    },
    // request headers (if any)
    "Request": {
        ...
    },
    // response headers (if any)
    "Response": {
        ...
    },
    // content type
    "Content-Type": ...,
    // requested file (if not exists, then ``null``)
    "Document": {
        // path of the file, relative path (to data root path ``PATH_DATA``) in_
        ↪container
        // - <proxy>/<scheme>/<host>/<name>_<timestamp>_raw.html
        // or if the document is of generic content type, i.e. not HTML
        // - <proxy>/<scheme>/<host>/<name>_<timestamp>.dat
        "path": ...,
        // content of the file (**base64** encoded)
        "data": ...,
    },
}

```

(continues on next page)

(continued from previous page)

```

    },
    // redirection history (if any)
    "History": [
        // same record data as the original response
        {"...": "..."}
    ]
}

```

See also:

- `darc.submit.API_REQUESTS`
- `darc.submit.submit()`
- `darc.submit.save_submit()`
- `darc.submit.get_metadata()`
- `darc.submit.get_raw()`
- `darc.crawl.crawler()`

`darc.submit.submit_selenium(time, link, html, screenshot)`

Submit selenium data.

After crawling with `requests`, we'll then render the URI using `selenium` with Google Chrome and its web driver, to provide a fully rendered web page. Such information will be submitted to the web UI.

Parameters

- **time** (`datetime.datetime`) – Timestamp of submission.
- **link** (`darc.link.Link`) – Link object of submission.
- **html** (`str`) – HTML source of the web page.
- **screenshot** (`Optional[str]`) – `base64` encoded screenshot.

If `API_SELENIUM` is `None`, the data for submission will directly be save through `save_submit()`.

Note: This information is optional, only provided if the content type from `requests` is HTML, status code not between 400 and 600, and HTML data not empty.

The data submitted should have following format:

```

{
    // metadata of URL
    "[metadata]": {
        // original URL - <scheme>://<netloc>/<path>;<params>?<query>#<fragment>
        "url": ...,
        // proxy type - null / tor / i2p / zeronet / freenet
        "proxy": ...,
        // hostname / netloc, c.f. ``urllib.parse.urlparse``
        "host": ...,
        // base folder, relative path (to data root path ``PATH_DATA``) in_
        ↪container - <proxy>/<scheme>/<host>
        "base": ...,
        // sha256 of URL as name for saved files (timestamp is in ISO format)
        //   JSON log as this one - <base>/<name>_<timestamp>.json
    }
}

```

(continues on next page)

(continued from previous page)

```

        // HTML from requests - <base>/<name>_<timestamp>_raw.html
        // HTML from selenium - <base>/<name>_<timestamp>.html
        // generic data files - <base>/<name>_<timestamp>.dat
        "name": ...
    },
    // requested timestamp in ISO format as in name of saved file
    "Timestamp": ...,
    // original URL
    "URL": ...,
    // rendered HTML document (if not exists, then ``null``)
    "Document": {
        // path of the file, relative path (to data root path ``PATH_DATA``) in
        ↪container
        // - <proxy>/<scheme>/<host>/<name>_<timestamp>.html
        "path": ...,
        // content of the file (**base64** encoded)
        "data": ...,
    },
    // web page screenshot (if not exists, then ``null``)
    "Screenshot": {
        // path of the file, relative path (to data root path ``PATH_DATA``) in
        ↪container
        // - <proxy>/<scheme>/<host>/<name>_<timestamp>.png
        "path": ...,
        // content of the file (**base64** encoded)
        "data": ...,
    }
}

```

See also:

- `darc.submit.API_SELENIUM`
- `darc.submit.submit()`
- `darc.submit.save_submit()`
- `darc.submit.get_metadata()`
- `darc.submit.get_html()`
- `darc.submit.get_screenshot()`
- `darc.crawl.loader()`

`darc.submit.PATH_API = '{PATH_DB}/api/'`

Path to the API submission records, i.e. `api` folder under the root of data storage.

See also:

- `darc.const.PATH_DB`

`darc.submit.SAVE_DB: bool`

Save submitted data to database.

Default `True`

Environ `SAVE_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`.

See also:

The `darc` provides a demo on how to implement a `darc`-compliant web backend for the data submission module. See the [demo](#) page for more information.

1.8 Requests Wrapper

The `darc.requests` module wraps around the `requests` module, and provides some simple interface for the `darc` project.

`darc.requests.default_user_agent(name='python-darc', proxy=None)`
 Generates the default user agent.

Parameters

- **name** (`str`) – Base name.
- **proxy** (`Optional[str]`) – Proxy type.

Returns User agent in format of `{name}/{darc.__version__} ({proxy} Proxy)`.

Return type `str`

`darc.requests.i2p_session(futures=False)`
 I2P (.i2p) session.

Parameters **futures** (`bool`) – If returns a `requests_futures.FuturesSession`.

Returns The session object with I2P proxy settings.

Return type `Union[requests.Session, requests_futures.FuturesSession]`

See also:

- `darc.proxy.i2p.I2P_REQUESTS_PROXY`

`darc.requests.null_session(futures=False)`

No proxy session.

Parameters `futures` (*bool*) – If returns a `requests_futures.FuturesSession`.

Returns The session object with no proxy settings.

Return type `Union[requests.Session, requests_futures.FuturesSession]`

`darc.requests.request_session(link, futures=False)`

Get requests session.

Parameters

- **link** (`darc.link.Link`) – Link requesting for `requests.Session`.
- **futures** (*bool*) – If returns a `requests_futures.FuturesSession`.

Returns The session object with corresponding proxy settings.

Return type `Union[requests.Session, requests_futures.FuturesSession]`

Raises *UnsupportedLink* – If the proxy type of `link` if not specified in the `LINK_MAP`.

See also:

- `darc.proxy.LINK_MAP`

`darc.requests.tor_session(futures=False)`

Tor (.onion) session.

Parameters `futures` (*bool*) – If returns a `requests_futures.FuturesSession`.

Returns The session object with Tor proxy settings.

Return type `Union[requests.Session, requests_futures.FuturesSession]`

See also:

- `darc.proxy.tor.TOR_REQUESTS_PROXY`

1.9 Selenium Wrapper

The `darc.selenium` module wraps around the `selenium` module, and provides some simple interface for the `darc` project.

`darc.selenium.get_capabilities(type='null')`

Generate desied capabilities.

Parameters `type` (*str*) – Proxy type for capabilities.

Returns The desied capabilities for the web driver Chrome.

Raises *UnsupportedProxy* – If the proxy type is **NOT** `null`, `tor` or `i2p`.

Return type `dict`

See also:

- `darc.proxy.tor.TOR_SELENIUM_PROXY`
- `darc.proxy.i2p.I2P_SELENIUM_PROXY`

`darc.selenium.get_options (type='null')`

Generate options.

Parameters `type (str)` – Proxy type for options.

Returns The options for the web driver Chrome.

Return type `selenium.webdriver.ChromeOptions`

Raises

- ***UnsupportedPlatform*** – If the operation system is **NOT** macOS or Linux.
- ***UnsupportedProxy*** – If the proxy type is **NOT** `null`, `tor` or `i2p`.

See also:

- `darc.proxy.tor.TOR_PORT`
- `darc.proxy.i2p.I2P_PORT`

References

- [Google Chrome command line switches](#)
 - Disable sandbox (`--no-sandbox`) when running as root user
 - <https://crbug.com/638180>
 - <https://stackoverflow.com/a/50642913/7218152>
 - Disable usage of `/dev/shm`
 - <http://crbug.com/715363>
 - [Using Socks proxy](#)
-

`darc.selenium.i2p_driver ()`

I2P (.i2p) driver.

Returns The web driver object with I2P proxy settings.

Return type `selenium.webdriver.Chrome`

See also:

- `darc.selenium.get_options ()`
- `darc.selenium.get_capabilities ()`

`darc.selenium.null_driver ()`

No proxy driver.

Returns The web driver object with no proxy settings.

Return type `selenium.webdriver.Chrome`

See also:

- `darc.selenium.get_options ()`

- `darc.selenium.get_capabilities()`

`darc.selenium.request_driver(link)`

Get selenium driver.

Parameters `link` (`darc.link.Link`) – Link requesting for Chrome.

Returns The web driver object with corresponding proxy settings.

Return type `selenium.webdriver.Chrome`

Raises *UnsupportedLink* – If the proxy type of `link` if not specified in the `LINK_MAP`.

See also:

- `darc.proxy.LINK_MAP`

`darc.selenium.tor_driver()`

Tor(.onion) driver.

Returns The web driver object with Tor proxy settings.

Return type `selenium.webdriver.Chrome`

See also:

- `darc.selenium.get_options()`
- `darc.selenium.get_capabilities()`

1.10 Proxy Utilities

The `darc.proxy` module provides various proxy support to the `darc` project.

1.10.1 Bitcoin Addresses

The `darc.proxy.bitcoin` module contains the auxiliary functions around managing and processing the bitcoin addresses.

Currently, the `darc` project directly save the bitcoin addresses extracted to the data storage file `PATH` without further processing.

`darc.proxy.bitcoin.save_bitcoin(link)`

Save bitcoin address.

The function will save bitcoin address to the file as defined in `PATH`.

Parameters `link` (`darc.link.Link`) – Link object representing the bitcoin address.

`darc.proxy.bitcoin.PATH = '{PATH_MISC}/bitcoin.txt'`

Path to the data storage of bitcoin addresses.

See also:

- `darc.const.PATH_MISC`

`darc.proxy.bitcoin.LOCK: Union[multiprocessing.Lock, threading.Lock, contextlib.nullcontext]`
I/O lock for saving bitcoin addresses `PATH`.

See also:

- `darc.const.get_lock()`

1.10.2 Data URI Schemes

The `darc.proxy.data` module contains the auxiliary functions around managing and processing the data URI schemes.

Currently, the `darc` project directly save the data URI schemes extracted to the data storage path `PATH` without further processing.

`darc.proxy.data.save_data(link)`
Save data URI.

The function will save data URIs to the data storage as defined in `PATH`.

Parameters `link` (`darc.link.Link`) – Link object representing the data URI.

`darc.proxy.data.PATH = '{PATH_MISC}/data/'`
Path to the data storage of data URI schemes.

See also:

- `darc.const.PATH_MISC`

1.10.3 ED2K Magnet Links

The `darc.proxy.ed2k` module contains the auxiliary functions around managing and processing the ED2K magnet links.

Currently, the `darc` project directly save the ED2K magnet links extracted to the data storage file `PATH` without further processing.

`darc.proxy.ed2k.save_ed2k(link)`
Save ed2k magnet link.

The function will save ED2K magnet link to the file as defined in `PATH`.

Parameters `link` (`darc.link.Link`) – Link object representing the ED2K magnet links.

`darc.proxy.ed2k.PATH = '{PATH_MISC}/ed2k.txt'`
Path to the data storage of bED2K magnet links.

See also:

- `darc.const.PATH_MISC`

`darc.proxy.ed2k.LOCK: Union[multiprocessing.Lock, threading.Lock, contextlib.nullcontext]`
I/O lock for saving ED2K magnet links `PATH`.

See also:

- `darc.const.get_lock()`

1.10.4 Freenet Proxy

The `darc.proxy.freenet` module contains the auxiliary functions around managing and processing the Freenet proxy.

`darc.proxy.freenet._freenet_bootstrap()`
Freenet bootstrap.

The bootstrap arguments are defined as `__FREENET_ARGS`.

Raises `subprocess.CalledProcessError` – If the return code of `__FREENET_PROC` is non-zero.

See also:

- `darc.proxy.freenet.freenet_bootstrap()`
- `darc.proxy.freenet.BS_WAIT`
- `darc.proxy.freenet.__FREENET_BS_FLAG`
- `darc.proxy.freenet.__FREENET_PROC`

`darc.proxy.freenet.freenet_bootstrap()`
Bootstrap wrapper for Freenet.

The function will bootstrap the Freenet proxy. It will retry for `FREENET_RETRY` times in case of failure.

Also, it will **NOT** re-bootstrap the proxy as is guaranteed by `__FREENET_BS_FLAG`.

Warns `FreenetBootstrapFailed` – If failed to bootstrap Freenet proxy.

Raises `UnsupportedPlatform` – If the system is not supported, i.e. not macOS or Linux.

See also:

- `darc.proxy.freenet._freenet_bootstrap()`
- `darc.proxy.freenet.FREENET_RETRY`
- `darc.proxy.freenet.__FREENET_BS_FLAG`

The following constants are configuration through environment variables:

`darc.proxy.freenet.FREENET_PORT: int`
Port for Freenet proxy connection.

Default 8888

Environ `FREENET_PORT`

`darc.proxy.freenet.FREENET_RETRY: int`
Retry times for Freenet bootstrap when failure.

Default 3

Environ `FREENET_RETRY`

`darc.proxy.freenet.BS_WAIT: float`
Time after which the attempt to start Freenet is aborted.

Default 90

Environ `FREENET_WAIT`

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

`darc.proxy.freenet.FREENET_PATH`: **str**

Path to the Freenet project.

Default `/usr/local/src/freenet`

Environ `FREENET_PATH`

`darc.proxy.freenet.FREENET_ARGS`: **List[str]**

Freenet bootstrap arguments for `run.sh start`.

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

Default `''`

Environ `FREENET_ARGS`

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

The following constants are defined for internal usage:

`darc.proxy.freenet._MNG_FREENET`: **bool**

If manage Freenet proxy through `darc`.

Default `True`

Environ `DARC_FREENET`

`darc.proxy.freenet._FREENET_BS_FLAG`: **bool**

If the Freenet proxy is bootstrapped.

`darc.proxy.freenet._FREENET_PROC`: **subprocess.Popen**

Freenet proxy process running in the background.

`darc.proxy.freenet._FREENET_ARGS`: **List[str]**

Freenet proxy bootstrap arguments.

1.10.5 I2P Proxy

The `darc.proxy.i2p` module contains the auxiliary functions around managing and processing the I2P proxy.

`darc.proxy.i2p._i2p_bootstrap()`

I2P bootstrap.

The bootstrap arguments are defined as `_I2P_ARGS`.

Raises `subprocess.CallProcessError` – If the return code of `_I2P_PROC` is non-zero.

See also:

- `darc.proxy.i2p.i2p_bootstrap()`
- `darc.proxy.i2p.BS_WAIT`
- `darc.proxy.i2p._I2P_BS_FLAG`
- `darc.proxy.i2p._I2P_PROC`

`darc.proxy.i2p.fetch_hosts(link, force=False)`
Fetch `hosts.txt`.

Parameters

- **link** (`darc.link.Link`) – Link object to fetch for its `hosts.txt`.
- **force** (`bool`) – Force refetch `hosts.txt`.

Returns Content of the `hosts.txt` file.

`darc.proxy.i2p.get_hosts(link)`
Read `hosts.txt`.

Parameters **link** (`darc.link.Link`) – Link object to read `hosts.txt`.

Returns

- If `hosts.txt` exists, return the data from `hosts.txt`.
 - `path` – relative path from `hosts.txt` to root of data storage `PATH_DB`, `<proxy>/<scheme>/<hostname>/hosts.txt`
 - `data` – `base64` encoded content of `hosts.txt`
- If not, return `None`.

Return type `Optional[Dict[str, str]]`

See also:

- `darc.submit.submit_new_host()`
- `darc.proxy.i2p.save_hosts()`

`darc.proxy.i2p.have_hosts(link)`
Check if `hosts.txt` already exists.

Parameters **link** (`darc.link.Link`) – Link object to check if `hosts.txt` already exists.

Returns

- If `hosts.txt` exists, return the path to `hosts.txt`, i.e. `<root>/<proxy>/<scheme>/<hostname>/hosts.txt`.
- If not, return `None`.

Return type `Optional[str]`

`darc.proxy.i2p.i2p_bootstrap()`
Bootstrap wrapper for I2P.

The function will bootstrap the I2P proxy. It will retry for `I2P_RETRY` times in case of failure.

Also, it will **NOT** re-bootstrap the proxy as is guaranteed by `_I2P_BS_FLAG`.

Warns `I2PBootstrapFailed` – If failed to bootstrap I2P proxy.

Raises `UnsupportedPlatform` – If the system is not supported, i.e. not macOS or Linux.

See also:

- `darc.proxy.i2p._i2p_bootstrap()`
- `darc.proxy.i2p.I2P_RETRY`
- `darc.proxy.i2p._I2P_BS_FLAG`

`darc.proxy.i2p.read_hosts(text, check=False)`
Read `hosts.txt`.

Parameters

- **text** (*str*) – Content of `hosts.txt`.
- **check** (*bool*) – If perform checks on extracted links, default to `CHECK`.

Returns List of links extracted.

Return type List[`darc.link.Link`]

`darc.proxy.i2p.save_hosts(link, text)`
Save `hosts.txt`.

Parameters

- **link** (`darc.link.Link`) – Link object of `hosts.txt`.
- **text** (*str*) – Content of `hosts.txt`.

Returns Saved path to `hosts.txt`, i.e. `<root>/<proxy>/<scheme>/<hostname>/hosts.txt`.

Return type *str*

See also:

- `darc.save.sanitise()`

`darc.proxy.i2p.I2P_REQUESTS_PROXY: Dict[str, Any]`
Proxy for I2P sessions.

See also:

- `darc.requests.i2p_session()`

`darc.proxy.i2p.I2P_SELENIUM_PROXY: selenium.webdriver.common.proxy.Proxy`
Proxy for I2P web drivers.

See also:

- `darc.selenium.i2p_driver()`

The following constants are configuration through environment variables:

`darc.proxy.i2p.I2P_PORT: int`
Port for I2P proxy connection.

Default 4444

Environ `I2P_PORT`

`darc.proxy.i2p.I2P_RETRY: int`
Retry times for I2P bootstrap when failure.

Default 3

Environ `I2P_RETRY`

`darc.proxy.i2p.BS_WAIT: float`
Time after which the attempt to start I2P is aborted.

Default 90

Environ `I2P_WAIT`

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

`darc.proxy.i2p.I2P_ARGS: List[str]`

I2P bootstrap arguments for `i2prouter` start.

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

Default `''`

Environ `I2P_ARGS`

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

The following constants are defined for internal usage:

`darc.proxy.i2p._MNG_I2P: bool`

If manage I2P proxy through `darc`.

Default `True`

Environ `DARC_I2P`

`darc.proxy.i2p._I2P_BS_FLAG: bool`

If the I2P proxy is bootstrapped.

`darc.proxy.i2p._I2P_PROC: subprocess.Popen`

I2P proxy process running in the background.

`darc.proxy.i2p._I2P_ARGS: List[str]`

I2P proxy bootstrap arguments.

1.10.6 IRC Addresses

The `darc.proxy.irc` module contains the auxiliary functions around managing and processing the IRC addresses.

Currently, the `darc` project directly save the IRC addresses extracted to the data storage file `PATH` without further processing.

`darc.proxy.irc.save_irc(link)`

Save IRC address.

The function will save IRC address to the file as defined in `PATH`.

Parameters `link` (`darc.link.Link`) – Link object representing the IRC address.

`darc.proxy.irc.PATH = '{PATH_MISC}/irc.txt'`

Path to the data storage of IRC addresses.

See also:

- `darc.const.PATH_MISC`

`darc.proxy.irc.LOCK: Union[multiprocessing.Lock, threading.Lock, contextlib.nullcontext]`

I/O lock for saving IRC addresses `PATH`.

See also:

- `darc.const.get_lock()`

1.10.7 Magnet Links

The `darc.proxy.magnet` module contains the auxiliary functions around managing and processing the magnet links.

Currently, the `darc` project directly save the magnet links extracted to the data storage file `PATH` without further processing.

`darc.proxy.magnet.save_magnet(link)`
Save magnet link.

The function will save magnet link to the file as defined in `PATH`.

Parameters `link` (`darc.link.Link`) – Link object representing the magnet link

`darc.proxy.magnet.PATH = '{PATH_MISC}/magnet.txt'`
Path to the data storage of magnet links.

See also:

- `darc.const.PATH_MISC`

`darc.proxy.magnet.LOCK: Union[multiprocessing.Lock, threading.Lock, contextlib.nullcontext]`
I/O lock for saving magnet links `PATH`.

See also:

- `darc.const.get_lock()`

1.10.8 Email Addresses

The `darc.proxy.mail` module contains the auxiliary functions around managing and processing the email addresses.

Currently, the `darc` project directly save the email addresses extracted to the data storage file `PATH` without further processing.

`darc.proxy.mail.save_mail(link)`
Save email address.

The function will save email address to the file as defined in `PATH`.

Parameters `link` (`darc.link.Link`) – Link object representing the email address.

`darc.proxy.mail.PATH = '{PATH_MISC}/mail.txt'`
Path to the data storage of email addresses.

See also:

- `darc.const.PATH_MISC`

`darc.proxy.mail.LOCK: Union[multiprocessing.Lock, threading.Lock, contextlib.nullcontext]`
I/O lock for saving email addresses `PATH`.

See also:

- `darc.const.get_lock()`

1.10.9 No Proxy

The `darc.proxy.null` module contains the auxiliary functions around managing and processing normal websites with no proxy.

`darc.proxy.null.fetch_sitemap(link, force=False)`
Fetch sitemap.

The function will first fetch the `robots.txt`, then fetch the sitemaps accordingly.

Parameters

- **link** (`darc.link.Link`) – Link object to fetch for its sitemaps.
- **force** (`bool`) – Force refetch its sitemaps.

Returns Contents of `robots.txt` and sitemaps.

See also:

- `darc.proxy.null.read_robots()`
- `darc.proxy.null.read_sitemap()`
- `darc.parse.get_sitemap()`

`darc.proxy.null.get_sitemap(link, text, host=None)`
Fetch link to other sitemaps from a sitemap.

Parameters

- **link** (`darc.link.Link`) – Original link to the sitemap.
- **text** (`str`) – Content of the sitemap.
- **host** (`Optional[str]`) – Hostname of the URL to the sitemap, the value may not be same as in `link`.

Returns List of link to sitemaps.

Return type List[`darc.link.Link`]

Note: As specified in the sitemap protocol, it may contain links to other sitemaps.*⁰

`darc.proxy.null.have_robots(link)`
Check if `robots.txt` already exists.

Parameters **link** (`darc.link.Link`) – Link object to check if `robots.txt` already exists.

Returns

- If `robots.txt` exists, return the path to `robots.txt`, i.e. `<root>/<proxy>/<scheme>/<hostname>/robots.txt`.
- If not, return `None`.

Return type `Optional[str]`

⁰ <https://www.sitemaps.org/protocol.html#index>

`darc.proxy.null.have_sitemap(link)`

Check if sitemap already exists.

Parameters `link` (`darc.link.Link`) – Link object to check if sitemap already exists.

Returns

- If sitemap exists, return the path to the sitemap, i.e. `<root>/<proxy>/<scheme>/<hostname>/sitemap_<hash>.xml`.
- If not, return `None`.

Return type `Optional[str]`

`darc.proxy.null.read_robots(link, text, host=None)`

Read `robots.txt` to fetch link to sitemaps.

Parameters

- `link` (`darc.link.Link`) – Original link to `robots.txt`.
- `text` (`str`) – Content of `robots.txt`.
- `host` (`Optional[str]`) – Hostname of the URL to `robots.txt`, the value may not be same as in `link`.

Returns List of link to sitemaps.

Return type `List[darc.link.Link]`

Note: If the link to sitemap is not specified in `robots.txt`⁰, the fallback link `/sitemap.xml` will be used.

`darc.proxy.null.read_sitemap(link, text, check=False)`

Read sitemap.

Parameters

- `link` (`darc.link.Link`) – Original link to the sitemap.
- `text` (`str`) – Content of the sitemap.
- `check` (`bool`) – If perform checks on extracted links, default to `CHECK`.

Returns List of links extracted.

Return type `List[darc.link.Link]`

See also:

- `darc.parse._check()`
- `darc.parse._check_ng()`

`darc.proxy.null.save_invalid(link)`

Save link with invalid scheme.

The function will save link with invalid scheme to the file as defined in `PATH`.

Parameters `link` (`darc.link.Link`) – Link object representing the link with invalid scheme.

`darc.proxy.null.save_robots(link, text)`

Save `robots.txt`.

⁰ https://www.sitemaps.org/protocol.html#submit_robots

Parameters

- **link** (`darc.link.Link`) – Link object of `robots.txt`.
- **text** (`str`) – Content of `robots.txt`.

Returns Saved path to `robots.txt`, i.e. `<root>/<proxy>/<scheme>/<hostname>/robots.txt`.

Return type `str`

See also:

- `darc.save.sanitise()`

`darc.proxy.null.save_sitemap(link, text)`
Save sitemap.

Parameters

- **link** (`darc.link.Link`) – Link object of sitemap.
- **text** (`str`) – Content of sitemap.

Returns Saved path to sitemap, i.e. `<root>/<proxy>/<scheme>/<hostname>/sitemap_<hash>.xml`.

Return type `str`

See also:

- `darc.save.sanitise()`

`darc.proxy.null.PATH = '{PATH_MISC}/invalid.txt'`
Path to the data storage of links with invalid scheme.

See also:

- `darc.const.PATH_MISC`

`darc.proxy.null.LOCK: Union[multiprocessing.Lock, threading.Lock, contextlib.nullcontext]`
I/O lock for saving links with invalid scheme `PATH`.

See also:

- `darc.const.get_lock()`

1.10.10 JavaScript Links

The `darc.proxy.script` module contains the auxiliary functions around managing and processing the JavaScript links.

Currently, the `darc` project directly save the JavaScript links extracted to the data storage path `PATH` without further processing.

`darc.proxy.script.save_script(link)`
Save JavaScript link.

The function will save JavaScript link to the file as defined in `PATH`.

Parameters **link** (`darc.link.Link`) – Link object representing the JavaScript link.

`darc.proxy.script.PATH = '{PATH_MISC}/script.txt'`
 Path to the data storage of bitcoin addresses.

See also:

- `darc.const.PATH_MISC`

`darc.proxy.script.LOCK: Union[multiprocessing.Lock, threading.Lock, contextlib.nullcontext]`
 I/O lock for saving JavaScript links `PATH`.

See also:

- `darc.const.get_lock()`

1.10.11 Telephone Numbers

The `darc.proxy.tel` module contains the auxiliary functions around managing and processing the telephone numbers.

Currently, the `darc` project directly save the telephone numbers extracted to the data storage file `PATH` without further processing.

`darc.proxy.tel.save_tel(link)`
 Save telephone number.

The function will save telephone number to the file as defined in `PATH`.

Parameters `link` (`darc.link.Link`) – Link object representing the telephone number.

`darc.proxy.tel.PATH = '{PATH_MISC}/tel.txt'`
 Path to the data storage of bitcoin addresses.

See also:

- `darc.const.PATH_MISC`

`darc.proxy.tel.LOCK: Union[multiprocessing.Lock, threading.Lock, contextlib.nullcontext]`
 I/O lock for saving telephone numbers `PATH`.

See also:

- `darc.const.get_lock()`

1.10.12 Tor Proxy

The `darc.proxy.tor` module contains the auxiliary functions around managing and processing the Tor proxy.

`darc.proxy.tor._tor_bootstrap()`
 Tor bootstrap.

The bootstrap configuration is defined as `_TOR_CONFIG`.

If `TOR_PASS` not provided, the function will request for it.

See also:

- `darc.proxy.tor.tor_bootstrap()`

- `darc.proxy.tor.BS_WAIT`
- `darc.proxy.tor.TOR_PASS`
- `darc.proxy.tor._TOR_BS_FLAG`
- `darc.proxy.tor._TOR_PROC`
- `darc.proxy.tor._TOR_CTRL`

`darc.proxy.tor.print_bootstrap_lines(line)`
Print Tor bootstrap lines.

Parameters `line` (*str*) – Tor bootstrap line.

`darc.proxy.tor.renew_tor_session()`
Renew Tor session.

`darc.proxy.tor.tor_bootstrap()`
Bootstrap wrapper for Tor.

The function will bootstrap the Tor proxy. It will retry for `TOR_RETRY` times in case of failure.

Also, it will **NOT** re-bootstrap the proxy as is guaranteed by `_TOR_BS_FLAG`.

Warns `TorBootstrapFailed` – If failed to bootstrap Tor proxy.

See also:

- `darc.proxy.tor._tor_bootstrap()`
- `darc.proxy.tor.TOR_RETRY`
- `darc.proxy.tor._TOR_BS_FLAG`

`darc.proxy.tor.TOR_REQUESTS_PROXY: Dict[str, Any]`
Proxy for Tor sessions.

See also:

- `darc.requests.tor_session()`

`darc.proxy.tor.TOR_SELENIUM_PROXY: selenium.webdriver.common.proxy.Proxy`
Proxy for Tor web drivers.

See also:

- `darc.selenium.tor_driver()`

The following constants are configuration through environment variables:

`darc.proxy.tor.TOR_PORT: int`
Port for Tor proxy connection.

Default 9050

Environ `TOR_PORT`

`darc.proxy.tor.TOR_CTRL: int`
Port for Tor controller connection.

Default 9051

Environ `TOR_CTRL`

`darc.proxy.tor.TOR_PASS: str`

Tor controller authentication token.

Default `None`

Environ `TOR_PASS`

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

`darc.proxy.tor.TOR_RETRY: int`

Retry times for Tor bootstrap when failure.

Default `3`

Environ `TOR_RETRY`

`darc.proxy.tor.BS_WAIT: float`

Time after which the attempt to start Tor is aborted.

Default `90`

Environ `TOR_WAIT`

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

`darc.proxy.tor.TOR_CFG: Dict[str, Any]`

Tor bootstrap configuration for `stem.process.launch_tor_with_config()`.

Default `{}`

Environ `TOR_CFG`

Note: If provided, it will be parsed from a JSON encoded string.

The following constants are defined for internal usage:

`darc.proxy.tor._MNG_TOR: bool`

If manage Tor proxy through *darc*.

Default `True`

Environ `DARC_TOR`

`darc.proxy.tor._TOR_BS_FLAG: bool`

If the Tor proxy is bootstrapped.

`darc.proxy.tor._TOR_PROC: subprocess.Popen`

Tor proxy process running in the background.

`darc.proxy.tor._TOR_CTRL: stem.control.Controller`

Tor controller process (`stem.control.Controller`) running in the background.

`darc.proxy.tor._TOR_CONFIG: List[str]`

Tor bootstrap configuration for `stem.process.launch_tor_with_config()`.

1.10.13 ZeroNet Proxy

The `darc.proxy.zeronet` module contains the auxiliary functions around managing and processing the ZeroNet proxy.

`darc.proxy.zeronet._zeronet_bootstrap()`
ZeroNet bootstrap.

The bootstrap arguments are defined as `__ZERONET_ARGS`.

Raises `subprocess.CalledProcessError` – If the return code of `__ZERONET_PROC` is non-zero.

See also:

- `darc.proxy.zeronet.zeronet_bootstrap()`
- `darc.proxy.zeronet.BS_WAIT`
- `darc.proxy.zeronet.__ZERONET_BS_FLAG`
- `darc.proxy.zeronet.__ZERONET_PROC`

`darc.proxy.zeronet.zeronet_bootstrap()`
Bootstrap wrapper for ZeroNet.

The function will bootstrap the ZeroNet proxy. It will retry for `ZERONET_RETRY` times in case of failure.

Also, it will **NOT** re-bootstrap the proxy as is guaranteed by `__ZERONET_BS_FLAG`.

Warns `ZeroNetBootstrapFailed` – If failed to bootstrap ZeroNet proxy.

Raises `UnsupportedPlatform` – If the system is not supported, i.e. not macOS or Linux.

See also:

- `darc.proxy.zeronet._zeronet_bootstrap()`
- `darc.proxy.zeronet.ZERONET_RETRY`
- `darc.proxy.zeronet.__ZERONET_BS_FLAG`

The following constants are configuration through environment variables:

`darc.proxy.zeronet.ZERONET_PORT: int`
Port for ZeroNet proxy connection.

Default 43110

Environ `ZERONET_PORT`

`darc.proxy.zeronet.ZERONET_RETRY: int`
Retry times for ZeroNet bootstrap when failure.

Default 3

Environ `ZERONET_RETRY`

`darc.proxy.zeronet.BS_WAIT: float`
Time after which the attempt to start ZeroNet is aborted.

Default 90

Environ `ZERONET_WAIT`

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

`darc.proxy.zeronet.ZERONET_PATH: str`

Path to the ZeroNet project.

Default `/usr/local/src/zeronet`

Environ `ZERONET_PATH`

`darc.proxy.zeronet.ZERONET_ARGS: List[str]`

ZeroNet bootstrap arguments for `run.sh start`.

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

Default `''`

Environ `ZERONET_ARGS`

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

The following constants are defined for internal usage:

`darc.proxy.zeronet._MNG_ZERONET: bool`

If manage ZeroNet proxy through `darc`.

Default `True`

Environ `DARC_ZERONET`

`darc.proxy.zeronet._ZERONET_BS_FLAG: bool`

If the ZeroNet proxy is bootstrapped.

`darc.proxy.zeronet._ZERONET_PROC: subprocess.Popen`

ZeroNet proxy process running in the background.

`darc.proxy.zeronet._ZERONET_ARGS: List[str]`

ZeroNet proxy bootstrap arguments.

To tell the `darc` project which proxy settings to be used for the `requests.Session` objects and `WebDriver` objects, you can specify such information in the `darc.proxy.LINK_MAP` mapping dictionary.

`darc.proxy.LINK_MAP: DefaultDict[str, Tuple[types.FunctionType, types.FunctionType]]`

```
LINK_MAP = collections.defaultdict(
    lambda: (darc.requests.null_session, darc.selenium.null_driver),
    dict(
        tor=(darc.requests.tor_session, darc.selenium.tor_driver),
        i2p=(darc.requests.i2p_session, darc.selenium.i2p_driver),
    )
)
```

The mapping dictionary for proxy type to its corresponding `requests.Session` factory function and `WebDriver` factory function.

The fallback value is the no proxy `requests.Session` object (`null_session()`) and `WebDriver` object (`null_driver()`).

See also:

- `darc.requests` – `requests.Session` factory functions
- `darc.selenium` – `WebDriver` factory functions

1.11 Sites Customisation

As websites may have authentication requirements, etc., over its content, the `darc.sites` module provides sites customisation hooks to both `requests` and `selenium` crawling processes.

1.11.1 Default Hooks

The `darc.sites.default` module is the fallback for sites customisation.

`darc.sites.default.crawler(session, link)`

Default crawler hook.

Parameters

- **session** (`requests.Session`) – Session object with proxy settings.
- **link** (`darc.link.Link`) – Link object to be crawled.

Returns The final response object with crawled data.

Return type `requests.Response`

See also:

- `darc.crawl.crawler()`

`darc.sites.default.loader(driver, link)`

Default loader hook.

When loading, if `SE_WAIT` is a valid time lapse, the function will sleep for such time to wait for the page to finish loading contents.

Parameters

- **driver** (`selenium.webdriver.Chrome`) – Web driver object with proxy settings.
- **link** (`darc.link.Link`) – Link object to be loaded.

Returns The web driver object with loaded data.

Return type `selenium.webdriver.Chrome`

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.

See also:

- `darc.crawl.loader()`
- `darc.const.SE_WAIT`

```

# -*- coding: utf-8 -*-
"""Default Hooks
=====

The :mod:`darc.sites.default` module is the fallback for sites
customisation.

"""

import time

import darc.typing as typing
from darc.const import SE_WAIT
from darc.link import Link

def crawler(session: typing.Session, link: Link) -> typing.Response:
    """Default crawler hook.

    Args:
        session (requests.Session): Session object with proxy settings.
        link: Link object to be crawled.

    Returns:
        requests.Response: The final response object with crawled data.

    See Also:
        * :func:`darc.crawl.crawler`

    """
    response = session.get(link.url, allow_redirects=True)
    return response

def loader(driver: typing.Driver, link: Link) -> typing.Driver:
    """Default loader hook.

    When loading, if :data:`~darc.const.SE_WAIT` is a valid time lapse,
    the function will sleep for such time to wait for the page to finish
    loading contents.

    Args:
        driver (selenium.webdriver.Chrome): Web driver object with proxy settings.
        link: Link object to be loaded.

    Returns:
        selenium.webdriver.Chrome: The web driver object with loaded data.

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

        .. |event| replace:: ``DOMContentLoaded``
        .. _event: https://developer.mozilla.org/en-US/docs/Web/API/Window/
        ↪ DOMContentLoaded_event

```

(continues on next page)

(continued from previous page)

```
See Also:
* :func:`darc.crawl.loader`
* :data:`darc.const.SE_WAIT`

"""
driver.get(link.url)

# wait for page to finish loading
if SE_WAIT is not None:
    time.sleep(SE_WAIT)

return driver
```

1.11.2 Bitcoin Addresses

The `darc.sites.bitcoin` module is customised to handle bitcoin addresses.

`darc.sites.bitcoin.crawler` (*session*, *link*)

Crawler hook for bitcoin addresses.

Parameters

- **session** (`requests.Session`) – Session object with proxy settings.
- **link** (`darc.link.Link`) – Link object to be crawled.

Raises `LinkNoReturn` – This link has no return response.

Return type NoReturn

`darc.sites.bitcoin.loader` (*driver*, *link*)

Not implemented.

Raises `LinkNoReturn` – This hook is not implemented.

Parameters

- **driver** (`selenium.webdriver.chrome.webdriver.WebDriver`) –
- **link** (`darc.link.Link`) –

Return type NoReturn

1.11.3 Data URI Schemes

The `darc.sites.data` module is customised to handle data URI schemes.

`darc.sites.data.crawler` (*session*, *link*)

Crawler hook for data URIs.

Parameters

- **session** (`requests.Session`) – Session object with proxy settings.
- **link** (`darc.link.Link`) – Link object to be crawled.

Raises `LinkNoReturn` – This link has no return response.

Return type NoReturn

`darc.sites.data.loader` (*driver*, *link*)

Not implemented.

Raises `LinkNoReturn` – This hook is not implemented.

Parameters

- **driver** (`selenium.webdriver.chrome.webdriver.WebDriver`) –
- **link** (`darc.link.Link`) –

Return type NoReturn

1.11.4 ED2K Magnet Links

The `darc.sites.ed2k` module is customised to handle ED2K magnet links.

`darc.sites.ed2k.crawler` (*session*, *link*)

Crawler hook for ED2K magnet links.

Parameters

- **session** (`requests.Session`) – Session object with proxy settings.
- **link** (`darc.link.Link`) – Link object to be crawled.

Raises `LinkNoReturn` – This link has no return response.

Return type NoReturn

`darc.sites.ed2k.loader` (*driver*, *link*)

Not implemented.

Raises `LinkNoReturn` – This hook is not implemented.

Parameters

- **driver** (`selenium.webdriver.chrome.webdriver.WebDriver`) –
- **link** (`darc.link.Link`) –

Return type NoReturn

1.11.5 IRC Addresses

The `darc.sites.script` module is customised to handle IRC addresses.

`darc.sites.irc.crawler` (*session*, *link*)

Crawler hook for IRC addresses.

Parameters

- **session** (`requests.Session`) – Session object with proxy settings.
- **link** (`darc.link.Link`) – Link object to be crawled.

Raises `LinkNoReturn` – This link has no return response.

Return type NoReturn

`darc.sites.irc.loader` (*driver*, *link*)

Not implemented.

Raises `LinkNoReturn` – This hook is not implemented.

Parameters

- **driver** (`selenium.webdriver.chrome.webdriver.WebDriver`) –
- **link** (`darc.link.Link`) –

Return type NoReturn

1.11.6 Magnet Links

The `darc.sites.magnet` module is customised to handle magnet links.

`darc.sites.magnet.crawler` (*session, link*)

Crawler hook for magnet links.

Parameters

- **session** (`requests.Session`) – Session object with proxy settings.
- **link** (`darc.link.Link`) – Link object to be crawled.

Raises `LinkNoReturn` – This link has no return response.

Return type NoReturn

`darc.sites.magnet.loader` (*driver, link*)

Not implemented.

Raises `LinkNoReturn` – This hook is not implemented.

Parameters

- **driver** (`selenium.webdriver.chrome.webdriver.WebDriver`) –
- **link** (`darc.link.Link`) –

Return type NoReturn

1.11.7 Email Addresses

The `darc.sites.mail` module is customised to handle email addresses.

`darc.sites.mail.crawler` (*session, link*)

Crawler hook for email addresses.

Parameters

- **session** (`requests.Session`) – Session object with proxy settings.
- **link** (`darc.link.Link`) – Link object to be crawled.

Raises `LinkNoReturn` – This link has no return response.

Return type NoReturn

`darc.sites.mail.loader` (*driver, link*)

Not implemented.

Raises `LinkNoReturn` – This hook is not implemented.

Parameters

- **driver** (`selenium.webdriver.chrome.webdriver.WebDriver`) –
- **link** (`darc.link.Link`) –

Return type NoReturn

1.11.8 JavaScript Links

The `darc.sites.script` module is customised to handle JavaScript links.

`darc.sites.script.crawler` (*session*, *link*)

Crawler hook for JavaScript links.

Parameters

- **session** (`requests.Session`) – Session object with proxy settings.
- **link** (`darc.link.Link`) – Link object to be crawled.

Raises `LinkNoReturn` – This link has no return response.

Return type NoReturn

`darc.sites.script.loader` (*driver*, *link*)

Not implemented.

Raises `LinkNoReturn` – This hook is not implemented.

Parameters

- **driver** (`selenium.webdriver.chrome.webdriver.WebDriver`) –
- **link** (`darc.link.Link`) –

Return type NoReturn

1.11.9 Telephone Numbers

The `darc.sites.tel` module is customised to handle telephone numbers.

`darc.sites.tel.crawler` (*session*, *link*)

Crawler hook for telephone numbers.

Parameters

- **session** (`requests.Session`) – Session object with proxy settings.
- **link** (`darc.link.Link`) – Link object to be crawled.

Raises `LinkNoReturn` – This link has no return response.

Return type NoReturn

`darc.sites.tel.loader` (*driver*, *link*)

Not implemented.

Raises `LinkNoReturn` – This hook is not implemented.

Parameters

- **driver** (`selenium.webdriver.chrome.webdriver.WebDriver`) –
- **link** (`darc.link.Link`) –

Return type NoReturn

To customise behaviours over `requests`, you sites customisation module should have a `crawler()` function, e.g. `crawler()`.

The function takes the `requests.Session` object with proxy settings and a `Link` object representing the link to be crawled, then returns a `requests.Response` object containing the final data of the crawling process.

`darc.sites.crawler_hook(link, session)`

Customisation as to `requests` sessions.

Parameters

- **link** (`darc.link.Link`) – Link object to be crawled.
- **session** (`requests.Session`) – Session object with proxy settings.

Returns The final response object with crawled data.

Return type `requests.Response`

See also:

- `darc.sites.SITE_MAP`
- `darc.sites._get_spec()`
- `darc.crawl.crawler()`

To customise behaviours over `selenium`, you sites customisation module should have a `loader()` function, e.g. `loader()`.

The function takes the `WebDriver` object with proxy settings and a `Link` object representing the link to be loaded, then returns the `WebDriver` object containing the final data of the loading process.

`darc.sites.loader_hook(link, driver)`

Customisation as to `selenium` drivers.

Parameters

- **link** (`darc.link.Link`) – Link object to be loaded.
- **driver** (`selenium.webdriver.Chrome`) – Web driver object with proxy settings.

Returns The web driver object with loaded data.

Return type `selenium.webdriver.Chrome`

See also:

- `darc.sites.SITE_MAP`
- `darc.sites._get_spec()`
- `darc.crawl.loader()`

To tell the `darc` project which sites customisation module it should use for a certain hostname, you can register such module to the `SITEMAP` mapping dictionary.

`darc.sites.SITEMAP: DefaultDict[str, str]`

```
SITEMAP = collections.defaultdict(lambda: 'default', {
    # 'www.sample.com': 'sample', # local customised module
})
```

The mapping dictionary for hostname to sites customisation modules.

The fallback value is `darc.sites.default`.

`darc.sites._get_spec(link)`

Load spec if any.

If the sites customisation failed to import, it will fallback to the default hooks, `default`.

Parameters `link` (`darc.link.Link`) – Link object to fetch sites customisation module.

Returns The sites customisation module.

Return type `types.ModuleType`

Warns `SiteNotFoundWarning` – If the sites customisation failed to import.

See also:

- `darc.sites.SITEMAP`

1.12 Module Constants

1.12.1 Auxiliary Function

`darc.const.getpid()`

Get process ID.

The process ID will be saved under the `PATH_DB` folder, in a file named `darc.pid`. If no such file exists, `-1` will be returned.

Returns The process ID.

Return type `int`

See also:

- `darc.const.PATH_ID`

`darc.const.get_lock()`

Get a lock.

Returns Lock context based on `FLAG_MP` and `FLAG_TH`.

Return type `Union[multiprocessing.context.BaseContext.Lock, _thread.allocate_lock, contextlib.nullcontext]`

1.12.2 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.

This can be useful especially when the capacity is limited and you wish to save some space before continuing next round. See *Docker integration* for more information.

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 direcorey.

`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`

1.12.3 Data Storage

See also:

See `darc.db` for more information about database integration.

`darc.const.REDIS: redis.Redis`

URL to the Redis database.

Default `redis://127.0.0.1`

Environ `REDIS_URL`

`darc.const.DB: peewee.Database`

URL to the RDS storage.

Default `sqlite://{PATH_DB}/darc.db`

Environ `:envvar`DB_URL``

`darc.const.DB: peewee.Database`

URL to the data submission storage.

Default `sqlite://{PATH_DB}/darcweb.db`

Environ `:envvar`DB_URL``

`darc.const.FLAG_DB: bool`

Flag if uses RDS as the task queue backend. If `REDIS_URL` is provided, then `False`; else, `True`.

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

See also:

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

1.12.4 Web Crawlers

`darc.const.DARC_WAIT: Optional[float]`

Time interval between each round when the `requests` and/or `selenium` database are empty.

Default 60

Environ `DARC_WAIT`

`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 long 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()`

1.12.5 White / Black Lists

`darc.const.LINK_WHITE_LIST: List[re.Pattern]`
 White list of hostnames should be crawled.

Default []

Environ `LINK_WHITE_LIST`

Note: Regular expressions are supported.

`darc.const.LINK_BLACK_LIST: List[re.Pattern]`
 Black list of hostnames should be crawled.

Default []

Environ `LINK_BLACK_LIST`

Note: Regular expressions are supported.

`darc.const.LINK_FALLBACK: bool`
 Fallback value for `match_host()`.

Default `False`

Environ `LINK_FALLBACK`

`darc.const.MIME_WHITE_LIST: List[re.Pattern]`
 White list of content types should be crawled.

Default []

Environ `MIME_WHITE_LIST`

Note: Regular expressions are supported.

`darc.const.MIME_BLACK_LIST: List[re.Pattern]`
 Black list of content types should be crawled.

Default []

Environ `MIME_BLACK_LIST`

Note: Regular expressions are supported.

`darc.const.MIME_FALLBACK: bool`
 Fallback value for `match_mime()`.

Default `False`

Environ `MIME_FALLBACK`

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

Default []

Environ `PROXY_WHITE_LIST`

Note: The proxy types are **case insensitive**.

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

Default `[]`

Environ `PROXY_BLACK_LIST`

Note: The proxy types are **case insensitive**.

`darc.const.PROXY_FALLBACK: bool`
Fallback value for `match_proxy()`.

Default `False`

Environ `PROXY_FALLBACK`

1.13 Custom Exceptions

The `render_error()` function can be used to render multi-line error messages with `stem.util.term` colours.

The `darc` project provides following custom exceptions:

- `LinkNoReturn`
- `UnsupportedLink`
- `UnsupportedPlatform`
- `UnsupportedProxy`

The `darc` project provides following custom exceptions:

- `TorBootstrapFailed`
- `I2PBootstrapFailed`
- `ZeroNetBootstrapFailed`
- `FreenetBootstrapFailed`
- `APIRequestFailed`
- `SiteNotFoundWarning`
- `LockWarning`
- `TorRenewFailed`
- `RedisCommandFailed`

exception `darc.error.APIRequestFailed`

Bases: `Warning`

API submit failed.

exception `darc.error.DatabaseOperationFailed`

Bases: `Warning`

Database operation execution failed.

exception `darc.error.FreenetBootstrapFailed`

Bases: `Warning`

Freenet bootstrap process failed.

exception `darc.error.I2PBootstrapFailed`

Bases: `Warning`

I2P bootstrap process failed.

exception `darc.error.LinkNoReturn`

Bases: `Exception`

The link has no return value from the hooks.

exception `darc.error.LockWarning`

Bases: `Warning`

Failed to acquire Redis lock.

exception `darc.error.RedisCommandFailed`

Bases: `Warning`

Redis command execution failed.

exception `darc.error.SiteNotFoundWarning`

Bases: `ImportWarning`

Site customisation not found.

exception `darc.error.TorBootstrapFailed`

Bases: `Warning`

Tor bootstrap process failed.

exception `darc.error.TorRenewFailed`

Bases: `Warning`

Tor renew request failed.

exception `darc.error.UnsupportedLink`

Bases: `Exception`

The link is not supported.

exception `darc.error.UnsupportedPlatform`

Bases: `Exception`

The platform is not supported.

exception `darc.error.UnsupportedProxy`

Bases: `Exception`

The proxy is not supported.

exception `darc.error.ZeroNetBootstrapFailed`

Bases: `Warning`

ZeroNet bootstrap process failed.

`darc.error.render_error` (*message*, *colour*)

Render error message.

The function wraps the `stem.util.term.format()` function to provide multi-line formatting support.

Parameters

- **message** (*str*) – Multi-line message to be rendered with `colour`.
- **colour** (*stem.util.term.Color*) – Front colour of text, c.f. `stem.util.term.Color`.

Returns The rendered error message.

Return type `str`

1.14 Data Models

The `darc.model` module contains all data models defined for the `darc` project, including RDS-based task queue and data submission.

1.14.1 Task Queues

The `darc.model.tasks` module defines the data models required for the task queue of `darc`.

See also:

Please refer to `darc.db` module for more information about the task queues.

Hostname Queue

Important: The hostname queue is a **set** named `queue_hostname` in a `Redis` based task queue.

The `darc.model.tasks.hostname` model contains the data model defined for the hostname queue.

```
class darc.model.tasks.hostname.HostnameQueueModel (*args, **kwargs)
```

```
    Bases: darc.model.abc.BaseModel
```

```
    Hostname task queue.
```

```
    DoesNotExist
```

```
        alias of HostnameQueueModelDoesNotExist
```

```
    hostname: Union[str, peewee.TextField] = <TextField: HostnameQueueModel.hostname>  
        Hostname (c.f. link.host).
```

```
    id = <AutoField: HostnameQueueModel.id>
```

```
    timestamp: Union[datetime.datetime, peewee.DateTimeField] = <DateTimeField: HostnameQueueModel.timestamp>  
        Timestamp of last update.
```

Crawler Queue

Important: The `crawler` queue is a **sorted set** named `queue_requests` in a `Redis` based task queue.

The `darc.model.tasks.requests` model contains the data model defined for the `crawler` queue.

```
class darc.model.tasks.requests.RequestsQueueModel (*args, **kwargs)
```

```
    Bases: darc.model.abc.BaseModel
```

```
    Task queue for crawler().
```

DoesNotExist

alias of RequestsQueueModelDoesNotExist

hash: Union[**str**, peewee.CharField] = <CharField: RequestsQueueModel.hash>
 Sha256 hash value (c.f. [Link.name](#)).

id = <AutoField: RequestsQueueModel.id>

link: Union[[darc.link.Link](#), [darc.model.utils.PickleField](#)] = <PickleField: RequestsQueueModel.link>
 Pickled target [Link](#) instance.

text: Union[**str**, peewee.TextField] = <TextField: RequestsQueueModel.text>
 URL as raw text (c.f. [Link.url](#)).

timestamp: Union[[datetime.datetime](#), peewee.DateTimeField] = <DateTimeField: RequestsQueueModel.timestamp>
 Timestamp of last update.

Loader Queue

Important: The *loader* queue is a **sorted set** named `queue_selenium` in a [Redis](#) based task queue.

The `darc.model.tasks.selenium` model contains the data model defined for the *loader* queue.

```
class darc.model.tasks.selenium.SeleniumQueueModel (*args, **kwargs)
```

Bases: `darc.model.abc.BaseModel`

Task queue for `loader()`.

DoesNotExist

alias of SeleniumQueueModelDoesNotExist

hash: Union[**str**, peewee.CharField] = <CharField: SeleniumQueueModel.hash>
 Sha256 hash value (c.f. [Link.name](#)).

id = <AutoField: SeleniumQueueModel.id>

link: Union[[darc.link.Link](#), [darc.model.utils.PickleField](#)] = <PickleField: SeleniumQueueModel.link>
 Pickled target [Link](#) instance.

text: Union[**str**, peewee.TextField] = <TextField: SeleniumQueueModel.text>
 URL as raw text (c.f. [Link.url](#)).

timestamp: Union[[datetime.datetime](#), peewee.DateTimeField] = <DateTimeField: SeleniumQueueModel.timestamp>
 Timestamp of last update.

1.14.2 Submission Data Models

The `darc.model.web` module defines the data models to store the data crawled from the *darc* project.

See also:

Please refer to `darc.submit` module for more information about data submission.

Hostname Records

The `darc.model.web.hostname` module defines the data model representing hostnames, specifically from `new_host` submission.

See also:

Please refer to `darc.submit.submit_new_host()` for more information.

class `darc.model.web.hostname.HostnameModel(*args, **kwargs)`

Bases: `darc.model.abc.BaseModelWeb`

Data model for a hostname record.

Important: The *alive* of a hostname is toggled if `crawler()` successfully requested a URL with such hostname.

DoesNotExist

alias of `HostnameModelDoesNotExist`

alive

If the hostname is still active.

We consider the hostname as *inactive*, only if all subsidiary URLs are *inactive*.

discovery: `datetime.datetime` = `<DateTimeField: HostnameModel.discovery>`

Timestamp of first `new_host` submission.

hostname: `str` = `<TextField: HostnameModel.hostname>`

Hostname (c.f. `link.host`).

hosts

id = `<AutoField: HostnameModel.id>`

last_seen: `datetime.datetime` = `<DateTimeField: HostnameModel.last_seen>`

Timestamp of last related submission.

proxy: `darc.model.utils.Proxy` = `<IntEnumField: HostnameModel.proxy>`

Proxy type (c.f. `link.proxy`).

robots

since

The hostname is active/inactive since such timestamp.

We consider the timestamp by the earliest timestamp of related subsidiary *active/inactive* URLs.

sitemaps

urls

URL Records

The `darc.model.web.url` module defines the data model representing URLs, specifically from `requests` and `selenium` submission.

See also:

Please refer to `darc.submit.submit_requests()` and `darc.submit.submit_selenium()` for more information.

class `darc.model.web.url.URLModel(*args, **kwargs)`

Bases: `darc.model.abc.BaseModelWeb`

Data model for a requested URL.

Important: The *alive* of a URL is toggled if `crawler()` successfully requested such URL and the status code is ok.

DoesNotExist

alias of `URLModelDoesNotExist`

alive: `bool` = `<BooleanField: URLModel.alive>`

If the hostname is still active.

discovery: `datetime.datetime` = `<DateTimeField: URLModel.discovery>`

Timestamp of first submission.

hash: `str` = `<CharField: URLModel.hash>`

Sha256 hash value (c.f. `Link.name`).

hostname: `darc.model.web.hostname.HostnameModel` = `<ForeignKeyField: URLModel.hostname>`

Hostname (c.f. `link.host`).

hostname_id = `<ForeignKeyField: URLModel.hostname>`

id = `<AutoField: URLModel.id>`

last_seen: `datetime.datetime` = `<DateTimeField: URLModel.last_seen>`

Timestamp of last submission.

proxy: `darc.model.utils.Proxy` = `<IntEnumField: URLModel.proxy>`

Proxy type (c.f. `link.proxy`).

requests

selenium

since: `datetime.datetime` = `<DateTimeField: URLModel.since>`

The hostname is active/inactive since this timestamp.

url: `str` = `<TextField: URLModel.url>`

Original URL (c.f. `link.url`).

robots.txt Records

The `darc.model.web.robots` module defines the data model representing `robots.txt` data, specifically from `new_host` submission.

See also:

Please refer to `darc.submit.submit_new_host()` for more information.

```
class darc.model.web.robots.RobotsModel(*args, **kwargs)
    Bases: darc.model.abc.BaseModelWeb

    Data model for robots.txt data.

    DoesNotExist
        alias of RobotsModelDoesNotExist

    document: str = <TextField: RobotsModel.document>
        Document data as str.

    host: darc.model.web.hostname.HostnameModel = <ForeignKeyField: RobotsModel.host>
        Hostname (c.f. link.host).

    host_id = <ForeignKeyField: RobotsModel.host>

    id = <AutoField: RobotsModel.id>

    timestamp: datetime.datetime = <DateTimeField: RobotsModel.timestamp>
        Timestamp of the submission.
```

sitemap.xml Records

The `darc.model.web.sitemap` module defines the data model representing `sitemap.xml` data, specifically from `new_host` submission.

See also:

Please refer to `darc.submit.submit_new_host()` for more information.

```
class darc.model.web.sitemap.SitemapModel(*args, **kwargs)
    Bases: darc.model.abc.BaseModelWeb

    Data model for sitemap.xml data.

    DoesNotExist
        alias of SitemapModelDoesNotExist

    document: str = <TextField: SitemapModel.document>
        Document data as str.

    host: darc.model.web.hostname.HostnameModel = <ForeignKeyField: SitemapModel.host>
        Hostname (c.f. link.host).

    host_id = <ForeignKeyField: SitemapModel.host>

    id = <AutoField: SitemapModel.id>

    timestamp: datetime.datetime = <DateTimeField: SitemapModel.timestamp>
        Timestamp of the submission.
```

hosts.txt Records

The `darc.model.web.hosts` module defines the data model representing `hosts.txt` data, specifically from `new_host` submission.

See also:

Please refer to `darc.submit.submit_new_host()` for more information.

```
class darc.model.web.hosts.HostsModel(*args, **kwargs)
    Bases: darc.model.abc.BaseModelWeb

    Data model for hosts.txt data.

    DoesNotExist
        alias of HostsModelDoesNotExist

    document: str = <TextField: HostsModel.document>
        Document data as str.

    host: darc.model.web.hostname.HostnameModel = <ForeignKeyField: HostsModel.host>
        Hostname (c.f. link.host).

    host_id = <ForeignKeyField: HostsModel.host>

    id = <AutoField: HostsModel.id>

    timestamp: datetime.datetime = <DateTimeField: HostsModel.timestamp>
        Timestamp of the submission.
```

Crawler Records

The `darc.model.web.requests` module defines the data model representing `crawler`, specifically from `requests` submission.

See also:

Please refer to `darc.submit.submit_requests()` for more information.

```
class darc.model.web.requests.RequestsHistoryModel(*args, **kwargs)
    Bases: darc.model.abc.BaseModelWeb

    Data model for history records from requests submission.

    DoesNotExist
        alias of RequestsHistoryModelDoesNotExist

    cookies: Dict[str, str] = <JSONField: RequestsHistoryModel.cookies>
        Response cookies.

    document: bytes = <BlobField: RequestsHistoryModel.document>
        Document data as bytes.

    id = <AutoField: RequestsHistoryModel.id>

    index: int = <IntegerField: RequestsHistoryModel.index>
        History index number.

    method: str = <CharField: RequestsHistoryModel.method>
        Request method (normally GET).

    model: darc.model.web.requests.RequestsModel = <ForeignKeyField: RequestsHistoryModel>
        Original record.
```

```
model_id = <ForeignKeyField: RequestsHistoryModel.model>
reason: str = <TextField: RequestsHistoryModel.reason>
    Response reason string.
request: Dict[str, str] = <JSONField: RequestsHistoryModel.request>
    Request headers.
response: Dict[str, str] = <JSONField: RequestsHistoryModel.response>
    Response headers.
status_code: int = <IntegerField: RequestsHistoryModel.status_code>
    Status code.
timestamp: datetime.datetime = <DateTimeField: RequestsHistoryModel.timestamp>
    Timestamp of the submission.
url: str = <TextField: RequestsHistoryModel.url>
    Request URL.

class darc.model.web.requests.RequestsModel(*args, **kwargs)
    Bases: darc.model.abc.BaseModelWeb
    Data model for documents from requests submission.

    DoesNotExist
        alias of RequestsModelDoesNotExist

    cookies: Dict[str, str] = <JSONField: RequestsModel.cookies>
        Response cookies.

    document: bytes = <BlobField: RequestsModel.document>
        Document data as bytes.

    history

    id = <AutoField: RequestsModel.id>

    is_html: bool = <BooleanField: RequestsModel.is_html>
        If document is HTML or miscellaneous data.

    method: str = <CharField: RequestsModel.method>
        Request method (normally GET).

    mime_type: str = <CharField: RequestsModel.mime_type>
        Content type.

    reason: str = <TextField: RequestsModel.reason>
        Response reason string.

    request: Dict[str, str] = <JSONField: RequestsModel.request>
        Request headers.

    response: Dict[str, str] = <JSONField: RequestsModel.response>
        Response headers.

    session: Dict[str, str] = <JSONField: RequestsModel.session>
        Session cookies.

    status_code: int = <IntegerField: RequestsModel.status_code>
        Status code.

    timestamp: datetime.datetime = <DateTimeField: RequestsModel.timestamp>
        Timestamp of the submission.
```



```
url: darc.model.web.url.URLModel = <ForeignKeyField: RequestsModel.url>
    Original URL (c.f. link.url).

url_id = <ForeignKeyField: RequestsModel.url>
```

Loader Records

The `darc.model.web.selenium` module defines the data model representing *loader*, specifically from selenium submission.

See also:

Please refer to `darc.submit.submit_selenium()` for more information.

```
class darc.model.web.selenium.SeleniumModel(*args, **kwargs)
    Bases: darc.model.abc.BaseModelWeb

    Data model for documents from selenium submission.

    DoesNotExist
        alias of SeleniumModelDoesNotExist

    document: str = <TextField: SeleniumModel.document>
        Document data as str.

    id = <AutoField: SeleniumModel.id>

    screenshot: Optional[bytes] = <BlobField: SeleniumModel.screenshot>
        Screenshot in PNG format as bytes.

    timestamp: datetime.datetime = <DateTimeField: SeleniumModel.timestamp>
        Timestamp of the submission.

    url: darc.model.web.url.URLModel = <ForeignKeyField: SeleniumModel.url>
        Original URL (c.f. link.url).

    url_id = <ForeignKeyField: SeleniumModel.url>
```

1.14.3 Base Model

The `darc.model.abc` module contains abstract base class of all data models for the *darc* project.

```
class darc.model.abc.BaseMeta
    Bases: object

    Basic metadata for data models.

    table_function()
        Generate table name dynamically (c.f. table_function()).

        Parameters model_class (peewee.Model) –
        Return type str

    database = <peewee.SqliteDatabase object>
        Reference database storage (c.f. DB).

class darc.model.abc.BaseMetaWeb
    Bases: darc.model.abc.BaseMeta

    Basic metadata for data models of data submission.
```

database = <peewee.SqliteDatabase object>

Reference database storage (c.f. *DB*).

class darc.model.abc.BaseModel (*args, **kwargs)

Bases: peewee.Model

Base model with standard patterns.

Notes

The model will implicitly have a `AutoField` attribute named as *id*.

DoesNotExist

alias of BaseModelDoesNotExist

to_dict (*keep_id=False*)

Convert record to *dict*.

Parameters *keep_id* (*bool*) – If keep the ID auto field.

Returns The data converted through `playhouse.shortcuts.model_to_dict()`.

Meta

Basic metadata for data models.

id = <AutoField: BaseModel.id>

class darc.model.abc.BaseModelWeb (*args, **kwargs)

Bases: *darc.model.abc.BaseModel*

Base model with standard patterns for data submission.

Notes

The model will implicitly have a `AutoField` attribute named as *id*.

DoesNotExist

alias of BaseModelWebDoesNotExist

Meta

Basic metadata for data models.

id = <AutoField: BaseModelWeb.id>

1.14.4 Miscellaneous Utilities

The *darc.model.utils* module contains several miscellaneous utility functions and data fields.

class darc.model.utils.IPField (*null=False, index=False, unique=False, column_name=None, default=None, primary_key=False, constraints=None, sequence=None, collation=None, unindexed=False, choices=None, help_text=None, verbose_name=None, index_type=None, db_column=None, _hidden=False*)

Bases: peewee.IPField

IP data field.

db_value (*val*)

Dump the value for database storage.

Parameters

- **value** – Source IP address instance.
- **val** (*Optional[Union[str, ipaddress.IPv4Address, ipaddress.IPv6Address]]*) –

Returns Integral representation of the IP address.

Return type *Optional[int]*

python_value (*val*)

Load the value from database storage.

Parameters

- **value** – Integral representation of the IP address.
- **val** (*Optional[int]*) –

Returns Original IP address instance.

Return type *Optional[Union[ipaddress.IPv4Address, ipaddress.IPv6Address]]*

```
class darc.model.utils.IntEnumField (null=False, index=False, unique=False, column_name=None, default=None, primary_key=False, constraints=None, sequence=None, collation=None, unindexed=False, choices=None, help_text=None, verbose_name=None, index_type=None, db_column=None, _hidden=False)
```

Bases: *peewee.IntegerField*

enum.IntEnum data field.

python_value (*value*)

Load the value from database storage.

Parameters **value** (*Optional[int]*) – Integral representation of the enumeration.

Returns Original enumeration object.

Return type *Optional[enum.IntEnum]*

choices: *enum.IntEnum*

The original *enum.IntEnum* class.

```
class darc.model.utils.JSONField (null=False, index=False, unique=False, column_name=None, default=None, primary_key=False, constraints=None, sequence=None, collation=None, unindexed=False, choices=None, help_text=None, verbose_name=None, index_type=None, db_column=None, _hidden=False)
```

Bases: *playhouse.mysql_ext.JSONField*

JSON data field.

db_value (*value*)

Dump the value for database storage.

Parameters **value** (*Any*) – Source JSON value.

Returns JSON serialised string data.

Return type *Optional[str]*

python_value (*value*)

Load the value from database storage.

Parameters **value** (*Optional[str]*) – Serialised JSON string.

Returns Original JSON data.

Return type Any

```
class darc.model.utils.PickleField(null=False, index=False, unique=False, column_name=None, default=None, primary_key=False, constraints=None, sequence=None, collation=None, unindexed=False, choices=None, help_text=None, verbose_name=None, index_type=None, db_column=None, _hidden=False)
```

Bases: `peewee.BlobField`

Pickled data field.

db_value (*value*)

Dump the value for database storage.

Parameters **value** (*Any*) – Source value.

Returns Picked bytestring data.

Return type `Optional[bytes]`

python_value (*value*)

Load the value from database storage.

Parameters **value** (*Optional[bytes]*) – SPicked bytestring data.

Returns Original data.

Return type Any

```
class darc.model.utils.Proxy(value)
```

Bases: `enum.IntEnum`

Proxy types supported by *darc*.

FREENET = 5

Freenet proxy.

I2P = 3

I2P proxy.

NULL = 1

No proxy.

TOR = 2

Tor proxy.

TOR2WEB = 6

//onion.sh/>`__, no proxy).

Type Proxied Tor (tor2web <https

ZERONET = 4

ZeroNet proxy.

```
darc.model.utils.table_function(model_class)
```

Generate table name dynamically.

The function strips `Model` from the class name and calls `peewee.make_snake_case()` to generate a proper table name.

Parameters `model_class` (`peewee.Model`) – Data model class.

Returns Generated table name.

Return type `str`

As the websites can be sometimes irritating for their anti-robots verification, login requirements, etc., the `darc` project also provides 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 army knife, only the main entrypoint function `darc.process.process()` is exported in global namespace (and renamed to `darc.darc()`), see below:

`darc.darc(worker)`

Main process.

The function will register `_signal_handler()` for SIGTERM, and start the main process of the `darc` darkweb crawlers.

Parameters `worker` (`Literal[crawler, loader]`) – Worker process type.

Raises `ValueError` – If `worker` is not a valid value.

Before starting the workers, the function will start proxies through

- `darc.proxy.tor.tor_proxy()`
- `darc.proxy.i2p.i2p_proxy()`
- `darc.proxy.zeronet.zeronet_proxy()`
- `darc.proxy.freenet.freenet_proxy()`

The general process can be described as following for `workers` of `crawler` type:

1. `process_crawler()`: obtain URLs from the `requests` link database (c.f. `load_requests()`), and feed such URLs to `crawler()`.

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.

2. `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`), `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()`).

The general process can be described as following for *workers* of loader type:

1. `process_loader()`: in the meanwhile, *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.

2. `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 Chrome object.

If successful, the rendered source HTML document will be saved, 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()`).

If in reboot mode, i.e. `REBOOT` is `True`, the function will exit after first round. If not, it will renew the Tor connections (if bootstrapped), c.f. `renew_tor_session()`, and start another round.

DOCKER INTEGRATION

The *darc* project is integrated with Docker and Compose. Though published to [Docker Hub](#), you can still build by yourself.

Important: The debug image contains miscellaneous documents, i.e. whole repository in it; and pre-installed some useful tools for debugging, such as IPython, etc.

The Docker image is based on [Ubuntu Bionic](#) (18.04 LTS), setting up all Python dependencies for the *darc* project, installing [Google Chrome](#) (version 79.0.3945.36) and corresponding [ChromeDriver](#), as well as installing and configuring [Tor](#), [I2P](#), [ZeroNet](#), [FreeNet](#), [NoIP](#) proxies.

Note: [NoIP](#) is currently not fully integrated in the *darc* due to misunderstanding in the configuration process. Contributions are welcome.

When building the image, there is an *optional* argument for setting up a *non-root* user, c.f. environment variable `DARC_USER` and module constant `DARC_USER`. By default, the username is *darc*.

```
FROM ubuntu:bionic

LABEL Name=darc \
      Version=0.6.9

STOPSIGNAL SIGINT
HEALTHCHECK --interval=1h --timeout=1m \
  CMD wget https://httpbin.org/get -O /dev/null || exit 1

ARG DARC_USER="darc"
ENV LANG="C.UTF-8" \
     LC_ALL="C.UTF-8" \
     PYTHONIOENCODING="UTF-8" \
     DEBIAN_FRONTEND="teletype" \
     DARC_USER="${DARC_USER}" \
     # DEBIAN_FRONTEND="noninteractive"

COPY extra/retry.sh /usr/local/bin/retry
COPY extra/install.py /usr/local/bin/pty-install
COPY vendor/jdk-11.0.8_linux-x64_bin.tar.gz /var/cache/oracle-jdk11-installer-local/

RUN set -x \
  && retry apt-get update \
  && retry apt-get install --yes --no-install-recommends \
```

(continues on next page)

(continued from previous page)

```

    apt-utils \
&& retry apt-get install --yes --no-install-recommends \
    gcc \
    g++ \
    libmagic1 \
    make \
    software-properties-common \
    tar \
    unzip \
    zlib1g-dev \
&& retry add-apt-repository ppa:deadsnakes/ppa --yes \
&& retry add-apt-repository ppa:linuxuprising/java --yes \
&& retry add-apt-repository ppa:i2p-maintainers/i2p --yes
RUN retry apt-get update \
&& retry apt-get install --yes --no-install-recommends \
    python3.8 \
    python3-pip \
    python3-setuptools \
    python3-wheel \
&& ln -sf /usr/bin/python3.8 /usr/local/bin/python3
RUN retry pty-install --stdin '6\n70' apt-get install --yes --no-install-recommends \
    tzdata \
&& retry pty-install --stdin 'yes' apt-get install --yes \
    oracle-java11-installer-local
RUN retry apt-get install --yes --no-install-recommends \
    sudo \
&& adduser --disabled-password --gecos '' ${DARC_USER} \
&& adduser ${DARC_USER} sudo \
&& echo '%sudo ALL=(ALL) NOPASSWD:ALL' >> /etc/sudoers

## Tor
RUN retry apt-get install --yes --no-install-recommends tor
COPY extra/torrc.bionic /etc/tor/torrc

## I2P
RUN retry apt-get install --yes --no-install-recommends i2p
COPY extra/i2p.bionic /etc/defaults/i2p

## ZeroNet
COPY vendor/ZeroNet-linux-dist-linux64.tar.gz /tmp
RUN set -x \
&& cd /tmp \
&& tar xvpfz ZeroNet-linux-dist-linux64.tar.gz \
&& mv ZeroNet-linux-dist-linux64 /usr/local/src/zeronet
COPY extra/zeronet.bionic.conf /usr/local/src/zeronet/zeronet.conf

## FreeNet
USER darc
COPY vendor/new_installer_offline.jar /tmp
RUN set -x \
&& cd /tmp \
&& ( pty-install --stdin '/home/darc/freenet\n1' java -jar new_installer_offline.jar \
→ || true ) \
&& sudo mv /home/darc/freenet /usr/local/src/freenet
USER root

## NoIP

```

(continues on next page)

(continued from previous page)

```

COPY vendor/noip-duc-linux.tar.gz /tmp
RUN set -x \
  && cd /tmp \
  && tar xvpfz noip-duc-linux.tar.gz \
  && mv noip-2.1.9-1 /usr/local/src/noip \
  && cd /usr/local/src/noip \
  && make
# && make install

# # set up timezone
# RUN echo 'Asia/Shanghai' > /etc/timezone \
# && rm -f /etc/localtime \
# && ln -snf /usr/share/zoneinfo/Asia/Shanghai /etc/localtime \
# && dpkg-reconfigure -f noninteractive tzdata

COPY vendor/chromedriver_linux64.zip \
  vendor/google-chrome-stable_current_amd64.deb /tmp/
RUN set -x \
  ## ChromeDriver
  && unzip -d /usr/bin /tmp/chromedriver_linux64.zip \
  && which chromedriver \
  ## Google Chrome
  && ( dpkg --install /tmp/google-chrome-stable_current_amd64.deb || true ) \
  && retry apt-get install --fix-broken --yes --no-install-recommends \
  && dpkg --install /tmp/google-chrome-stable_current_amd64.deb \
  && which google-chrome

# Using pip:
COPY requirements.txt /tmp
RUN python3 -m pip install -r /tmp/requirements.txt --no-cache-dir

RUN set -x \
  && rm -rf \
    ## APT repository lists
    /var/lib/apt/lists/* \
    ## Python dependencies
    /tmp/requirements.txt \
    /tmp/pip \
    ## ChromeDriver
    /tmp/chromedriver_linux64.zip \
    ## Google Chrome
    /tmp/google-chrome-stable_current_amd64.deb \
    ## Vendors
    /tmp/new_installer_offline.jar \
    /tmp/noip-duc-linux.tar.gz \
    /tmp/ZeroNet-linux-dist-linux64.tar.gz \
  &&& apt-get remove --auto-remove --yes \
  #   software-properties-common \
  #   unzip \
  && apt-get autoremove -y \
  && apt-get autoclean \
  && apt-get clean

ENTRYPOINT [ "python3", "-m", "darc" ]
#ENTRYPOINT [ "bash", "/app/run.sh" ]
CMD [ "--help" ]

```

(continues on next page)

(continued from previous page)

```

WORKDIR /app
COPY darc/ /app/darc/
COPY LICENSE \
      MANIFEST.in \
      README.rst \
      extra/run.sh \
      setup.cfg \
      setup.py \
      test_darc.py /app/
RUN python3 -m pip install -e .

```

Note:

- `retry` is a shell script for retrying the commands until success

```

#!/usr/bin/env bash

while true; do
    >&2 echo "+ $@"
    $@ && break
    >&2 echo "exit: $?"
done
>&2 echo "exit: 0"

```

- `pty-install` is a Python script simulating user input for APT package installation with `DEBIAN_FRONTEND` set as Teletype.

```

#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""Install packages requiring interactions."""

import argparse
import os
import subprocess
import sys
import tempfile

def get_parser():
    """Argument parser."""
    parser = argparse.ArgumentParser('install',
                                     description='pseudo-interactive package installer
→')

    parser.add_argument('-i', '--stdin', help='content for input')
    parser.add_argument('command', nargs=argparse.REMAINDER, help='command to execute
→')

    return parser

def main():
    """Entrypoint."""
    parser = get_parser()
    args = parser.parse_args()
    text = args.stdin.encode().decode('unicode_escape')

```

(continues on next page)

(continued from previous page)

```

path = tempfile.mktemp(prefix='install-')
with open(path, 'w') as file:
    file.write(text)

with open(path, 'r') as file:
    proc = subprocess.run(args.command, stdin=file) # pylint: disable=subprocess-
↳run-check

os.remove(path)
return proc.returncode

if __name__ == "__main__":
    sys.exit(main())

```

As always, you can also use Docker Compose to manage the `darc` image. Environment variables can be set as described in the `configuration` section.

```

version: '3'

services:
  crawler:
    image: jsnbzh/darc:latest
    build: &build
    context: .
    args:
      # non-root user
      DARC_USER: "darc"
    container_name: crawler
    #entrypoint: [ "bash", "/app/run.sh" ]
    command: [ "--type", "crawler",
               "--file", "/app/text/tor.txt",
               "--file", "/app/text/tor2web.txt",
               "--file", "/app/text/i2p.txt",
               "--file", "/app/text/zeronet.txt",
               "--file", "/app/text/freenet.txt" ]
    environment:
      ## [PYTHON] force the stdout and stderr streams to be unbuffered
      PYTHONUNBUFFERED: 1
      # reboot mode
      DARC_REBOOT: 0
      # debug mode
      DARC_DEBUG: 0
      # verbose mode
      DARC_VERBOSE: 1
      # force mode (ignore robots.txt)
      DARC_FORCE: 1
      # check mode (check proxy and hostname before crawling)
      DARC_CHECK: 1
      # check mode (check content type before crawling)
      DARC_CHECK_CONTENT_TYPE: 0
      # save mode
      DARC_SAVE: 0
      # save mode (for requests)

```

(continues on next page)

(continued from previous page)

```

DAVE_SAVE_REQUESTS: 0
# save mode (for selenium)
DAVE_SAVE_SELENIUM: 0
# processes
DARC_CPU: 16
# multiprocessing
DARC_MULTIPROCESSING: 1
# multithreading
DARC_MULTITHREADING: 0
# time lapse
DARC_WAIT: 60
# bulk size
DARC_BULK_SIZE: 1000
# data storage
PATH_DATA: "data"
# save data submitssion
SAVE_DB: 0
# Redis URL
REDIS_URL: 'redis://
↳:UCf7y123aHgaYeGnvLRasALjFfDVHGCz6KiR5Z0WC0DL4ExvSGw5SkcOxBywc0qtZBHVRsvx2QMGewXNP6qVow@redis
↳ '
# database URL
#DB_URL: 'mysql://
↳root:b8y9dpz3MJSQtwnZIW77ydASBOYfzA7HJfugv77wLrWQzrjCx5m3spoiqRi4kU52syYy2jxJZR3U2kwPkEVTA@db
↳ '
# max pool
DARC_MAX_POOL: 10
# Tor proxy & control port
TOR_PORT: 9050
TOR_CTRL: 9051
# Tor management method
TOR_STEM: 1
# Tor authentication
TOR_PASS: "16:B9D36206B5374B3F609045F9609EE670F17047D88FF713EFB9157EA39F"
# Tor bootstrap retry
TOR_RETRY: 10
# Tor bootstrap wait
TOR_WAIT: 90
# Tor bootstrap config
TOR_CFG: "{}"
# I2P port
I2P_PORT: 4444
# I2P bootstrap retry
I2P_RETRY: 10
# I2P bootstrap wait
I2P_WAIT: 90
# I2P bootstrap config
I2P_ARGS: ""
# ZeroNet port
ZERONET_PORT: 43110
# ZeroNet bootstrap retry
ZERONET_RETRY: 10
# ZeroNet project path
ZERONET_PATH: "/usr/local/src/zeronet"
# ZeroNet bootstrap wait
ZERONET_WAIT: 90
# ZeroNet bootstrap config

```

(continues on next page)

(continued from previous page)

```

ZERONET_ARGS: ""
# Freenet port
FRENET_PORT: 8888
# Freenet bootstrap retry
FRENET_RETRY: 0
# Freenet project path
FRENET_PATH: "/usr/local/src/freenet"
# Freenet bootstrap wait
FRENET_WAIT: 90
# Freenet bootstrap config
FRENET_ARGS: ""
# time delta for caches in seconds
TIME_CACHE: 2_592_000 # 30 days
# time to wait for selenium
SE_WAIT: 5
# extract link pattern
LINK_WHITE_LIST: '[
    ".*?\\.onion",
    ".*?\\.i2p", "127\\.0\\.0\\.1:7657", "localhost:7657", "127\\.0\\.0\\.1:7658",
↪ "localhost:7658",
    "127\\.0\\.0\\.1:43110", "localhost:43110",
    "127\\.0\\.0\\.1:8888", "localhost:8888"
]'
# link black list
LINK_BLACK_LIST: '[ "(.*\\.)?facebookcorewwi\\.onion", "(.*\\.)?
↪ nytimes3xbfgragh\\.onion" ]'
# link fallback flag
LINK_FALLBACK: 1
# content type white list
MIME_WHITE_LIST: '[ "text/html", "application/xhtml+xml" ]'
# content type black list
MIME_BLACK_LIST: '[ "text/css", "application/javascript", "text/json" ]'
# content type fallback flag
MIME_FALLBACK: 0
# proxy type white list
PROXY_WHITE_LIST: '[ "tor", "i2p", "freenet", "zeronet", "tor2web" ]'
# proxy type black list
PROXY_BLACK_LIST: '[ "null", "data" ]'
# proxy type fallback flag
PROXY_FALLBACK: 0
# API retry times
API_RETRY: 10
# API URLs
#API_NEW_HOST: 'https://example.com/api/new_host'
#API_REQUESTS: 'https://example.com/api/requests'
#API_SELENIUM: 'https://example.com/api/selenium'
restart: "always"
networks: &networks
- darc
volumes: &volumes
- ./text:/app/text
- ./extra:/app/extra
- /data/darc:/app/data

loader:
image: jsnbzh/darc:latest
build: *build

```

(continues on next page)

(continued from previous page)

```

container_name: loader
#entrypoint: [ "bash", "/app/run.sh" ]
command: [ "--type", "loader" ]
environment:
  ## [PYTHON] force the stdout and stderr streams to be unbuffered
  PYTHONUNBUFFERED: 1
  # reboot mode
  DARC_REBOOT: 0
  # debug mode
  DARC_DEBUG: 0
  # verbose mode
  DARC_VERBOSE: 1
  # force mode (ignore robots.txt)
  DARC_FORCE: 1
  # check mode (check proxy and hostname before crawling)
  DARC_CHECK: 1
  # check mode (check content type before crawling)
  DARC_CHECK_CONTENT_TYPE: 0
  # save mode
  DARC_SAVE: 0
  # save mode (for requests)
  DAVE_SAVE_REQUESTS: 0
  # save mode (for selenium)
  DAVE_SAVE_SELENIUM: 0
  # processes
  DARC_CPU: 1
  # multiprocessing
  DARC_MULTIPROCESSING: 0
  # multithreading
  DARC_MULTITHREADING: 0
  # time lapse
  DARC_WAIT: 60
  # data storage
  PATH_DATA: "data"
  # Redis URL
  REDIS_URL: 'redis://
↳UCf7y123aHgaYeGnvLRasALjFfDVHGCz6KiR5Z0WC0DL4ExvSGw5SkcOxBywc0qtZBHVrSVx2QMGewXNP6qVow@redis
↳'
  # database URL
  #DB_URL: 'mysql://
↳root:b8y9dpz3MJSQtwnZIW77ydASBOYfzA7HJfugv77wLrWQzrjCx5m3spoaigRi4kU52syYy2jxJZR3U2kwPkEVTA@db
↳'

  # max pool
  DARC_MAX_POOL: 10
  # save data submission
  SAVE_DB: 0
  # Tor proxy & control port
  TOR_PORT: 9050
  TOR_CTRL: 9051
  # Tor management method
  TOR_STEM: 1
  # Tor authentication
  TOR_PASS: "16:B9D36206B5374B3F609045F9609EE670F17047D88FF713EFB9157EA39F"
  # Tor bootstrap retry
  TOR_RETRY: 10
  # Tor bootstrap wait
  TOR_WAIT: 90

```

(continues on next page)

(continued from previous page)

```

# Tor bootstrap config
TOR_CFG: "{}"
# I2P port
I2P_PORT: 4444
# I2P bootstrap retry
I2P_RETRY: 10
# I2P bootstrap wait
I2P_WAIT: 90
# I2P bootstrap config
I2P_ARGS: ""
# ZeroNet port
ZERONET_PORT: 43110
# ZeroNet bootstrap retry
ZERONET_RETRY: 10
# ZeroNet project path
ZERONET_PATH: "/usr/local/src/zeronet"
# ZeroNet bootstrap wait
ZERONET_WAIT: 90
# ZeroNet bootstrap config
ZERONET_ARGS: ""
# Freenet port
FREENET_PORT: 8888
# Freenet bootstrap retry
FREENET_RETRY: 0
# Freenet project path
FREENET_PATH: "/usr/local/src/freenet"
# Freenet bootstrap wait
FREENET_WAIT: 90
# Freenet bootstrap config
FREENET_ARGS: ""
# time delta for caches in seconds
TIME_CACHE: 2_592_000 # 30 days
# time to wait for selenium
SE_WAIT: 5
# extract link pattern
LINK_WHITE_LIST: '[
    ".*?\\.onion",
    ".*?\\.i2p", "127\\.0\\.0\\.1:7657", "localhost:7657", "127\\.0\\.0\\.1:7658",
↪ "localhost:7658",
    "127\\.0\\.0\\.1:43110", "localhost:43110",
    "127\\.0\\.0\\.1:8888", "localhost:8888"
]'
# link black list
LINK_BLACK_LIST: '[ "(.*\\.)?facebookcorewwi\\.onion", "(.*\\.)?
↪nytimes3xbfgragh\\.onion" ]'
# link fallback flag
LINK_FALLBACK: 1
# content type white list
MIME_WHITE_LIST: '[ "text/html", "application/xhtml+xml" ]'
# content type black list
MIME_BLACK_LIST: '[ "text/css", "application/javascript", "text/json" ]'
# content type fallback flag
MIME_FALLBACK: 0
# proxy type white list
PROXY_WHITE_LIST: '[ "tor", "i2p", "freenet", "zeronet", "tor2web" ]'
# proxy type black list
PROXY_BLACK_LIST: '[ "null", "data" ]'

```

(continues on next page)

(continued from previous page)

```

# proxy type fallback flag
PROXY_FALLBACK: 0
# API retry times
API_RETRY: 10
# API URLs
#API_NEW_HOST: 'https://example.com/api/new_host'
#API_REQUESTS: 'https://example.com/api/requests'
#API_SELENIUM: 'https://example.com/api/selenium'
restart: "always"
networks: *networks
volumes: *volumes

# network settings
networks:
  darc:
    driver: bridge

```

Note: Should you wish to run *darc* in reboot mode, i.e. set *DARC_REBOOT* and/or *REBOOT* as *True*, you may wish to change the entrypoint to

```
bash /app/run.sh
```

where *run.sh* is a shell script wraps around *darc* especially for reboot mode.

```

#!/usr/bin/env bash

set -e

# time lapse
WAIT=${DARC_WAIT=10}

# signal handlers
trap '[ -f ${PATH_DATA}/darc.pid ] && kill -2 $(cat ${PATH_DATA}/darc.pid)' SIGINT_
↳SIGTERM SIGKILL

# initialise
echo "+ Starting application..."
python3 -m darc $@
sleep ${WAIT}

# mainloop
while true; do
  echo "+ Restarting application..."
  python3 -m darc
  sleep ${WAIT}
done

```

In such scenario, you can customise your *run.sh* to, for instance, archive then upload current data crawled by *darc* to somewhere else and save up some disk space.

WEB BACKEND DEMO

This is a demo of API for communication between the *darc* crawlers (*darc.submit*) and web UI.

See also:

Please refer to *data schema* for more information about the submission data.

Assuming the web UI is developed using the *Flask* microframework.

```
# -*- coding: utf-8 -*-

import sys

import flask # pylint: disable=import-error

# Flask application
app = flask.Flask(__file__)

@app.route('/api/new_host', methods=['POST'])
def new_host():
    """When a new host is discovered, the :mod:`darc` crawler will submit the
    host information. Such includes ``robots.txt`` (if exists) and
    ``sitemap.xml`` (if any).

    Data format::

        {
            // partial flag - true / false
            "$PARTIAL$": ...,
            // force flag - true / false
            "$FORCE$": ...,
            // metadata of URL
            "[metadata]": {
                // original URL - <scheme>://<netloc>/<path>;<params>?<query>#
                <fragment>
                "url": ...,
                // proxy type - null / tor / i2p / zeronet / freenet
                "proxy": ...,
                // hostname / netloc, c.f. ``urllib.parse.urlparse``
                "host": ...,
                // base folder, relative path (to data root path ``PATH_DATA``) in_
                <container> - <proxy>/<scheme>/<host>
                "base": ...,
                // sha256 of URL as name for saved files (timestamp is in ISO format)
                // JSON log as this one - <base>/<name>_<timestamp>.json
            }
        }
    """
```

(continues on next page)

(continued from previous page)

```

        // HTML from requests - <base>/<name>_<timestamp>_raw.html
        // HTML from selenium - <base>/<name>_<timestamp>.html
        // generic data files - <base>/<name>_<timestamp>.dat
        "name": ...
    },
    // requested timestamp in ISO format as in name of saved file
    "Timestamp": ...,
    // original URL
    "URL": ...,
    // robots.txt from the host (if not exists, then ``null``)
    "Robots": {
        // path of the file, relative path (to data root path ``PATH_DATA``)
→in container
        // - <proxy>/<scheme>/<host>/robots.txt
        "path": ...,
        // content of the file (**base64** encoded)
        "data": ...,
    },
    // sitemaps from the host (if none, then ``null``)
    "Sitemaps": [
        {
            // path of the file, relative path (to data root path ``PATH_
→DATA``) in container
            // - <proxy>/<scheme>/<host>/sitemap_<name>.xml
            "path": ...,
            // content of the file (**base64** encoded)
            "data": ...,
        },
        ...
    ],
    // hosts.txt from the host (if proxy type is ``i2p``; if not exists, then
→``null``)
    "Hosts": {
        // path of the file, relative path (to data root path ``PATH_DATA``)
→in container
        // - <proxy>/<scheme>/<host>/hosts.txt
        "path": ...,
        // content of the file (**base64** encoded)
        "data": ...,
    }
}

"""
# JSON data from the request
data = flask.request.json # pylint: disable=unused-variable

# do whatever processing needed
...

@app.route('/api/requests', methods=['POST'])
def from_requests():
    """When crawling, we'll first fetch the URL using ``requests``, to check
    its availability and to save its HTTP headers information. Such information
    will be submitted to the web UI.

    Data format::

```

(continues on next page)

(continued from previous page)

```

{
    // metadata of URL
    "[metadata]": {
        // original URL - <scheme>://<netloc>/<path>;<params>?<query>#
        <fragment>
        "url": ...,
        // proxy type - null / tor / i2p / zeronet / freenet
        "proxy": ...,
        // hostname / netloc, c.f. ``urllib.parse.urlparse``
        "host": ...,
        // base folder, relative path (to data root path ``PATH_DATA``) in
        <container> - <proxy>/<scheme>/<host>
        "base": ...,
        // sha256 of URL as name for saved files (timestamp is in ISO format)
        // JSON log as this one - <base>/<name>_<timestamp>.json
        // HTML from requests - <base>/<name>_<timestamp>_raw.html
        // HTML from selenium - <base>/<name>_<timestamp>.html
        // generic data files - <base>/<name>_<timestamp>.dat
        "name": ...
    },
    // requested timestamp in ISO format as in name of saved file
    "Timestamp": ...,
    // original URL
    "URL": ...,
    // request method
    "Method": "GET",
    // response status code
    "Status-Code": ...,
    // response reason
    "Reason": ...,
    // response cookies (if any)
    "Cookies": {
        ...
    },
    // session cookies (if any)
    "Session": {
        ...
    },
    // request headers (if any)
    "Request": {
        ...
    },
    // response headers (if any)
    "Response": {
        ...
    },
    // content type
    "Content-Type": ...,
    // requested file (if not exists, then ``null``)
    "Document": {
        // path of the file, relative path (to data root path ``PATH_DATA``)
        <in container>
        // - <proxy>/<scheme>/<host>/<name>_<timestamp>_raw.html
        // or if the document is of generic content type, i.e. not HTML
        // - <proxy>/<scheme>/<host>/<name>_<timestamp>.dat
        "path": ...,

```

(continues on next page)

(continued from previous page)

```

        // content of the file (**base64** encoded)
        "data": ...,
    },
    // redirection history (if any)
    "History": [
        // same records as the original response
        {"...": "..."}
    ]
}

"""
# JSON data from the request
data = flask.request.json # pylint: disable=unused-variable

# do whatever processing needed
...

@app.route('/api/selenium', methods=['POST'])
def from_selenium():
    """After crawling with ``requests``, we'll then render the URL using
    ``selenium`` with Google Chrome and its driver, to provide a fully rendered
    web page. Such information will be submitted to the web UI.

    Note:
        This information is optional, only provided if the content type from
        ``requests`` is HTML, status code < 400, and HTML data not empty.

    Data format::

        {
            // metadata of URL
            "[metadata]": {
                // original URL - <scheme>://<netloc>/<path>;<params>?<query>#
                <fragment>
                "url": ...,
                // proxy type - null / tor / i2p / zeronet / freenet
                "proxy": ...,
                // hostname / netloc, c.f. ``urllib.parse.urlparse``
                "host": ...,
                // base folder, relative path (to data root path ``PATH_DATA``) in
                <container> - <proxy>/<scheme>/<host>
                "base": ...,
                // sha256 of URL as name for saved files (timestamp is in ISO format)
                // JSON log as this one - <base>/<name>_<timestamp>.json
                // HTML from requests - <base>/<name>_<timestamp>_raw.html
                // HTML from selenium - <base>/<name>_<timestamp>.html
                // generic data files - <base>/<name>_<timestamp>.dat
                "name": ...
            },
            // requested timestamp in ISO format as in name of saved file
            "Timestamp": ...,
            // original URL
            "URL": ...,
            // rendered HTML document (if not exists, then ``null``)
            "Document": {
                // path of the file, relative path (to data root path ``PATH_DATA``)
                <in container>

```

(continues on next page)

(continued from previous page)

```

        // - <proxy>/<scheme>/<host>/<name>_<timestamp>.html
        "path": ...,
        // content of the file (**base64** encoded)
        "data": ...,
    },
    // web page screenshot (if not exists, then ``null``)
    "Screenshot": {
        // path of the file, relative path (to data root path ``PATH_DATA``)
↪in container        // - <proxy>/<scheme>/<host>/<name>_<timestamp>.png
        "path": ...,
        // content of the file (**base64** encoded)
        "data": ...,
    }
}

"""
# JSON data from the request
data = flask.request.json # pylint: disable=unused-variable

# do whatever processing needed
...

if __name__ == "__main__":
    sys.exit(app.run()) # type: ignore

```


DATA MODELS DEMO

This is a demo of data models for database storage of the submitted data from the *darc* crawlers.

Assuming the database is using *peewee* as ORM and *MySQL* as backend.

```
# -*- coding: utf-8 -*-

import datetime
import os

import peewee
import playhouse.shortcuts

# database client
DB = playhouse.db_url.connect(os.getenv('DB_URL', 'mysql://127.0.0.1'))

def table_function(model_class: peewee.Model) -> str:
    """Generate table name dynamically.

    The function strips ``Model`` from the class name and
    calls :func:`peewee.make_snake_case` to generate a
    proper table name.

    Args:
        model_class: Data model class.

    Returns:
        Generated table name.

    """
    name: str = model_class.__name__
    if name.endswith('Model'):
        name = name[:-5] # strip ``Model`` suffix
    return peewee.make_snake_case(name)

class BaseMeta:
    """Basic metadata for data models."""

    #: Reference database storage (c.f. :class:`~darc.const.DB`).
    database = DB

    #: Generate table name dynamically (c.f. :func:`~darc.model.table_function`).
    table_function = table_function
```

(continues on next page)

(continued from previous page)

```

class BaseModel(peewee.Model):
    """Base model with standard patterns.

    Notes:
        The model will implicitly have a :class:`~peewee.AutoField`
        attribute named as :attr:`id`.

    """

    #: Basic metadata for data models.
    Meta = BaseMeta

    def to_dict(self, keep_id: bool = False):
        """Convert record to :obj:`dict`.

        Args:
            keep_id: If keep the ID auto field.

        Returns:
            The data converted through :func:`playhouse.shortcuts.model_to_dict`.

        """
        data = playhouse.shortcuts.model_to_dict(self)
        if keep_id:
            return data

        if 'id' in data:
            del data['id']
        return data

class HostnameModel(BaseModel):
    """Data model for a hostname record."""

    #: Hostname (c.f. :attr:`link.host` <darc.link.Link.host>).
    hostname: str = peewee.TextField()
    #: Proxy type (c.f. :attr:`link.proxy` <darc.link.Link.proxy>).
    proxy: str = peewee.CharField(max_length=8)

    #: Timestamp of first ``new_host`` submission.
    discovery: datetime.datetime = peewee.DateTimeField()
    #: Timestamp of last related submission.
    last_seen: datetime.datetime = peewee.DateTimeField()

class RobotsModel(BaseModel):
    """Data model for ``robots.txt`` data."""

    #: Hostname (c.f. :attr:`link.host` <darc.link.Link.host>).
    host: HostnameModel = peewee.ForeignKeyField(HostnameModel, backref='robots')
    #: Timestamp of the submission.
    timestamp: datetime.datetime = peewee.DateTimeField()

    #: Document data as :obj:`bytes`.
    data: bytes = peewee.BlobField()

```

(continues on next page)

(continued from previous page)

```

#: Path to the document.
path: str = peewee.CharField()

class SitemapModel(BaseModel):
    """Data model for ``sitemap.xml`` data."""

    #: Hostname (c.f. :attr:`link.host` <darc.link.Link.host>`).
    host: HostnameModel = peewee.ForeignKeyField(HostnameModel, backref='sitemaps')
    #: Timestamp of the submission.
    timestamp: datetime.datetime = peewee.DateTimeField()

    #: Document data as :obj:`bytes`.
    data: bytes = peewee.BlobField()
    #: Path to the document.
    path: str = peewee.CharField()

class HostsModel(BaseModel):
    """Data model for ``hosts.txt`` data."""

    #: Hostname (c.f. :attr:`link.host` <darc.link.Link.host>`).
    host: HostnameModel = peewee.ForeignKeyField(HostnameModel, backref='hosts')
    #: Timestamp of the submission.
    timestamp: datetime.datetime = peewee.DateTimeField()

    #: Document data as :obj:`bytes`.
    data: bytes = peewee.BlobField()
    #: Path to the document.
    path: str = peewee.CharField()

class URLModel(BaseModel):
    """Data model for a requested URL."""

    #: Timestamp of last related submission.
    last_seen: datetime.datetime = peewee.DateTimeField()
    #: Original URL (c.f. :attr:`link.url` <darc.link.Link.url>`).
    url: str = peewee.TextField()

    #: Hostname (c.f. :attr:`link.host` <darc.link.Link.host>`).
    host: HostnameModel = peewee.ForeignKeyField(HostnameModel, backref='urls')
    #: Proxy type (c.f. :attr:`link.proxy` <darc.link.Link.proxy>`).
    proxy: str = peewee.CharField(max_length=8)

    #: Base path (c.f. :attr:`link.base` <darc.link.Link.base>`).
    base: str = peewee.CharField()
    #: Link hash (c.f. :attr:`link.name` <darc.link.Link.name>`).
    name: str = peewee.FixedCharField(max_length=64)

class RequestsDocumentModel(BaseModel):
    """Data model for documents from ``requests`` submission."""

    #: Original URL (c.f. :attr:`link.url` <darc.link.Link.url>`).
    url: URLModel = peewee.ForeignKeyField(URLModel, backref='requests')

```

(continues on next page)

(continued from previous page)

```
#: Document data as :obj:`bytes`.
data: bytes = peewee.BlobField()
#: Path to the document.
path: str = peewee.CharField()

class SeleniumDocumentModel(BaseModel):
    """Data model for documents from ``selenium`` submission."""

    #: Original URL (c.f. :attr:`link.url` <darc.link.Link.url>`).
    url: URLModel = peewee.ForeignKeyField(URLModel, backref='selenium')

    #: Document data as :obj:`bytes`.
    data: bytes = peewee.BlobField()
    #: Path to the document.
    path: str = peewee.CharField()
```

SUBMISSION DATA SCHEMA

To better describe the submitted data, *darc* provides several JSON schema generated from *pydantic* models.

5.1 New Host Submission

The data submission from *darc.submit.submit_new_host()*.

```
{
  "title": "new_host",
  "description": "Data submission from :func:`darc.submit.submit_new_host`.",
  "type": "object",
  "properties": {
    "$PARTIAL$": {
      "title": "$Partial$",
      "description": "partial flag - true / false",
      "type": "boolean"
    },
    "$RELOAD$": {
      "title": "$Reload$",
      "description": "reload flag - true / false",
      "type": "boolean"
    },
    "[metadata]": {
      "title": "[Metadata]",
      "description": "metadata of URL",
      "allOf": [
        {
          "$ref": "#/definitions/Metadata"
        }
      ]
    },
    "Timestamp": {
      "title": "Timestamp",
      "description": "requested timestamp in ISO format as in name of saved file",
      "type": "string",
      "format": "date-time"
    },
    "URL": {
      "title": "Url",
      "description": "original URL",
      "minLength": 1,
      "maxLength": 65536,
      "format": "uri",

```

(continues on next page)

(continued from previous page)

```

    "type": "string"
  },
  "Robots": {
    "title": "Robots",
    "description": "robots.txt from the host (if not exists, then ``null``)",
    "allOf": [
      {
        "$ref": "#/definitions/RobotsDocument"
      }
    ]
  },
  "Sitemaps": {
    "title": "Sitemaps",
    "description": "sitemaps from the host (if none, then ``null``)",
    "type": "array",
    "items": {
      "$ref": "#/definitions/SitemapDocument"
    }
  },
  "Hosts": {
    "title": "Hosts",
    "description": "hosts.txt from the host (if proxy type is ``i2p``; if not_
↪exists, then ``null``)",
    "allOf": [
      {
        "$ref": "#/definitions/HostsDocument"
      }
    ]
  },
  "required": [
    "$PARTIAL$",
    "$RELOAD$",
    "[metadata]",
    "Timestamp",
    "URL"
  ],
  "definitions": {
    "Proxy": {
      "title": "Proxy",
      "description": "Proxy type.",
      "enum": [
        "null",
        "tor",
        "i2p",
        "zeronet",
        "freenet"
      ],
      "type": "string"
    },
    "Metadata": {
      "title": "metadata",
      "description": "Metadata of URL.",
      "type": "object",
      "properties": {
        "url": {
          "title": "Url",

```

(continues on next page)

(continued from previous page)

```

        "description": "original URL - <scheme>://<netloc>/<path>;<params>?<query>#
↪<fragment>",
        "minLength": 1,
        "maxLength": 65536,
        "format": "uri",
        "type": "string"
    },
    "proxy": {
        "$ref": "#/definitions/Proxy"
    },
    "host": {
        "title": "Host",
        "description": "hostname / netloc, c.f. ``urllib.parse.urlparse``,
        "type": "string"
    },
    "base": {
        "title": "Base",
        "description": "base folder, relative path (to data root path ``PATH_
↪DATA``) in container - <proxy>/<scheme>/<host>",
        "type": "string"
    },
    "name": {
        "title": "Name",
        "description": "sha256 of URL as name for saved files (timestamp is in ISO_
↪format) - JSON log as this one: <base>/<name>_<timestamp>.json; - HTML from_
↪requests: <base>/<name>_<timestamp>_raw.html; - HTML from selenium: <base>/<name>_
↪<timestamp>.html; - generic data files: <base>/<name>_<timestamp>.dat",
        "type": "string"
    }
},
"required": [
    "url",
    "proxy",
    "host",
    "base",
    "name"
],
"RobotsDocument": {
    "title": "RobotsDocument",
    "description": "``robots.txt`` document data.",
    "type": "object",
    "properties": {
        "path": {
            "title": "Path",
            "description": "path of the file, relative path (to data root path ``PATH_
↪DATA``) in container - <proxy>/<scheme>/<host>/robots.txt",
            "type": "string"
        },
        "data": {
            "title": "Data",
            "description": "content of the file (**base64** encoded)",
            "type": "string"
        }
    },
    "required": [
        "path",

```

(continues on next page)

(continued from previous page)

```

    "data"
  ]
},
"SiteMapDocument": {
  "title": "SiteMapDocument",
  "description": "Sitemaps document data.",
  "type": "object",
  "properties": {
    "path": {
      "title": "Path",
      "description": "path of the file, relative path (to data root path ``PATH_
↪DATA``) in container - <proxy>/<scheme>/<host>/sitemap_<name>.xml",
      "type": "string"
    },
    "data": {
      "title": "Data",
      "description": "content of the file (**base64** encoded)",
      "type": "string"
    }
  },
  "required": [
    "path",
    "data"
  ]
},
"HostsDocument": {
  "title": "HostsDocument",
  "description": "``hosts.txt`` document data.",
  "type": "object",
  "properties": {
    "path": {
      "title": "Path",
      "description": "path of the file, relative path (to data root path ``PATH_
↪DATA``) in container - <proxy>/<scheme>/<host>/hosts.txt",
      "type": "string"
    },
    "data": {
      "title": "Data",
      "description": "content of the file (**base64** encoded)",
      "type": "string"
    }
  },
  "required": [
    "path",
    "data"
  ]
}
}

```

5.2 Requests Submission

The data submission from `darc.submit.submit_requests()`.

```
{
  "title": "requests",
  "description": "Data submission from :func:`darc.submit.submit_requests`.",
  "type": "object",
  "properties": {
    "$PARTIAL$": {
      "title": "$Partial$",
      "description": "partial flag - true / false",
      "type": "boolean"
    },
    "[metadata]": {
      "title": "[Metadata]",
      "description": "metadata of URL",
      "allOf": [
        {
          "$ref": "#/definitions/Metadata"
        }
      ]
    },
    "Timestamp": {
      "title": "Timestamp",
      "description": "requested timestamp in ISO format as in name of saved file",
      "type": "string",
      "format": "date-time"
    },
    "URL": {
      "title": "Url",
      "description": "original URL",
      "minLength": 1,
      "maxLength": 65536,
      "format": "uri",
      "type": "string"
    },
    "Method": {
      "title": "Method",
      "description": "request method",
      "type": "string"
    },
    "Status-Code": {
      "title": "Status-Code",
      "description": "response status code",
      "exclusiveMinimum": 0,
      "type": "integer"
    },
    "Reason": {
      "title": "Reason",
      "description": "response reason",
      "type": "string"
    },
    "Cookies": {
      "title": "Cookies",
      "description": "response cookies (if any)",
      "type": "object",

```

(continues on next page)

(continued from previous page)

```

    "additionalProperties": {
      "type": "string"
    }
  },
  "Session": {
    "title": "Session",
    "description": "session cookies (if any)",
    "type": "object",
    "additionalProperties": {
      "type": "string"
    }
  },
  "Request": {
    "title": "Request",
    "description": "request headers (if any)",
    "type": "object",
    "additionalProperties": {
      "type": "string"
    }
  },
  "Response": {
    "title": "Response",
    "description": "response headers (if any)",
    "type": "object",
    "additionalProperties": {
      "type": "string"
    }
  },
  "Content-Type": {
    "title": "Content-Type",
    "description": "content type",
    "pattern": "[a-zA-Z0-9.-]+/[a-zA-Z0-9.-]+",
    "type": "string"
  },
  "Document": {
    "title": "Document",
    "description": "requested file (if not exists, then ``null``)",
    "allOf": [
      {
        "$ref": "#/definitions/RequestsDocument"
      }
    ]
  },
  "History": {
    "title": "History",
    "description": "redirection history (if any)",
    "type": "array",
    "items": {
      "$ref": "#/definitions/HistoryModel"
    }
  }
},
"required": [
  "$PARTIAL$",
  "[metadata]",
  "Timestamp",
  "URL",

```

(continues on next page)

(continued from previous page)

```

    "Method",
    "Status-Code",
    "Reason",
    "Cookies",
    "Session",
    "Request",
    "Response",
    "Content-Type",
    "History"
  ],
  "definitions": {
    "Proxy": {
      "title": "Proxy",
      "description": "Proxy type.",
      "enum": [
        "null",
        "tor",
        "i2p",
        "zeronet",
        "freenet"
      ],
      "type": "string"
    },
    "Metadata": {
      "title": "metadata",
      "description": "Metadata of URL.",
      "type": "object",
      "properties": {
        "url": {
          "title": "Url",
          "description": "original URL - <scheme>://<netloc>/<path>;<params>?<query>#<fragment>",
          "minLength": 1,
          "maxLength": 65536,
          "format": "uri",
          "type": "string"
        },
        "proxy": {
          "$ref": "#/definitions/Proxy"
        },
        "host": {
          "title": "Host",
          "description": "hostname / netloc, c.f. ``urllib.parse.urlparse``",
          "type": "string"
        },
        "base": {
          "title": "Base",
          "description": "base folder, relative path (to data root path ``PATH_DATA``) in container - <proxy>/<scheme>/<host>",
          "type": "string"
        },
        "name": {
          "title": "Name",
          "description": "sha256 of URL as name for saved files (timestamp is in ISO format) - JSON log as this one: <base>/<name>_<timestamp>.json; - HTML from requests: <base>/<name>_<timestamp>_raw.html; - HTML from selenium: <base>/<name>_<timestamp>.html; - generic data files: <base>/<name>_<timestamp>.dat",

```

(continues on next page)

(continued from previous page)

```

        "type": "string"
    },
    },
    "required": [
        "url",
        "proxy",
        "host",
        "base",
        "name"
    ]
},
"RequestsDocument": {
    "title": "RequestsDocument",
    "description": ":mod:`requests` document data.",
    "type": "object",
    "properties": {
        "path": {
            "title": "Path",
            "description": "path of the file, relative path (to data root path ``PATH_
↳ DATA``) in container - <proxy>/<scheme>/<host>/<name>_<timestamp>_raw.html; or if
↳ the document is of generic content type, i.e. not HTML - <proxy>/<scheme>/<host>/
↳ <name>_<timestamp>.dat",
            "type": "string"
        },
        "data": {
            "title": "Data",
            "description": "content of the file (**base64** encoded)",
            "type": "string"
        }
    },
    "required": [
        "path",
        "data"
    ]
},
"HistoryModel": {
    "title": "HistoryModel",
    "description": ":mod:`requests` history data.",
    "type": "object",
    "properties": {
        "URL": {
            "title": "Url",
            "description": "original URL",
            "minLength": 1,
            "maxLength": 65536,
            "format": "uri",
            "type": "string"
        },
        "Method": {
            "title": "Method",
            "description": "request method",
            "type": "string"
        },
        "Status-Code": {
            "title": "Status-Code",
            "description": "response status code",
            "exclusiveMinimum": 0,

```

(continues on next page)

(continued from previous page)

```

        "type": "integer"
    },
    "Reason": {
        "title": "Reason",
        "description": "response reason",
        "type": "string"
    },
    "Cookies": {
        "title": "Cookies",
        "description": "response cookies (if any)",
        "type": "object",
        "additionalProperties": {
            "type": "string"
        }
    },
    "Session": {
        "title": "Session",
        "description": "session cookies (if any)",
        "type": "object",
        "additionalProperties": {
            "type": "string"
        }
    },
    "Request": {
        "title": "Request",
        "description": "request headers (if any)",
        "type": "object",
        "additionalProperties": {
            "type": "string"
        }
    },
    "Response": {
        "title": "Response",
        "description": "response headers (if any)",
        "type": "object",
        "additionalProperties": {
            "type": "string"
        }
    },
    "Document": {
        "title": "Document",
        "description": "content of the file (**base64** encoded)",
        "type": "string"
    }
},
"required": [
    "URL",
    "Method",
    "Status-Code",
    "Reason",
    "Cookies",
    "Session",
    "Request",
    "Response",
    "Document"
]
}

```

(continues on next page)

(continued from previous page)

```
}
}
```

5.3 Selenium Submission

The data submission from `darc.submit.submit_selenium()`.

```
{
  "title": "selenium",
  "description": "Data submission from :func:`darc.submit.submit_requests`.",
  "type": "object",
  "properties": {
    "$PARTIAL$": {
      "title": "$Partial$",
      "description": "partial flag - true / false",
      "type": "boolean"
    },
    "[metadata]": {
      "title": "[Metadata]",
      "description": "metadata of URL",
      "allOf": [
        {
          "$ref": "#/definitions/Metadata"
        }
      ]
    },
    "Timestamp": {
      "title": "Timestamp",
      "description": "requested timestamp in ISO format as in name of saved file",
      "type": "string",
      "format": "date-time"
    },
    "URL": {
      "title": "Url",
      "description": "original URL",
      "minLength": 1,
      "maxLength": 65536,
      "format": "uri",
      "type": "string"
    },
    "Document": {
      "title": "Document",
      "description": "rendered HTML document (if not exists, then ``null``)",
      "allOf": [
        {
          "$ref": "#/definitions/SeleniumDocument"
        }
      ]
    },
    "Screenshot": {
      "title": "Screenshot",
      "description": "web page screenshot (if not exists, then ``null``)",
      "allOf": [
        {
```

(continues on next page)

(continued from previous page)

```

        "$ref": "#/definitions/ScreenshotDocument"
    }
  ]
},
"required": [
  "$PARTIAL$",
  "[metadata]",
  "Timestamp",
  "URL"
],
"definitions": {
  "Proxy": {
    "title": "Proxy",
    "description": "Proxy type.",
    "enum": [
      "null",
      "tor",
      "i2p",
      "zeronet",
      "freenet"
    ],
    "type": "string"
  },
  "Metadata": {
    "title": "metadata",
    "description": "Metadata of URL.",
    "type": "object",
    "properties": {
      "url": {
        "title": "Url",
        "description": "original URL - <scheme>://<netloc>/<path>;<params>?<query>#<fragment>",
        "minLength": 1,
        "maxLength": 65536,
        "format": "uri",
        "type": "string"
      },
      "proxy": {
        "$ref": "#/definitions/Proxy"
      },
      "host": {
        "title": "Host",
        "description": "hostname / netloc, c.f. ``urllib.parse.urlparse``,",
        "type": "string"
      },
      "base": {
        "title": "Base",
        "description": "base folder, relative path (to data root path ``PATH_DATA``) in container - <proxy>/<scheme>/<host>",
        "type": "string"
      },
      "name": {
        "title": "Name",
        "description": "sha256 of URL as name for saved files (timestamp is in ISO format) - JSON log as this one: <base>/<name>_<timestamp>.json; - HTML from requests: <base>/<name>_<timestamp>_raw.html; - HTML from selenium: <base>/<name>_<timestamp>.html; - generic data files: <base>/<name>_<timestamp>.dat"
      }
    }
  }
}

```

(continued from previous page)

```

        "type": "string"
    },
    },
    "required": [
        "url",
        "proxy",
        "host",
        "base",
        "name"
    ]
},
"SeleniumDocument": {
    "title": "SeleniumDocument",
    "description": ":mod:`selenium` document data.",
    "type": "object",
    "properties": {
        "path": {
            "title": "Path",
            "description": "path of the file, relative path (to data root path ``PATH_
↪DATA``) in container - <proxy>/<scheme>/<host>/<name>_<timestamp>.html",
            "type": "string"
        },
        "data": {
            "title": "Data",
            "description": "content of the file (**base64** encoded)",
            "type": "string"
        }
    },
    "required": [
        "path",
        "data"
    ]
},
"ScreenshotDocument": {
    "title": "ScreenshotDocument",
    "description": "Screenshot document data.",
    "type": "object",
    "properties": {
        "path": {
            "title": "Path",
            "description": "path of the file, relative path (to data root path ``PATH_
↪DATA``) in container - <proxy>/<scheme>/<host>/<name>_<timestamp>.png",
            "type": "string"
        },
        "data": {
            "title": "Data",
            "description": "content of the file (**base64** encoded)",
            "type": "string"
        }
    },
    "required": [
        "path",
        "data"
    ]
}
}
}

```

5.4 Model Definitions

```
# -*- coding: utf-8 -*-
"""JSON schema generator."""
# pylint: disable=no-member

import enum

import pydantic.schema

import darc.typing as typing

__all__ = ['NewHostModel', 'RequestsModel', 'SeleniumModel']

#####
# Miscellaneous auxiliaries
#####

class Proxy(str, enum.Enum):
    """Proxy type."""

    null = 'null'
    tor = 'tor'
    i2p = 'i2p'
    zeronet = 'zeronet'
    freenet = 'freenet'

class Metadata(pydantic.BaseModel):
    """Metadata of URL."""

    url: pydantic.AnyUrl = pydantic.Field(
        description='original URL - <scheme>://<netloc>/<path>;<params>?<query>#<fragment>')
    proxy: Proxy = pydantic.Field(
        description='proxy type - null / tor / i2p / zeronet / freenet')
    host: str = pydantic.Field(
        description='hostname / netloc, c.f. ``urllib.parse.urlparse``')
    base: str = pydantic.Field(
        description=('base folder, relative path (to data root path ``PATH_DATA``) in container '
                     '- <proxy>/<scheme>/<host>'))
    name: str = pydantic.Field(
        description=('sha256 of URL as name for saved files (timestamp is in ISO format) '
                     '- JSON log as this one: <base>/<name>_<timestamp>.json; '
                     '- HTML from requests: <base>/<name>_<timestamp>_raw.html; '
                     '- HTML from selenium: <base>/<name>_<timestamp>.html; '
                     '- generic data files: <base>/<name>_<timestamp>.dat'))

    class Config:
        title = 'metadata'

class RobotsDocument(pydantic.BaseModel):
    """``robots.txt`` document data."""
```

(continues on next page)

(continued from previous page)

```

    path: str = pydantic.Field(
        description=('path of the file, relative path (to data root path ``PATH_
↳DATA``) in container '
                    '- <proxy>/<scheme>/<host>/robots.txt'))
    data: str = pydantic.Field(
        description='content of the file (**base64** encoded)')

class SitemapDocument(pydantic.BaseModel):
    """Sitemaps document data."""

    path: str = pydantic.Field(
        description=('path of the file, relative path (to data root path ``PATH_
↳DATA``) in container '
                    '- <proxy>/<scheme>/<host>/sitemap_<name>.xml'))
    data: str = pydantic.Field(
        description='content of the file (**base64** encoded)')

class HostsDocument(pydantic.BaseModel):
    """`hosts.txt` document data."""

    path: str = pydantic.Field(
        description=('path of the file, relative path (to data root path ``PATH_
↳DATA``) in container '
                    '- <proxy>/<scheme>/<host>/hosts.txt'))
    data: str = pydantic.Field(
        description='content of the file (**base64** encoded)')

class RequestsDocument(pydantic.BaseModel):
    """`mod:requests` document data."""

    path: str = pydantic.Field(
        description=('path of the file, relative path (to data root path ``PATH_
↳DATA``) in container '
                    '- <proxy>/<scheme>/<host>/<name>_<timestamp>_raw.html; '
                    'or if the document is of generic content type, i.e. not HTML '
                    '- <proxy>/<scheme>/<host>/<name>_<timestamp>.dat'))
    data: str = pydantic.Field(
        description='content of the file (**base64** encoded)')

class HistoryModel(pydantic.BaseModel):
    """`mod:requests` history data."""

    URL: pydantic.AnyUrl = pydantic.Field(
        description='original URL')

    Method: str = pydantic.Field(
        description='request method')
    status_code: pydantic.PositiveInt = pydantic.Field(
        alias='Status-Code',
        description='response status code')
    Reason: str = pydantic.Field(
        description='response reason')

```

(continues on next page)

(continued from previous page)

```

Cookies: typing.Cookies = pydantic.Field(
    description='response cookies (if any)')
Session: typing.Cookies = pydantic.Field(
    description='session cookies (if any)')

Request: typing.Headers = pydantic.Field(
    description='request headers (if any)')
Response: typing.Headers = pydantic.Field(
    description='response headers (if any)')

Document: str = pydantic.Field(
    description='content of the file (**base64** encoded)')

class SeleniumDocument(pydantic.BaseModel):
    """:mod:`selenium` document data."""

    path: str = pydantic.Field(
        description=('path of the file, relative path (to data root path ``PATH_`
→DATA``) in container '
                    '- <proxy>/<scheme>/<host>/<name>_<timestamp>.html'))
    data: str = pydantic.Field(
        description='content of the file (**base64** encoded)')

class ScreenshotDocument(pydantic.BaseModel):
    """Screenshot document data."""

    path: str = pydantic.Field(
        description=('path of the file, relative path (to data root path ``PATH_`
→DATA``) in container '
                    '- <proxy>/<scheme>/<host>/<name>_<timestamp>.png'))
    data: str = pydantic.Field(
        description='content of the file (**base64** encoded)')

#####
# JSON schema definitions
#####

class NewHostModel(pydantic.BaseModel):
    """Data submission from :func:`darc.submit.submit_new_host`."""

    partial: bool = pydantic.Field(
        alias='$PARTIAL$',
        description='partial flag - true / false')
    reload: bool = pydantic.Field(
        alias='$RELOAD$',
        description='reload flag - true / false')
    metadata: Metadata = pydantic.Field(
        alias='[metadata]',
        description='metadata of URL')

    Timestamp: typing.Datetime = pydantic.Field(
        description='requested timestamp in ISO format as in name of saved file')

```

(continues on next page)

(continued from previous page)

```

URL: pydantic.AnyUrl = pydantic.Field(
    description='original URL')

Robots: typing.Optional[RobotsDocument] = pydantic.Field(
    description='robots.txt from the host (if not exists, then ``null``)')
Sitemaps: typing.Optional[typing.List[SitemapDocument]] = pydantic.Field(
    description='sitemaps from the host (if none, then ``null``)')
Hosts: typing.Optional[HostsDocument] = pydantic.Field(
    description='hosts.txt from the host (if proxy type is ``i2p``; if not exists,
→ then ``null``)')

class Config:
    title = 'new_host'

class RequestsModel(pydantic.BaseModel):
    """Data submission from :func:`darc.submit.submit_requests`."""

    partial: bool = pydantic.Field(
        alias='$PARTIAL$',
        description='partial flag - true / false')
    metadata: Metadata = pydantic.Field(
        alias='[metadata]',
        description='metadata of URL')

    Timestamp: typing.Datetime = pydantic.Field(
        description='requested timestamp in ISO format as in name of saved file')
    URL: pydantic.AnyUrl = pydantic.Field(
        description='original URL')

    Method: str = pydantic.Field(
        description='request method')
    status_code: pydantic.PositiveInt = pydantic.Field(
        alias='Status-Code',
        description='response status code')
    Reason: str = pydantic.Field(
        description='response reason')

    Cookies: typing.Cookies = pydantic.Field(
        description='response cookies (if any)')
    Session: typing.Cookies = pydantic.Field(
        description='session cookies (if any)')

    Request: typing.Headers = pydantic.Field(
        description='request headers (if any)')
    Response: typing.Headers = pydantic.Field(
        description='response headers (if any)')
    content_type: str = pydantic.Field(
        alias='Content-Type',
        regex='[a-zA-Z0-9.-]+/[a-zA-Z0-9.-]+' ,
        description='content type')

    Document: typing.Optional[RequestsDocument] = pydantic.Field(
        description='requested file (if not exists, then ``null``)')
    History: typing.List[HistoryModel] = pydantic.Field(
        description='redirection history (if any)')

```

(continues on next page)

(continued from previous page)

```

class Config:
    title = 'requests'

class SeleniumModel(pydantic.BaseModel):
    """Data submission from :func:`darc.submit.submit_requests`."""

    partial: bool = pydantic.Field(
        alias='$PARTIAL$',
        description='partial flag - true / false')
    metadata: Metadata = pydantic.Field(
        alias='[metadata]',
        description='metadata of URL')

    Timestamp: typing.Datetime = pydantic.Field(
        description='requested timestamp in ISO format as in name of saved file')
    URL: pydantic.AnyUrl = pydantic.Field(
        description='original URL')

    Document: typing.Optional[SeleniumDocument] = pydantic.Field(
        description='rendered HTML document (if not exists, then `null`)' )
    Screenshot: typing.Optional[ScreenshotDocument] = pydantic.Field(
        description='web page screenshot (if not exists, then `null`)' )

class Config:
    title = 'selenium'

if __name__ == "__main__":
    import json
    import os

    os.makedirs('schema', exist_ok=True)

    with open('schema/new_host.schema.json', 'w') as file:
        print(NewHostModel.schema_json(indent=2), file=file)
    with open('schema/requests.schema.json', 'w') as file:
        print(RequestsModel.schema_json(indent=2), file=file)
    with open('schema/selenium.schema.json', 'w') as file:
        print(SeleniumModel.schema_json(indent=2), file=file)

    schema = pydantic.schema.schema([NewHostModel, RequestsModel, SeleniumModel],
                                    title='DARC Data Submission JSON Schema')
    with open('schema/darc.schema.json', 'w') as file:
        json.dump(schema, file, indent=2)

```


AUXILIARY SCRIPTS

Since the *darc* project can be deployed through *Docker Integration*, we provided some auxiliary scripts to help with the deployment.

6.1 Health Check

File location

- Entry point: `extra/healthcheck.py`
- System V service: `extra/healthcheck.service`

```
usage: healthcheck [-h] [-f FILE] [-i INTERVAL] ...

health check running container

positional arguments:
  services              name of services

optional arguments:
  -h, --help            show this help message and exit
  -f FILE, --file FILE  path to compose file
  -i INTERVAL, --interval INTERVAL
                        interval (in seconds) of health check
```

This script will watch the running status of containers managed by Docker Compose. If the containers are stopped or of *unhealthy* status, it will bring the containers back alive.

Also, as the internal program may halt unexpectedly whilst the container remains *healthy*, the script will watch if the program is still active through its output messages. If inactive, the script will restart the containers.

6.2 Upload API Submission Files

File location

- Entry point: `extra/upload.py`
- Helper script: `extra/upload.sh`
- System V service: `extra/upload.service`

```
usage: upload [-h] [-f FILE] [-p PATH] [-i INTERVAL] -H HOST [-U USER]

upload API submission files

optional arguments:
  -h, --help            show this help message and exit
  -f FILE, --file FILE  path to compose file
  -p PATH, --path PATH  path to data storage
  -i INTERVAL, --interval INTERVAL
                        interval (in seconds) to upload
  -H HOST, --host HOST  upstream hostname
  -U USER, --user USER  upstream user credential
```

This script will automatically upload API submission files, c.f. `darc.submit`, using `curl(1)`. The `--user` option is supplied for the same option of `curl(1)`.

When uploading, the script will *pause* the running containers and it will *unpause* them upon completion.

6.3 Remove Repeated Lines

File location `extra/uniq.py`

This script works the same as `uniq(1)`, except it filters one input line at a time without putting pressure onto memory utilisation.

`darc` is designed as a swiss army 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.

There are two types of *workers*:

- `crawler` – runs the `darc.crawl.crawler()` to provide a fresh view of a link and test its connectability
- `loader` – run the `darc.crawl.loader()` to provide an in-depth view of a link and provide more visual information

The general process can be described as following for *workers* of `crawler` type:

1. `process_crawler()`: obtain URLs from the `requests` link database (c.f. `load_requests()`), and feed such URLs to `crawler()`.

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.

2. `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`), `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()`).

The general process can be described as following for *workers* of loader type:

1. `process_loader()`: in the meanwhile, `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.

2. `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 `WebDriver` object.

If successful, the rendered source HTML document will be saved, 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()`).

INSTALLATION

Note: `darc` supports Python all versions above and includes **3.6**. Currently, it only supports and is tested on Linux (*Ubuntu 18.04*) and macOS (*Catalina*).

When installing in Python versions below **3.8**, `darc` will use `walrus` to compile itself for backport compatibility.

```
pip install darc
```

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

Important: Starting from version **0.3.0**, we introduced `Redis` for the task queue database backend.

Since version **0.6.0**, we introduced relationship database storage (e.g. `MySQL`, `SQLite`, `PostgreSQL`, etc.) for the task queue database backend, besides the `Redis` database, since it can be too much memory-costly when the task queue becomes vary large.

Please make sure you have one of the backend database installed, configured, and running when using the `darc` project.

However, the `darc` project is shipped with Docker and Compose support. Please see *Docker Integration* for more information.

Or, you may refer to and/or install from the `Docker Hub` repository:

```
docker pull jsnbzh/darc[:TAGNAME]
```


USAGE

The *darc* project provides a simple CLI:

```
usage: darc [-h] [-v] -t {crawler,loader} [-f FILE] ...

the darkweb crawling swiss army knife

positional arguments:
  link                  links to crawl

optional arguments:
  -h, --help            show this help message and exit
  -v, --version          show program's version number and exit
  -t {crawler,loader}, --type {crawler,loader}
                        type of worker process
  -f FILE, --file FILE  read links from file
```

It can also be called through module entrypoint:

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

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

CONFIGURATION

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

9.1 General Configurations

DARC_REBOOT

Type `bool(int)`

Default 0

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.

This can be useful especially when the capacity is limited and you wish to save some space before continuing next round. See *Docker integration* for more information.

DARC_DEBUG

Type `bool(int)`

Default 0

If run the program in debugging mode.

DARC_VERBOSE

Type `bool(int)`

Default 0

If run the program in verbose mode. If `DARC_DEBUG` is `True`, then the verbose mode will be always enabled.

DARC_FORCE

Type `bool(int)`

Default 0

If ignore `robots.txt` rules when crawling (c.f. `crawler()`).

DARC_CHECK

Type `bool(int)`

Default 0

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`.

DARC_CHECK_CONTENT_TYPE

Type `bool(int)`

Default `0`

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

DARC_CPU

Type `int`

Default `None`

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

DARC_MULTIPROCESSING

Type `bool(int)`

Default `1`

If enable *multiprocessing* support.

DARC_MULTITHREADING

Type `bool(int)`

Default `0`

If enable *multithreading* support.

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

DARC_USER

Type `str`

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

Non-root user for proxies.

9.2 Data Storage

See also:

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

See `darc.db` for more information about database integration.

PATH_DATA

Type `str(path)`

Default `data`

Path to data storage.

REDIS_URL

Type `str(url)`

Default `redis://127.0.0.1`

URL to the Redis database.

DB_URL

Type `str` (url)

URL to the RDS storage.

Important: The task queues will be saved to `darc` database; the data submission will be saved to `darcweb` database.

Thus, when providing this environment variable, please do **NOT** specify the database name.

DARC_BULK_SIZE

Type `int`

Default 100

Bulk size for updating databases.

See also:

- `darc.db.save_requests()`
- `darc.db.save_selenium()`

LOCK_TIMEOUT

Type `float`

Default 10

Lock blocking timeout.

Note: If is an infinit `inf`, no timeout will be applied.

See also:

Get a lock from `darc.db.get_lock()`.

DARC_MAX_POOL

Type `int`

Default 1_000

Maximum number of links loaded from the database.

Note: If is an infinit `inf`, no limit will be applied.

See also:

- `darc.db.load_requests()`
- `darc.db.load_selenium()`

REDIS_LOCK

Type `bool` (int)

Default 0

If use Redis (Lua) lock to ensure process/thread-safely operations.

See also:

Toggles the behaviour of `darc.db.get_lock()`.

RETRY_INTERVAL

Type `int`

Default 10

Retry interval between each Redis command failure.

Note: If is an infinit `inf`, no interval will be applied.

See also:

Toggles the behaviour of `darc.db.redis_command()`.

9.3 Web Crawlers

DARC_WAIT

Type `float`

Default 60

Time interval between each round when the `requests` and/or `selenium` database are empty.

DARC_SAVE

Type `bool(int)`

Default 0

If save processed link back to database.

Note: If `DARC_SAVE` is `True`, then `DARC_SAVE_REQUESTS` and `DARC_SAVE_SELENIUM` will be forced to be `True`.

See also:

See `darc.db` for more information about link database.

DARC_SAVE_REQUESTS

Type `bool(int)`

Default 0

If save `crawler()` crawled link back to `requests` database.

See also:

See `darc.db` for more information about link database.

DARC_SAVE_SELENIUM

Type `bool(int)`

Default 0

If save `loader()` crawled link back to selenium database.

See also:

See `darc.db` for more information about link database.

TIME_CACHE**Type** float**Default** 60

Time delta for caches in seconds.

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

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

SE_WAIT**Type** float**Default** 60

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.

9.4 White / Black Lists

LINK_WHITE_LIST**Type** List[str] (JSON)**Default** []

White list of hostnames should be crawled.

Note: Regular expressions are supported.

LINK_BLACK_LIST**Type** List[str] (JSON)**Default** []

Black list of hostnames should be crawled.

Note: Regular expressions are supported.

LINK_FALLBACK**Type** bool (int)

Default 0

Fallback value for `match_host()`.

MIME_WHITE_LIST

Type `List[str]` (JSON)

Default []

White list of content types should be crawled.

Note: Regular expressions are supported.

MIME_BLACK_LIST

Type `List[str]` (JSON)

Default []

Black list of content types should be crawled.

Note: Regular expressions are supported.

MIME_FALLBACK

Type `bool(int)`

Default 0

Fallback value for `match_mime()`.

PROXY_WHITE_LIST

Type `List[str]` (JSON)

Default []

White list of proxy types should be crawled.

Note: The proxy types are **case insensitive**.

PROXY_BLACK_LIST

Type `List[str]` (JSON)

Default []

Black list of proxy types should be crawled.

Note: The proxy types are **case insensitive**.

PROXY_FALLBACK

Type `bool(int)`

Default 0

Fallback value for `match_proxy()`.

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.

9.5 Data Submission

`SAVE_DB`

Type `bool`

Default `True`

Save submitted data to database.

`API_RETRY`

Type `int`

Default `3`

Retry times for API submission when failure.

`API_NEW_HOST`

Type `str`

Default `None`

API URL for `submit_new_host()`.

`API_REQUESTS`

Type `str`

Default `None`

API URL for `submit_requests()`.

`API_SELENIUM`

Type `str`

Default `None`

API URL for `submit_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_DATA`.

9.6 Tor Proxy Configuration

DARC_TOR

Type `bool(int)`

Default `1`

If manage the Tor proxy through *darc*.

TOR_PORT

Type `int`

Default `9050`

Port for Tor proxy connection.

TOR_CTRL

Type `int`

Default `9051`

Port for Tor controller connection.

TOR_PASS

Type `str`

Default `None`

Tor controller authentication token.

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

TOR_RETRY

Type `int`

Default `3`

Retry times for Tor bootstrap when failure.

TOR_WAIT

Type `float`

Default `90`

Time after which the attempt to start Tor is aborted.

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

TOR_CFG

Type `Dict[str, Any]` (JSON)

Default `{}`

Tor bootstrap configuration for `stem.process.launch_tor_with_config()`.

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

9.7 I2P Proxy Configuration

DARC_I2P

Type `bool (int)`

Default `1`

If manage the I2P proxy through *darc*.

I2P_PORT

Type `int`

Default `4444`

Port for I2P proxy connection.

I2P_RETRY

Type `int`

Default `3`

Retry times for I2P bootstrap when failure.

I2P_WAIT

Type `float`

Default `90`

Time after which the attempt to start I2P is aborted.

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

I2P_ARGS

Type `str (Shell)`

Default `' '`

I2P bootstrap arguments for `i2prouter start`.

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

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

9.8 ZeroNet Proxy Configuration

DARC_ZERONET

Type `bool (int)`

Default `1`

If manage the ZeroNet proxy through *darc*.

ZERONET_PORT

Type `int`

Default `4444`

Port for ZeroNet proxy connection.

ZERONET_RETRY

Type `int`

Default `3`

Retry times for ZeroNet bootstrap when failure.

ZERONET_WAIT

Type `float`

Default `90`

Time after which the attempt to start ZeroNet is aborted.

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

ZERONET_PATH

Type `str (path)`

Default `/usr/local/src/zernet`

Path to the ZeroNet project.

ZERONET_ARGS

Type `str (Shell)`

Default `' '`

ZeroNet bootstrap arguments for `ZeroNet.sh main`.

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

9.9 Freenet Proxy Configuration

DARC_FREENET

Type `bool (int)`

Default 1

If manage the Freenet proxy through *darc*.

FREENET_PORT

Type `int`

Default 8888

Port for Freenet proxy connection.

FREENET_RETRY

Type `int`

Default 3

Retry times for Freenet bootstrap when failure.

FREENET_WAIT

Type `float`

Default 90

Time after which the attempt to start Freenet is aborted.

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

FREENET_PATH

Type `str (path)`

Default `/usr/local/src/freenet`

Path to the Freenet project.

FREENET_ARGS

Type `str (Shell)`

Default `' '`

Freenet bootstrap arguments for `run.sh start`.

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

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

INDICES AND TABLES

- `genindex`
- `modindex`
- `search`

PYTHON MODULE INDEX

d

- darc, 1
- darc.crawl, 3
- darc.db, 13
- darc.error, 60
- darc.link, 5
- darc.model, 62
- darc.model.abc, 69
- darc.model.tasks, 62
- darc.model.tasks.hostname, 62
- darc.model.tasks.requests, 62
- darc.model.tasks.selenium, 63
- darc.model.utils, 70
- darc.model.web, 63
- darc.model.web.hostname, 63
- darc.model.web.hosts, 66
- darc.model.web.requests, 67
- darc.model.web.robots, 65
- darc.model.web.selenium, 69
- darc.model.web.sitemap, 66
- darc.model.web.url, 64
- darc.parse, 8
- darc.process, 1
- darc.proxy, 32
- darc.proxy.bitcoin, 32
- darc.proxy.data, 33
- darc.proxy.ed2k, 33
- darc.proxy.freenet, 33
- darc.proxy.i2p, 35
- darc.proxy.irc, 38
- darc.proxy.magnet, 39
- darc.proxy.mail, 39
- darc.proxy.null, 40
- darc.proxy.script, 42
- darc.proxy.tel, 43
- darc.proxy.tor, 43
- darc.proxy.zeronet, 45
- darc.requests, 29
- darc.save, 10
- darc.selenium, 30
- darc.sites, 48
- darc.sites.bitcoin, 50
- darc.sites.data, 50
- darc.sites.default, 48
- darc.sites.ed2k, 51
- darc.sites.irc, 51
- darc.sites.magnet, 52
- darc.sites.mail, 52
- darc.sites.script, 53
- darc.sites.tel, 53
- darc.submit, 21

Symbols

_WORKER_POOL (in module *darc.process*), 3
 __hash__() (*darc.link.Link* method), 5
 _check() (in module *darc.parse*), 8
 _check_ng() (in module *darc.parse*), 8
 _db_operation() (in module *darc.db*), 13
 _drop_hostname_db() (in module *darc.db*), 13
 _drop_hostname_redis() (in module *darc.db*), 14
 _drop_requests_db() (in module *darc.db*), 14
 _drop_requests_redis() (in module *darc.db*), 14
 _drop_selenium_db() (in module *darc.db*), 14
 _drop_selenium_redis() (in module *darc.db*), 14
 _freenet_bootstrap() (in module *darc.proxy.freenet*), 34
 _gen_arg_msg() (in module *darc.db*), 14
 _get_spec() (in module *darc.sites*), 55
 _have_hostname_db() (in module *darc.db*), 14
 _have_hostname_redis() (in module *darc.db*), 14
 _i2p_bootstrap() (in module *darc.proxy.i2p*), 35
 _load_requests_db() (in module *darc.db*), 15
 _load_requests_redis() (in module *darc.db*), 15
 _load_selenium_db() (in module *darc.db*), 15
 _load_selenium_redis() (in module *darc.db*), 15
 _process() (in module *darc.process*), 1
 _redis_command() (in module *darc.db*), 15
 _redis_get_lock() (in module *darc.db*), 16
 _save_requests_db() (in module *darc.db*), 16
 _save_requests_redis() (in module *darc.db*), 16
 _save_selenium_db() (in module *darc.db*), 17
 _save_selenium_redis() (in module *darc.db*), 17
 _signal_handler() (in module *darc.process*), 1
 _tor_bootstrap() (in module *darc.proxy.tor*), 43
 _zeronet_bootstrap() (in module *darc.proxy.zeronet*), 46

A

alive (*darc.model.web.hostname.HostnameModel* attribute), 64
 alive (*darc.model.web.url.URLModel* attribute), 65
 API_NEW_HOST, 29
 API_REQUESTS, 29
 API_RETRY, 29

API_SELENIUM, 29
 APIRequestFailed, 60

B

base (*darc.link.Link* attribute), 5
 BaseMeta (class in *darc.model.abc*), 69
 BaseMetaWeb (class in *darc.model.abc*), 69
 BaseModel (class in *darc.model.abc*), 70
 BaseModelWeb (class in *darc.model.abc*), 70

C

check_robots() (in module *darc.parse*), 9
 choices (*darc.model.utils.IntEnumField* attribute), 71
 cookies (*darc.model.web.requests.RequestsHistoryModel* attribute), 67
 cookies (*darc.model.web.requests.RequestsModel* attribute), 68
 crawler() (in module *darc.crawl*), 3
 crawler() (in module *darc.sites.bitcoin*), 50
 crawler() (in module *darc.sites.data*), 50
 crawler() (in module *darc.sites.default*), 48
 crawler() (in module *darc.sites.ed2k*), 51
 crawler() (in module *darc.sites.irc*), 51
 crawler() (in module *darc.sites.magnet*), 52
 crawler() (in module *darc.sites.mail*), 52
 crawler() (in module *darc.sites.script*), 53
 crawler() (in module *darc.sites.tel*), 53
 crawler_hook() (in module *darc.sites*), 54

D

darc
 module, 1
 darc() (in module *darc*), 73
 darc.const.CHECK (built-in variable), 56
 darc.const.CHECK_NG (built-in variable), 56
 darc.const.CWD (built-in variable), 56
 darc.const.DARC_CPU (built-in variable), 56
 darc.const.DARC_USER (built-in variable), 57
 darc.const.DARC_WAIT (built-in variable), 58
 darc.const.DB (built-in variable), 57
 darc.const.DEBUG (built-in variable), 55
 darc.const.FLAG_DB (built-in variable), 57

`darc.const.FLAG_MP` (*built-in variable*), 56
`darc.const.FLAG_TH` (*built-in variable*), 56
`darc.const.FORCE` (*built-in variable*), 56
`darc.const.LINK_BLACK_LIST` (*built-in variable*), 59
`darc.const.LINK_FALLBACK` (*built-in variable*), 59
`darc.const.LINK_WHITE_LIST` (*built-in variable*), 59
`darc.const.MIME_BLACK_LIST` (*built-in variable*), 59
`darc.const.MIME_FALLBACK` (*built-in variable*), 59
`darc.const.MIME_WHITE_LIST` (*built-in variable*), 59
`darc.const.PATH_DB` (*built-in variable*), 57
`darc.const.PATH_ID` (*built-in variable*), 58
`darc.const.PATH_LN` (*built-in variable*), 57
`darc.const.PATH_MISC` (*built-in variable*), 57
`darc.const.PROXY_BLACK_LIST` (*built-in variable*), 60
`darc.const.PROXY_FALLBACK` (*built-in variable*), 60
`darc.const.PROXY_WHITE_LIST` (*built-in variable*), 59
`darc.const.REBOOT` (*built-in variable*), 55
`darc.const.REDIS` (*built-in variable*), 57
`darc.const.ROOT` (*built-in variable*), 56
`darc.const.SE_EMPTY` (*built-in variable*), 58
`darc.const.SE_WAIT` (*built-in variable*), 58
`darc.const.TIME_CACHE` (*built-in variable*), 58
`darc.const.VERBOSE` (*built-in variable*), 56
`darc.crawl`
 module, 3
`darc.db`
 module, 13
`darc.db.BULK_SIZE` (*in module darc.db*), 20
`darc.db.LOCK_TIMEOUT` (*in module darc.db*), 20
`darc.db.MAX_POOL` (*in module darc.db*), 20
`darc.db.REDIS_LOCK` (*in module darc.db*), 21
`darc.db.RETRY_INTERVAL` (*in module darc.db*), 21
`darc.error`
 module, 60
`darc.link`
 module, 5
`darc.model`
 module, 62
`darc.model.abc`
 module, 69
`darc.model.tasks`
 module, 62
`darc.model.tasks.hostname`
 module, 62
`darc.model.tasks.requests`
 module, 62
`darc.model.tasks.selenium`
 module, 63
`darc.model.utils`
 module, 70
`darc.model.web`
 module, 63
`darc.model.web.hostname`
 module, 63
`darc.model.web.hosts`
 module, 66
`darc.model.web.requests`
 module, 67
`darc.model.web.robots`
 module, 65
`darc.model.web.selenium`
 module, 69
`darc.model.web.sitemap`
 module, 66
`darc.model.web.url`
 module, 64
`darc.parse`
 module, 8
`darc.process`
 module, 1
`darc.proxy`
 module, 32
`darc.proxy.bitcoin`
 module, 32
`darc.proxy.bitcoin.LOCK` (*in module darc.proxy.bitcoin*), 32
`darc.proxy.bitcoin.PATH` (*in module darc.proxy.bitcoin*), 32
`darc.proxy.data`
 module, 33
`darc.proxy.data.PATH` (*in module darc.proxy.data*), 33
`darc.proxy.ed2k`
 module, 33
`darc.proxy.ed2k.LOCK` (*in module darc.proxy.ed2k*), 33
`darc.proxy.ed2k.PATH` (*in module darc.proxy.ed2k*), 33
`darc.proxy.freenet`
 module, 33
`darc.proxy.freenet._FREENET_ARGS` (*in module darc.proxy.freenet*), 35
`darc.proxy.freenet._FREENET_BS_FLAG` (*in module darc.proxy.freenet*), 35
`darc.proxy.freenet._FREENET_PROC` (*in module darc.proxy.freenet*), 35
`darc.proxy.freenet._MNG_FREENET` (*in module darc.proxy.freenet*), 35

darc.proxy.freenet.BS_WAIT (in module [darc.proxy.freenet](#)), 34
 darc.proxy.freenet.FREENET_ARGS (in module [darc.proxy.freenet](#)), 35
 darc.proxy.freenet.FREENET_PATH (in module [darc.proxy.freenet](#)), 35
 darc.proxy.freenet.FREENET_PORT (in module [darc.proxy.freenet](#)), 34
 darc.proxy.freenet.FREENET_RETRY (in module [darc.proxy.freenet](#)), 34
 darc.proxy.i2p module, 35
 darc.proxy.i2p._I2P_ARGS (in module [darc.proxy.i2p](#)), 38
 darc.proxy.i2p._I2P_BS_FLAG (in module [darc.proxy.i2p](#)), 38
 darc.proxy.i2p._I2P_PROC (in module [darc.proxy.i2p](#)), 38
 darc.proxy.i2p._MNG_I2P (in module [darc.proxy.i2p](#)), 38
 darc.proxy.i2p.BS_WAIT (in module [darc.proxy.i2p](#)), 37
 darc.proxy.i2p.I2P_ARGS (in module [darc.proxy.i2p](#)), 38
 darc.proxy.i2p.I2P_PORT (in module [darc.proxy.i2p](#)), 37
 darc.proxy.i2p.I2P_REQUESTS_PROXY (in module [darc.proxy.i2p](#)), 37
 darc.proxy.i2p.I2P_RETRY (in module [darc.proxy.i2p](#)), 37
 darc.proxy.i2p.I2P_SELENIUM_PROXY (in module [darc.proxy.i2p](#)), 37
 darc.proxy.irc module, 38
 darc.proxy.irc.LOCK (in module [darc.proxy.irc](#)), 38
 darc.proxy.irc.PATH (in module [darc.proxy.irc](#)), 38
 darc.proxy.LINK_MAP (in module [darc.proxy](#)), 47
 darc.proxy.magnet module, 39
 darc.proxy.magnet.LOCK (in module [darc.proxy.magnet](#)), 39
 darc.proxy.magnet.PATH (in module [darc.proxy.magnet](#)), 39
 darc.proxy.mail module, 39
 darc.proxy.mail.LOCK (in module [darc.proxy.mail](#)), 39
 darc.proxy.mail.PATH (in module [darc.proxy.mail](#)), 39
 darc.proxy.null module, 40
 darc.proxy.null.LOCK (in module [darc.proxy.null](#)), 42
 darc.proxy.null.PATH (in module [darc.proxy.null](#)), 42
 darc.proxy.script module, 42
 darc.proxy.script.LOCK (in module [darc.proxy.script](#)), 43
 darc.proxy.script.PATH (in module [darc.proxy.script](#)), 42
 darc.proxy.tel module, 43
 darc.proxy.tel.LOCK (in module [darc.proxy.tel](#)), 43
 darc.proxy.tel.PATH (in module [darc.proxy.tel](#)), 43
 darc.proxy.tor module, 43
 darc.proxy.tor._MNG_TOR (in module [darc.proxy.tor](#)), 45
 darc.proxy.tor._TOR_BS_FLAG (in module [darc.proxy.tor](#)), 45
 darc.proxy.tor._TOR_CONFIG (in module [darc.proxy.tor](#)), 45
 darc.proxy.tor._TOR_CTRL (in module [darc.proxy.tor](#)), 45
 darc.proxy.tor._TOR_PROC (in module [darc.proxy.tor](#)), 45
 darc.proxy.tor.BS_WAIT (in module [darc.proxy.tor](#)), 45
 darc.proxy.tor.TOR_CFG (in module [darc.proxy.tor](#)), 45
 darc.proxy.tor.TOR_CTRL (in module [darc.proxy.tor](#)), 44
 darc.proxy.tor.TOR_PASS (in module [darc.proxy.tor](#)), 44
 darc.proxy.tor.TOR_PORT (in module [darc.proxy.tor](#)), 44
 darc.proxy.tor.TOR_REQUESTS_PROXY (in module [darc.proxy.tor](#)), 44
 darc.proxy.tor.TOR_RETRY (in module [darc.proxy.tor](#)), 45
 darc.proxy.tor.TOR_SELENIUM_PROXY (in module [darc.proxy.tor](#)), 44
 darc.proxy.zeronet module, 45
 darc.proxy.zeronet._MNG_ZERONET (in module [darc.proxy.zeronet](#)), 47
 darc.proxy.zeronet._ZERONET_ARGS (in module [darc.proxy.zeronet](#)), 47
 darc.proxy.zeronet._ZERONET_BS_FLAG (in module [darc.proxy.zeronet](#)), 47
 darc.proxy.zeronet._ZERONET_PROC (in module [darc.proxy.zeronet](#)), 47
 darc.proxy.zeronet.BS_WAIT (in module

darc.proxy.zeronet), 46
darc.proxy.zeronet.ZERONET_ARGS (in module *darc.proxy.zeronet*), 47
darc.proxy.zeronet.ZERONET_PATH (in module *darc.proxy.zeronet*), 47
darc.proxy.zeronet.ZERONET_PORT (in module *darc.proxy.zeronet*), 46
darc.proxy.zeronet.ZERONET_RETRY (in module *darc.proxy.zeronet*), 46
darc.requests
 module, 29
darc.save
 module, 10
darc.save._SAVE_LOCK (in module *darc.save*), 13
darc.selenium
 module, 30
darc.sites
 module, 48
darc.sites.bitcoin
 module, 50
darc.sites.data
 module, 50
darc.sites.default
 module, 48
darc.sites.ed2k
 module, 51
darc.sites.irc
 module, 51
darc.sites.magnet
 module, 52
darc.sites.mail
 module, 52
darc.sites.script
 module, 53
darc.sites.SITEMAP (in module *darc.sites*), 54
darc.sites.tel
 module, 53
darc.submit
 module, 21
darc.submit.API_NEW_HOST (in module *darc.submit*), 29
darc.submit.API_REQUESTS (in module *darc.submit*), 29
darc.submit.API_RETRY (in module *darc.submit*), 28
darc.submit.API_SELENIUM (in module *darc.submit*), 29
darc.submit.PATH_API (in module *darc.submit*), 28
darc.submit.SAVE_DB (in module *darc.submit*), 28
DARC_BULK_SIZE, 20
DARC_CHECK, 56
DARC_CHECK_CONTENT_TYPE, 56
DARC_CPU, 56
DARC_DEBUG, 56
DARC_FORCE, 56
DARC_LOCK_TIMEOUT, 20
DARC_MAX_POOL, 21
DARC_MULTIPROCESSING, 56
DARC_MULTITHREADING, 56
DARC_REBOOT, 55, 84
DARC_REDIS_LOCK, 21
DARC_RETRY, 21
DARC_SAVE, 124
DARC_SAVE_REQUESTS, 124
DARC_SAVE_SELENIUM, 124
DARC_USER, 57
DARC_VERBOSE, 56
DARC_WAIT, 58
database (*darc.model.abc.BaseMeta* attribute), 69
database (*darc.model.abc.BaseMetaWeb* attribute), 69
DatabaseOperationFailed, 60
db_value () (*darc.model.utils.IPField* method), 70
db_value () (*darc.model.utils.JSONField* method), 71
db_value () (*darc.model.utils.PickleField* method), 72
default_user_agent () (in module *darc.requests*), 29
discovery (*darc.model.web.hostname.HostnameModel* attribute), 64
discovery (*darc.model.web.url.URLModel* attribute), 65
document (*darc.model.web.hosts.HostsModel* attribute), 67
document (*darc.model.web.requests.RequestsHistoryModel* attribute), 67
document (*darc.model.web.requests.RequestsModel* attribute), 68
document (*darc.model.web.robots.RobotsModel* attribute), 66
document (*darc.model.web.selenium.SeleniumModel* attribute), 69
document (*darc.model.web.sitemap.SitemapModel* attribute), 66
DoesNotExist (*darc.model.abc.BaseModel* attribute), 70
DoesNotExist (*darc.model.abc.BaseModelWeb* attribute), 70
DoesNotExist (*darc.model.tasks.hostname.HostnameQueueModel* attribute), 62
DoesNotExist (*darc.model.tasks.requests.RequestsQueueModel* attribute), 62
DoesNotExist (*darc.model.tasks.selenium.SeleniumQueueModel* attribute), 63
DoesNotExist (*darc.model.web.hostname.HostnameModel* attribute), 64
DoesNotExist (*darc.model.web.hosts.HostsModel* attribute), 67
DoesNotExist (*darc.model.web.requests.RequestsHistoryModel*

attribute), 67
 DoesNotExist (*darc.model.web.requests.RequestsModel attribute*), 68
 DoesNotExist (*darc.model.web.robots.RobotsModel attribute*), 66
 DoesNotExist (*darc.model.web.selenium.SeleniumModel attribute*), 69
 DoesNotExist (*darc.model.web.sitemap.SitemapModel attribute*), 66
 DoesNotExist (*darc.model.web.url.URLModel attribute*), 65
 drop_hostname() (*in module darc.db*), 18
 drop_requests() (*in module darc.db*), 18
 drop_selenium() (*in module darc.db*), 18

E

environment variable
 API_NEW_HOST, 29, 127
 API_REQUESTS, 29, 127
 API_RETRY, 29, 127
 API_SELENIUM, 29, 127
 DARC_BULK_SIZE, 20, 123
 DARC_CHECK, 56, 121
 DARC_CHECK_CONTENT_TYPE, 56, 121
 DARC_CPU, 56, 122
 DARC_DEBUG, 56, 121
 DARC_FORCE, 56, 121
 DARC_FREENET, 131
 DARC_I2P, 129
 DARC_LOCK_TIMEOUT, 20
 DARC_MAX_POOL, 21, 123
 DARC_MULTIPROCESSING, 56, 122
 DARC_MULTITHREADING, 56, 122
 DARC_REBOOT, 55, 84, 121
 DARC_REDIS_LOCK, 21
 DARC_RETRY, 21
 DARC_SAVE, 124
 DARC_SAVE_REQUESTS, 124
 DARC_SAVE_SELENIUM, 124
 DARC_TOR, 128
 DARC_USER, 57, 122
 DARC_VERBOSE, 56, 121
 DARC_WAIT, 58, 124
 DARC_ZERONET, 130
 DB_URL, 123
 FREENET_ARGS, 131
 FREENET_PATH, 131
 FREENET_PORT, 131
 FREENET_RETRY, 131
 FREENET_WAIT, 131
 I2P_ARGS, 129
 I2P_PORT, 129
 I2P_RETRY, 129
 I2P_WAIT, 129

LINK_BLACK_LIST, 59, 125
 LINK_FALLBACK, 59, 125
 LINK_WHITE_LIST, 59, 125
 LOCK_TIMEOUT, 123
 MIME_BLACK_LIST, 59, 126
 MIME_FALLBACK, 59, 126
 MIME_WHITE_LIST, 59, 126
 PATH_DATA, 57, 122
 PROXY_BLACK_LIST, 60, 126
 PROXY_FALLBACK, 60, 126
 PROXY_WHITE_LIST, 59, 126
 REDIS_LOCK, 123
 REDIS_URL, 57, 122
 RETRY_INTERVAL, 124
 SAVE_DB, 28, 127
 SE_WAIT, 58, 125
 TIME_CACHE, 58, 125
 TOR_CFG, 128
 TOR_CTRL, 128
 TOR_PASS, 128
 TOR_PORT, 128
 TOR_RETRY, 128
 TOR_WAIT, 128
 ZERONET_ARGS, 130
 ZERONET_PATH, 130
 ZERONET_PORT, 130
 ZERONET_RETRY, 130
 ZERONET_WAIT, 130

extract_links() (*in module darc.parse*), 9

F

fetch_hosts() (*in module darc.proxy.i2p*), 35
 fetch_sitemap() (*in module darc.proxy.null*), 40
 FREENET (*darc.model.utils.Proxy attribute*), 72
 freenet_bootstrap() (*in module darc.proxy.freenet*), 34
 FreenetBootstrapFailed, 60

G

get_capabilities() (*in module darc.selenium*), 30
 get_content_type() (*in module darc.parse*), 9
 get_hosts() (*in module darc.proxy.i2p*), 36
 get_hosts() (*in module darc.submit*), 21
 get_lock() (*in module darc.const*), 55
 get_metadata() (*in module darc.submit*), 22
 get_options() (*in module darc.selenium*), 31
 get_robots() (*in module darc.submit*), 22
 get_sitemap() (*in module darc.proxy.null*), 40
 get_sitemaps() (*in module darc.submit*), 22
 getpid() (*in module darc.const*), 55

H

hash (*darc.model.tasks.requests.RequestsQueueModel attribute*), 63

- `hash` (*darc.model.tasks.selenium.SeleniumQueueModel attribute*), 63
 - `hash` (*darc.model.web.url.URLModel attribute*), 65
 - `have_hostname()` (in module *darc.db*), 18
 - `have_hosts()` (in module *darc.proxy.i2p*), 36
 - `have_robots()` (in module *darc.proxy.null*), 40
 - `have_sitemap()` (in module *darc.proxy.null*), 40
 - `history` (*darc.model.web.requests.RequestsModel attribute*), 68
 - `host` (*darc.link.Link attribute*), 5
 - `host` (*darc.model.web.hosts.HostsModel attribute*), 67
 - `host` (*darc.model.web.robots.RobotsModel attribute*), 66
 - `host` (*darc.model.web.sitemap.SitemapModel attribute*), 66
 - `host_id` (*darc.model.web.hosts.HostsModel attribute*), 67
 - `host_id` (*darc.model.web.robots.RobotsModel attribute*), 66
 - `host_id` (*darc.model.web.sitemap.SitemapModel attribute*), 66
 - `hostname` (*darc.model.tasks.hostname.HostnameQueueModel attribute*), 62
 - `hostname` (*darc.model.web.hostname.HostnameModel attribute*), 64
 - `hostname` (*darc.model.web.url.URLModel attribute*), 65
 - `hostname_id` (*darc.model.web.url.URLModel attribute*), 65
 - `HostnameModel` (class in *darc.model.web.hostname*), 64
 - `HostnameQueueModel` (class in *darc.model.tasks.hostname*), 62
 - `hosts` (*darc.model.web.hostname.HostnameModel attribute*), 64
 - `HostsModel` (class in *darc.model.web.hosts*), 67
- I**
- `I2P` (*darc.model.utils.Proxy attribute*), 72
 - `i2p_bootstrap()` (in module *darc.proxy.i2p*), 36
 - `i2p_driver()` (in module *darc.selenium*), 31
 - `i2p_session()` (in module *darc.requests*), 29
 - `I2PBootstrapFailed`, 61
 - `id` (*darc.model.abc.BaseModel attribute*), 70
 - `id` (*darc.model.abc.BaseModelWeb attribute*), 70
 - `id` (*darc.model.tasks.hostname.HostnameQueueModel attribute*), 62
 - `id` (*darc.model.tasks.requests.RequestsQueueModel attribute*), 63
 - `id` (*darc.model.tasks.selenium.SeleniumQueueModel attribute*), 63
 - `id` (*darc.model.web.hostname.HostnameModel attribute*), 64
 - `id` (*darc.model.web.hosts.HostsModel attribute*), 67
 - `id` (*darc.model.web.requests.RequestsHistoryModel attribute*), 67
 - `id` (*darc.model.web.requests.RequestsModel attribute*), 68
 - `id` (*darc.model.web.robots.RobotsModel attribute*), 66
 - `id` (*darc.model.web.selenium.SeleniumModel attribute*), 69
 - `id` (*darc.model.web.sitemap.SitemapModel attribute*), 66
 - `id` (*darc.model.web.url.URLModel attribute*), 65
 - `index` (*darc.model.web.requests.RequestsHistoryModel attribute*), 67
 - `IntEnumField` (class in *darc.model.utils*), 71
 - `IPField` (class in *darc.model.utils*), 70
 - `is_html` (*darc.model.web.requests.RequestsModel attribute*), 68
- J**
- `JSONField` (class in *darc.model.utils*), 71
- L**
- `last_seen` (*darc.model.web.hostname.HostnameModel attribute*), 64
 - `last_seen` (*darc.model.web.url.URLModel attribute*), 65
 - `Link` (class in *darc.link*), 5
 - `link` (*darc.model.tasks.requests.RequestsQueueModel attribute*), 63
 - `link` (*darc.model.tasks.selenium.SeleniumQueueModel attribute*), 63
 - `LINK_BLACK_LIST`, 59
 - `LINK_FALLBACK`, 59
 - `LINK_WHITE_LIST`, 59
 - `LinkNoReturn`, 61
 - `load_requests()` (in module *darc.db*), 18
 - `load_selenium()` (in module *darc.db*), 19
 - `loader()` (in module *darc.crawl*), 4
 - `loader()` (in module *darc.sites.bitcoin*), 50
 - `loader()` (in module *darc.sites.data*), 50
 - `loader()` (in module *darc.sites.default*), 48
 - `loader()` (in module *darc.sites.ed2k*), 51
 - `loader()` (in module *darc.sites.irc*), 51
 - `loader()` (in module *darc.sites.magnet*), 52
 - `loader()` (in module *darc.sites.mail*), 52
 - `loader()` (in module *darc.sites.script*), 53
 - `loader()` (in module *darc.sites.tel*), 53
 - `loader_hook()` (in module *darc.sites*), 54
 - `LockWarning`, 61
- M**
- `match_host()` (in module *darc.parse*), 9
 - `match_mime()` (in module *darc.parse*), 10
 - `match_proxy()` (in module *darc.parse*), 10
 - `Meta` (*darc.model.abc.BaseModel attribute*), 70
 - `Meta` (*darc.model.abc.BaseModelWeb attribute*), 70

method (*darc.model.web.requests.RequestsHistoryModel attribute*), 67

method (*darc.model.web.requests.RequestsModel attribute*), 68

MIME_BLACK_LIST, 59

MIME_FALLBACK, 59

mime_type (*darc.model.web.requests.RequestsModel attribute*), 68

MIME_WHITE_LIST, 59

model (*darc.model.web.requests.RequestsHistoryModel attribute*), 67

model_id (*darc.model.web.requests.RequestsHistoryModel attribute*), 67

module

- darc, 1
- darc.crawl, 3
- darc.db, 13
- darc.error, 60
- darc.link, 5
- darc.model, 62
- darc.model.abc, 69
- darc.model.tasks, 62
- darc.model.tasks.hostname, 62
- darc.model.tasks.requests, 62
- darc.model.tasks.selenium, 63
- darc.model.utils, 70
- darc.model.web, 63
- darc.model.web.hostname, 63
- darc.model.web.hosts, 66
- darc.model.web.requests, 67
- darc.model.web.robots, 65
- darc.model.web.selenium, 69
- darc.model.web.sitemap, 66
- darc.model.web.url, 64
- darc.parse, 8
- darc.process, 1
- darc.proxy, 32
- darc.proxy.bitcoin, 32
- darc.proxy.data, 33
- darc.proxy.ed2k, 33
- darc.proxy.freenet, 33
- darc.proxy.i2p, 35
- darc.proxy.irc, 38
- darc.proxy.magnet, 39
- darc.proxy.mail, 39
- darc.proxy.null, 40
- darc.proxy.script, 42
- darc.proxy.tel, 43
- darc.proxy.tor, 43
- darc.proxy.zeronet, 45
- darc.requests, 29
- darc.save, 10
- darc.selenium, 30
- darc.sites, 48

- darc.sites.bitcoin, 50
- darc.sites.data, 50
- darc.sites.default, 48
- darc.sites.ed2k, 51
- darc.sites.irc, 51
- darc.sites.magnet, 52
- darc.sites.mail, 52
- darc.sites.script, 53
- darc.sites.tel, 53
- darc.submit, 21

N

name (*darc.link.Link attribute*), 5

NULL (*darc.model.utils.Proxy attribute*), 72

null_driver() (*in module darc.selenium*), 31

null_session() (*in module darc.requests*), 30

P

parse_link() (*in module darc.link*), 5

PATH_DATA, 57

PickleField (*class in darc.model.utils*), 72

print_bootstrap_lines() (*in module darc.proxy.tor*), 44

process() (*in module darc.process*), 1

process_crawler() (*in module darc.process*), 3

process_loader() (*in module darc.process*), 3

Proxy (*class in darc.model.utils*), 72

proxy (*darc.link.Link attribute*), 5

proxy (*darc.model.web.hostname.HostnameModel attribute*), 64

proxy (*darc.model.web.url.URLModel attribute*), 65

PROXY_BLACK_LIST, 60

PROXY_FALLBACK, 60

PROXY_WHITE_LIST, 59

python_value() (*darc.model.utils.IntEnumField method*), 71

python_value() (*darc.model.utils.IPField method*), 71

python_value() (*darc.model.utils.JSONField method*), 71

python_value() (*darc.model.utils.PickleField method*), 72

Q

quote() (*in module darc.link*), 7

R

read_hosts() (*in module darc.proxy.i2p*), 36

read_robots() (*in module darc.proxy.null*), 41

read_sitemap() (*in module darc.proxy.null*), 41

reason (*darc.model.web.requests.RequestsHistoryModel attribute*), 68

reason (*darc.model.web.requests.RequestsModel attribute*), 68

REDIS_URL, 57
 RedisCommandFailed, 61
 render_error() (in module darc.error), 61
 renew_tor_session() (in module darc.proxy.tor), 44
 request (darc.model.web.requests.RequestsHistoryModel attribute), 68
 request (darc.model.web.requests.RequestsModel attribute), 68
 request_driver() (in module darc.selenium), 32
 request_session() (in module darc.requests), 30
 requests (darc.model.web.url.URLModel attribute), 65
 RequestsHistoryModel (class in darc.model.web.requests), 67
 RequestsModel (class in darc.model.web.requests), 68
 RequestsQueueModel (class in darc.model.tasks.requests), 62
 response (darc.model.web.requests.RequestsHistoryModel attribute), 68
 response (darc.model.web.requests.RequestsModel attribute), 68
 robots (darc.model.web.hostname.HostnameModel attribute), 64
 RobotsModel (class in darc.model.web.robots), 66

S

sanitise() (in module darc.save), 11
 save_bitcoin() (in module darc.proxy.bitcoin), 32
 save_data() (in module darc.proxy.data), 33
 SAVE_DB, 28
 save_ed2k() (in module darc.proxy.ed2k), 33
 save_headers() (in module darc.save), 11
 save_hosts() (in module darc.proxy.i2p), 37
 save_invalid() (in module darc.proxy.null), 41
 save_irc() (in module darc.proxy.irc), 38
 save_link() (in module darc.save), 12
 save_magnet() (in module darc.proxy.magnet), 39
 save_mail() (in module darc.proxy.mail), 39
 save_requests() (in module darc.db), 19
 save_robots() (in module darc.proxy.null), 41
 save_script() (in module darc.proxy.script), 42
 save_selenium() (in module darc.db), 20
 save_sitemap() (in module darc.proxy.null), 42
 save_submit() (in module darc.submit), 23
 save_tel() (in module darc.proxy.tel), 43
 screenshot (darc.model.web.selenium.SeleniumModel attribute), 69
 SE_WAIT, 58
 selenium (darc.model.web.url.URLModel attribute), 65
 SeleniumModel (class in darc.model.web.selenium), 69
 SeleniumQueueModel (class in darc.model.tasks.selenium), 63
 session (darc.model.web.requests.RequestsModel attribute), 68
 since (darc.model.web.hostname.HostnameModel attribute), 64
 since (darc.model.web.url.URLModel attribute), 65
 SitemapModel (class in darc.model.web.sitemap), 66
 sitemaps (darc.model.web.hostname.HostnameModel attribute), 64
 SiteNotFoundWarning, 61
 status_code (darc.model.web.requests.RequestsHistoryModel attribute), 68
 status_code (darc.model.web.requests.RequestsModel attribute), 68
 submit() (in module darc.submit), 23
 submit_new_host() (in module darc.submit), 23
 submit_requests() (in module darc.submit), 25
 submit_selenium() (in module darc.submit), 27

T

table_function() (darc.model.abc.BaseMeta method), 69
 table_function() (in module darc.model.utils), 72
 text (darc.model.tasks.requests.RequestsQueueModel attribute), 63
 text (darc.model.tasks.selenium.SeleniumQueueModel attribute), 63
 TIME_CACHE, 58
 timestamp (darc.model.tasks.hostname.HostnameQueueModel attribute), 62
 timestamp (darc.model.tasks.requests.RequestsQueueModel attribute), 63
 timestamp (darc.model.tasks.selenium.SeleniumQueueModel attribute), 63
 timestamp (darc.model.web.hosts.HostsModel attribute), 67
 timestamp (darc.model.web.requests.RequestsHistoryModel attribute), 68
 timestamp (darc.model.web.requests.RequestsModel attribute), 68
 timestamp (darc.model.web.robots.RobotsModel attribute), 66
 timestamp (darc.model.web.selenium.SeleniumModel attribute), 69
 timestamp (darc.model.web.sitemap.SitemapModel attribute), 66
 to_dict() (darc.model.abc.BaseModel method), 70
 TOR (darc.model.utils.Proxy attribute), 72
 TOR2WEB (darc.model.utils.Proxy attribute), 72
 tor_bootstrap() (in module darc.proxy.tor), 44
 tor_driver() (in module darc.selenium), 32
 tor_session() (in module darc.requests), 30
 TorBootstrapFailed, 61

TorRenewFailed, 61

U

unquote() (in module *darc.link*), 7

UnsupportedLink, 61

UnsupportedPlatform, 61

UnsupportedProxy, 61

url (*darc.link.Link* attribute), 5

url (*darc.model.web.requests.RequestsHistoryModel* attribute), 68

url (*darc.model.web.requests.RequestsModel* attribute), 68

url (*darc.model.web.selenium.SeleniumModel* attribute), 69

url (*darc.model.web.url.URLModel* attribute), 65

url_id (*darc.model.web.requests.RequestsModel* attribute), 69

url_id (*darc.model.web.selenium.SeleniumModel* attribute), 69

url_parse (*darc.link.Link* attribute), 5

urljoin() (in module *darc.link*), 7

URLModel (class in *darc.model.web.url*), 65

urlparse() (in module *darc.link*), 8

urls (*darc.model.web.hostname.HostnameModel* attribute), 64

Z

ZERONET (*darc.model.utils.Proxy* attribute), 72

zeronet_bootstrap() (in module *darc.proxy.zeronet*), 46

ZeroNetBootstrapFailed, 61